

Netfilter

Rusty Russell

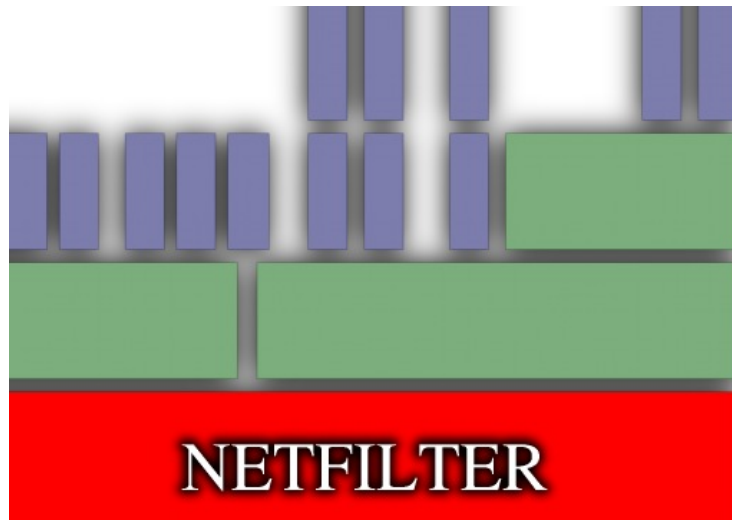
1. Netfilter Architecture

1.1 Why?

2.2: portforward, redirection, masquerading, filtering:

- Related, but close to the stack (fragile), and independent from each other.
- Needed an in-kernel packet snarfing framework: no more hard-coded kernel **#ifdef** hacks.

Netfilter is the framework.

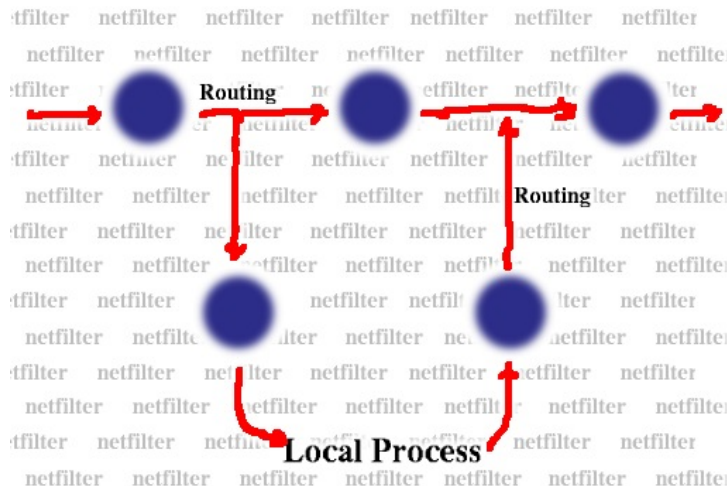


1.2 The Hooks

Parts of the kernel can register with netfilter to see packets at various points in the stack (similar to old firewall.h hooks).

IPv4: PRE_ROUTING, LOCAL_IN, FORWARD, LOCAL_OUT, POST_ROUTING.

Each hook can alter packets, return NF_DROP, NF_ACCEPT, NF_QUEUE, NF_REPEAT or NF_STOLEN.



Netfilter

Rusty Russell

1.3 Other Protocols

Hooks have been inserted in the IPv6 and DecNET stacks as well.

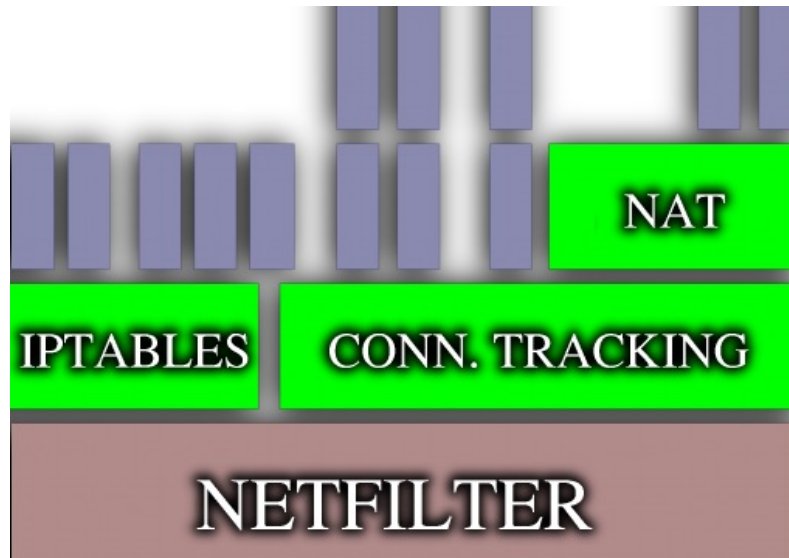
IPv6: PRE_ROUTING, LOCAL_IN, FORWARD, LOCAL_OUT, POST_ROUTING.

DecNET: PRE_ROUTING, LOCAL_IN, FORWARD, LOCAL_OUT, POST_ROUTING, HELLO, ROUTE

1.4 On Top of Netfilter

Currently, four major subsystems exist on top of netfilter:

- The backwards-compatibility ipchains & ipfwadm +masq/redir modules.
- The `iptables' packet classification system.
- The connection-tracking system.
- The NAT system.



1.5 Summary

Linux 2.4's IP stack now has sufficient hooks to cleanly extend functionality for filtering, NAT and random hacks.

2. iptables

2.1 What It Is

Kernel: Lists of packet matching rules similar to ipchains/ipfwadm

Userspace: program `iptables' and library `libiptc' which access tables

Simple functionality (IP header matching) built in

Supports multiple tables

2.2 What We Use It For

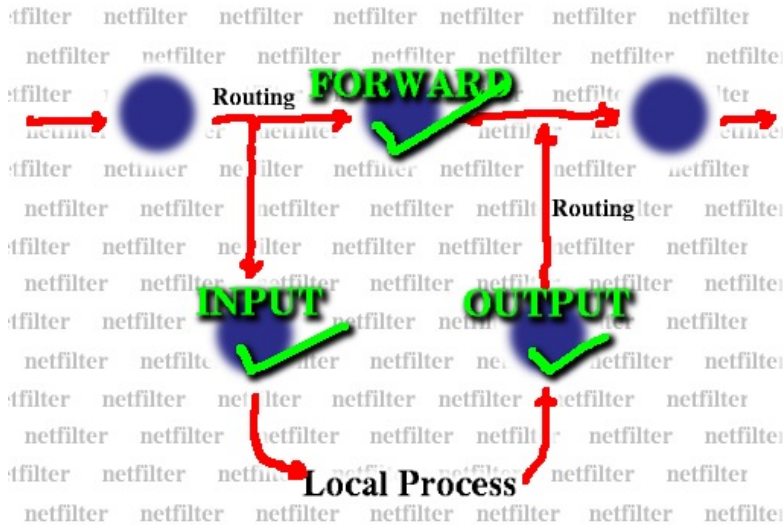
Netfilter

Rusty Russell

Currently there are three tables: **filter**, **nat**, **mangle**.

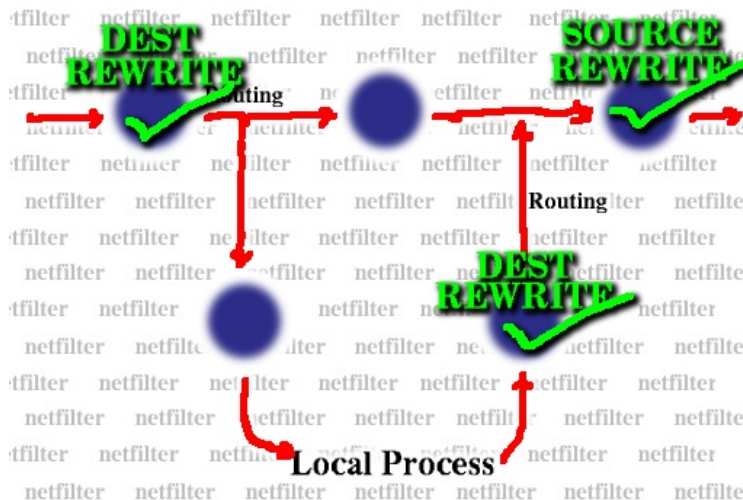
filter table used by packet filtering system

- hooks in at **LOCAL_IN** (INPUT), **FORWARD**, **LOCAL_OUT** (OUTPUT)
- iptable_filter hooks in at those points and passes all packets to the table
- default table operated on by iptables program



nat table used to control nat

- hooks in at **LOCAL_OUT** (OUTPUT), **PREROUTING**, **POSTROUTING**
- iptable_nat hooks in and passes packets whose connections have not seen NAT table to the table



mangle table used for special effects

- hooks in at **LOCAL_OUT** (OUTPUT), **PREROUTING**
- iptable_mangle hooks in and passes all packets to the table

2.3 User-visible improvements over ipchains

Netfilter

Rusty Russell

- -i and -o flags
- DROP, not DENY
- Zeroing single chains while listing them works.
- Zeroing built-in chains also clears policy counters.
- Listing chains gives you the counters as an atomic snapshot.
- Chain names can be up to 31 characters.

2.4 Match Extensions

We can write new **match** criteria for rules
Each rule can have 0 or more of these extensions attached.
Consists of two parts:

- Shared library to provide new cmd-line options to iptables
- Module to provide actual packet matching code

tcp, udp, icmp
limit, mac, mark, multiport, owner, state, tos, unclean
Unofficial

- pool, random, length

2.5 Target Extensions

We can write new **targets** for rules
Consists of two parts:

- Shared library to provide new cmd-line options to iptables
- Module to provide actual packet action

LOG, MIRROR, REJECT
MARK, TOS
SNAT, DNAT, MASQUERADE, REDIRECT
Unofficial

- POOL, TCPMSS

2.6 Writing an Extension

Documented clearly in the netfilter-hacking HOWTO (English, German)
Really simple to do
Example here is writing a very simple REJECT extension

- Only does ICMP rejects
- Doesn't care which table it is placed in
- Doesn't have userspace component: takes no options

```
#include <linux/module.h>
#include <linux/skbuff.h>
#include <net/icmp.h>
```

Netfilter Rusty Russell

```
#include <net/route.h>
#include "packet-filter/kernel/ip_tables.h"
EXPORT_NO_SYMBOLS;

static unsigned int reject(struct sk_buff **pskb,
                          unsigned int hooknum,
                          const struct net_device *in,
                          const struct net_device *out,
                          const void *targinfo)
{
    icmp_send(*pskb, ICMP_DEST_UNREACH, ICMP_PORT_UNREACH,
              0);
    return NF_DROP;
}

static int check(const char *tablename,
                void *targinfo,
                unsigned int targinfosize,
                unsigned int hook_mask)
{
    return (targinfosize == 0
            && !(hook_mask & ~((1 << NF_IP_LOCAL_IN)
                               | (1 << NF_IP_FORWARD)
                               | (1 << NF_IP_LOCAL_OUT))));
}

static struct ipt_target ipt_reject_reg
= { { NULL, NULL }, "REJECT", NETFILTER_VERSION,
    reject, check, THIS_MODULE };

int __init init(void)
{
    if (ipt_register_target(&ipt_reject_reg))
        return -EINVAL;
    return 0;
}

void __exit cleanup(void)
{
    ipt_unregister_target(&ipt_reject_reg);
}

module_init(init);
module_exit(cleanup);
```

2.7 IPv6

Philip Blundell ported to IPv6
Userspace `ip6tables' shares almost all code, using macros.
Kernel space shares almost no code.
Only ip6table_filter supported.

2.8 Summary

Linux 2.4 has a reasonably nice, familiar, general, extensible packet selection framework.

Netfilter

Rusty Russell

Might even last two kernel generations!
Customisation seems to be popular.

3. Packet Filtering

3.1 A Simple Scenario

Single machine with no forwarding: PPP interface
Only ident requests to come in from outside

3.2 A Simple Solution

```
iptables -A INPUT -i lo -j ACCEPT
iptables -P INPUT DROP
```

```
iptables -N LOGDROP
iptables -A LOGDROP --m limit --limit 5/hour -j LOG
iptables -A LOGDROP -j DROP
```

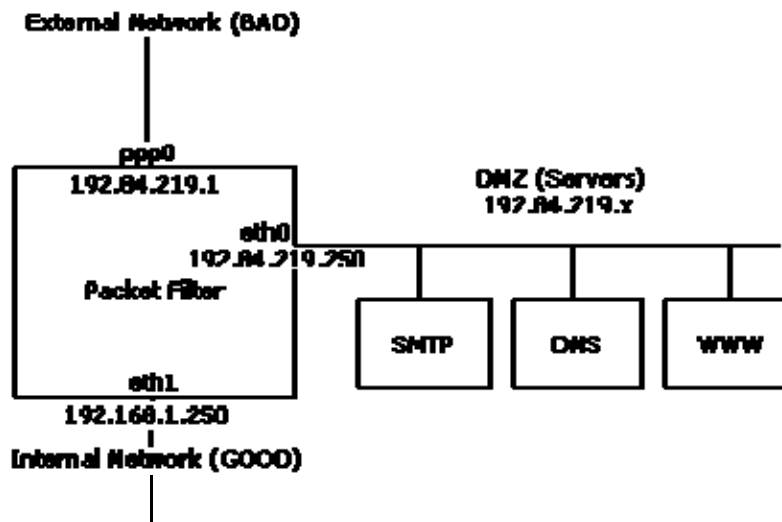
```
iptables -N ppp-incoming
iptables -A ppp-incoming -p ! tcp -f -j LOGDROP
iptables -A ppp-incoming -p tcp --dport ident -j ACCEPT
iptables -A ppp-incoming -p tcp ! --syn -j ACCEPT
iptables -A ppp-incoming -p udp --dport 53 -j ACCEPT
iptables -A ppp-incoming -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A ppp-incoming -p icmp --icmp-type pong -j ACCEPT
iptables -A ppp-incoming -j LOGDROP
```

In ip-up script:

```
iptables -A INPUT -i $1 -j ppp-incoming
```

3.3 A More Realistic Scenario

Internal Network, DMZ



Netfilter

Rusty Russell

We have the following requirements:

Packet Filter box:

- PING any network
- TRACEROUTE any network
- Access DNS

DMZ:

Mail server

- SMTP to external
- Accept SMTP from internal and external
- Accept POP-3 from internal

Name server

- Send DNS to external
- Accept DNS from internal, external and packet filter box

Web server

- Accept HTTP from internal and external
- Rsync access from internal

Internal:

- Allow WWW, ftp, traceroute, ssh to external
- Allow SMTP to Mail server
- Allow POP-3 to Mail server
- Allow DNS to Name server
- Allow rsync to Web server
- Allow WWW to Web server
- Allow ping to packet filter box

3.4 A Solution

```
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do echo 1 > $f; done
```

```
iptables -A INPUT -i ! lo -j DROP
iptables -A OUTPUT -i ! lo -j DROP
iptables -A FORWARD -j DROP
```

```
ifconfig eth0 192.84.219.0 netmask 255.255.255.0
ifconfig eth1 192.168.1.0 netmask 255.255.255.0
pppd
```

```
route add default ppp0
```

```
iptables -N internal-dmz
iptables -N external-dmz
iptables -N internal-external
iptables -N dmz-internal
iptables -N dmz-external
```

Netfilter

Rusty Russell

```
iptables -N external-internal
iptables -N icmp-accept
iptables -N NEVER
iptables -N LOGDROP
```

```
iptables -A NEVER -j LOG --log-level alert --log-prefix "filter ERROR: "
iptables -A NEVER -j DROP
```

```
iptables -A LOGDROP -m limit -j LOG --log-prefix "filter: "
iptables -A LOGDROP -j DROP
```

```
iptables -A FORWARD -i eth1 -o eth0 -j internal-dmz
iptables -A FORWARD -i eth1 -o ppp0 -j internal-external
iptables -A FORWARD -i eth0 -o ppp0 -j dmz-external
iptables -A FORWARD -i eth0 -o eth1 -j dmz-internal
iptables -A FORWARD -i ppp0 -o eth0 -j external-dmz
iptables -A FORWARD -i ppp0 -o eth1 -j external-internal
iptables -A FORWARD -j NEVER
```

Define the icmp-accept chain:

```
iptables -A icmp-accept -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A icmp-accept -p icmp --icmp-type source-quench -j ACCEPT
iptables -A icmp-accept -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A icmp-accept -p icmp --icmp-type parameter-problem -j ACCEPT
```

```
iptables -A internal-dmz -p tcp -d $MAILSERVER --dport smtp -j ACCEPT
iptables -A internal-dmz -p tcp -d $MAILSERVER --dport pop-3 -j ACCEPT
iptables -A internal-dmz -p udp -d $NAMESERVER --dport domain -j ACCEPT
iptables -A internal-dmz -p tcp -d $NAMESERVER --dport domain -j ACCEPT
iptables -A internal-dmz -p tcp -d $WEBSERVER --dport www -j ACCEPT
iptables -A internal-dmz -p tcp -d $WEBSERVER --dport rsync -j ACCEPT
iptables -A internal-dmz -p icmp -j icmp-accept
iptables -A internal-dmz -j LOGDROP
```

```
iptables -A external-dmz -p tcp -d $MAILSERVER --dport smtp -j ACCEPT
iptables -A external-dmz -p udp -d $NAMESERVER --dport domain -j ACCEPT
iptables -A external-dmz -p tcp -d $NAMESERVER --dport domain -j ACCEPT
iptables -A external-dmz -p tcp -d $WEBSERVER --dport www -j ACCEPT
iptables -A external-dmz -p icmp -j icmp-accept
iptables -A external-dmz -j DROP
```

```
iptables -A internal-external -p tcp --dport www -j ACCEPT
iptables -A internal-external -p tcp --dport ssh -j ACCEPT
iptables -A internal-external -p udp --dport 33434:33500 -j ACCEPT
iptables -A internal-external -p tcp --dport ftp -j ACCEPT
iptables -A internal-external -p tcp --dport 1024:65535 -j ACCEPT
iptables -A internal-external -p icmp --icmp-type ping -j ACCEPT
iptables -A internal-external -j LOG
iptables -A internal-external -j REJECT
```

```
iptables -A dmz-internal -p tcp ! --syn -s $MAILSERVER smtp -j ACCEPT
iptables -A dmz-internal -p udp -s $NAMESERVER domain -j ACCEPT
iptables -A dmz-internal -p tcp ! --syn -s $NAMESERVER domain -j ACCEPT
iptables -A dmz-internal -p tcp ! --syn -s $WEBSERVER www -j ACCEPT
iptables -A dmz-internal -p tcp ! --syn -s $WEBSERVER rsync -j ACCEPT
iptables -A dmz-internal -p icmp -j icmp-accept
iptables -A dmz-internal -j NEVER
```

Netfilter

Rusty Russell

```
iptables -A dmz-external -p tcp -s $MAILSERVER smtp -j ACCEPT
iptables -A dmz-external -p udp -s $NAMESERVER domain -j ACCEPT
iptables -A dmz-external -p tcp -s $NAMESERVER domain -j ACCEPT
iptables -A dmz-external -p tcp ! --syn -s $WEBSERVER www -j ACCEPT
iptables -A dmz-external -p icmp -j icmp-accept
iptables -A dmz-external -j NEVER

iptables -A external-internal -p tcp ! --syn --sport www -j ACCEPT
iptables -A external-internal -p tcp ! --syn --sport ssh -j ACCEPT
iptables -A external-internal -p tcp ! --syn --sport ftp -j ACCEPT
iptables -A external-internal -p tcp ! --syn --sport 1024:65535 -j ACCEPT
iptables -A external-internal -p icmp --icmp-type pong -j ACCEPT
iptables -A external-internal -j DROP

iptables -N external-if
iptables -N dmz-if
iptables -N internal-if

iptables -A INPUT -i ppp0 -j external-if
iptables -A INPUT -i eth0 -j dmz-if
iptables -A INPUT -i eth1 -j internal-if

iptables -A external-if -p icmp --icmp-type pong -j ACCEPT
iptables -A external-if -j icmp-accept
iptables -A external-if -j DROP

iptables -A dmz-if -p tcp ! --syn -s $NAMESERVER 53 -j ACCEPT
iptables -A dmz-if -p udp -s $NAMESERVER 53 -j ACCEPT
iptables -A dmz-if -p icmp --icmp-type pong -j ACCEPT
iptables -A dmz-if -j icmp-accept
iptables -A dmz-if -j NEVER

iptables -A internal-if -p icmp --icmp-type ping -j ACCEPT
iptables -A internal-if -p icmp --icmp-type pong -j ACCEPT
iptables -A internal-if -j icmp-accept
iptables -A internal-if -j LOGDROP

iptables -D 1 input
iptables -D 1 forward
iptables -D 1 output
```

3.5 Summary

Packet filtering this way is logical, but painful
Use routing's source verification
Define your allowed set thoroughly
Always limit external logging
Correct placement of LOG rules can help with problems

4. Connection Tracking

4.1 The Idea

We keep track of relationship of packets: `connections'

Netfilter

Rusty Russell

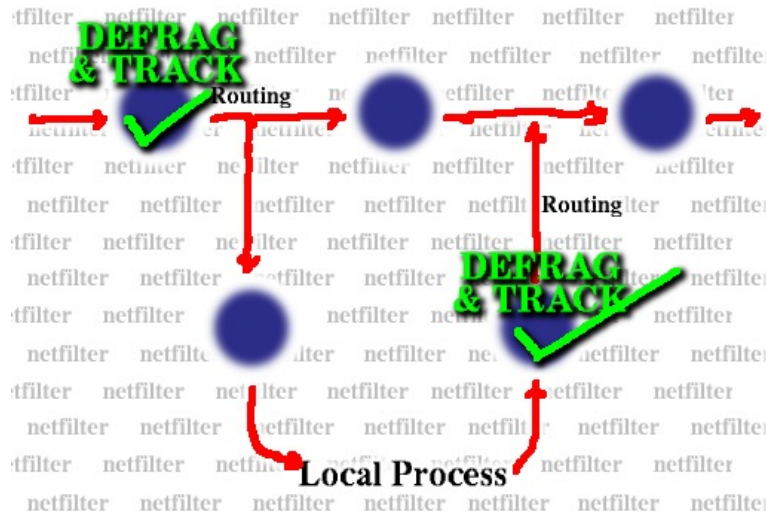
This applies even to 'connectionless' protocols, such as ping and UDP DNS. It will always be heuristic based: we cannot know what a reasonable delay is.

4.2 The Mechanism

When a packet comes in the **LOCAL_OUT** or **PRE_ROUTING** hooks, it indicates what connection it is part of, and how.

- Hooks in before packet filtering in **LOCAL_OUT**

Packets get defragmented



It classifies the packet as one of the following:

- NEW
- ESTABLISHED (either direction)
- RELATED (either direction)
- INVALID

Under stress, the code will timeout unreplied connections early. The ip_conntrack module has ip_conntrack_ftp extension (passive and active ftp)

4.3 With Packet Filtering

The 'state' packet matching allows you to match based on the conntrack results. This allows more reasonable UDP filtering, TCP fin-scanning filtering, FTP filtering eg. the above external->internal 6 rules becomes:

```
iptables -A external-internal -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A external-internal -j DROP
```

Effectively, you need only filter on initial (--state NEW) connections.

4.4 Summary

Netfilter

Rusty Russell

Packet filtering is markedly simplified by connection tracking.
Hopefully people won't screw up ICMP filtering now.

5. NAT

5.1 The Idea

To transparently make one set of IP addresses appear to be another set to external eyes
Manipulate the source/dest of outgoing packets, and the dest/source of incoming packets
I call them **DNAT** and **SNAT**.

5.2 Why Would I Want To Do That

To map an entire network onto one IP address, eg. dialup

- **SNAT** (masquerading)

To forward certain connections to servers with private addresses

- **DNAT** (port forwarding)

To distribute load over several machines

- **DNAT** (load balancing)

Special effects...

5.3 Dumb NAT vs Full NAT/NAPT

The Linux routing code has simple NAT capability

- Use `ip route nat` to control one-way mapping of IP addresses
- See IP Reference Manual in `iproute2` distribution

```
ip route add nat 192.203.80.192/26 via 193.233.7.64
ip route add prio 320 from 193.233.7.64/26 nat 192.203.80.192/26
```

On top of netfilter is built full Network Address Port Translation

- Requires connection tracking: keeps state
- Hence all packets must pass through this box
- Doesn't alter routing tables/advertisements

Advantage is that we are not limited to static N:N translations: we can compress address space.
We can also do more complex protocol manipulations (eg FTP).

5.4 Full NAT

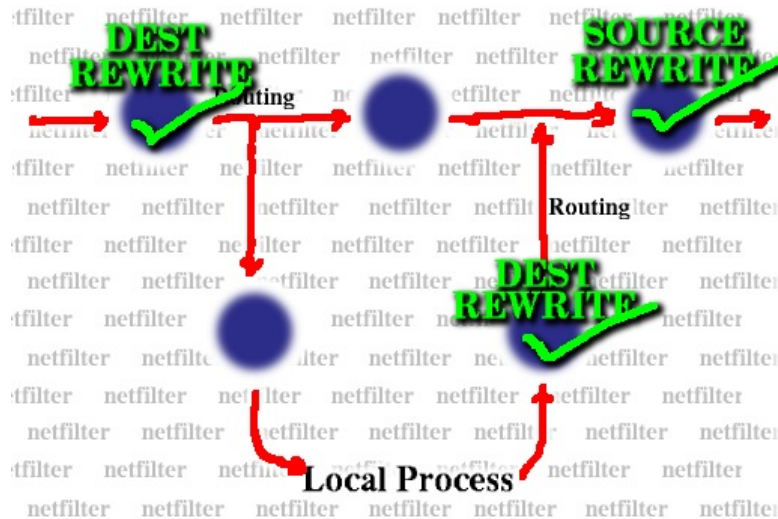
To do full, dynamic translation with port translation, connections must be tracked.

Netfilter

Rusty Russell

Hence the NAT on top of netfilter hooks in after connection tracking

- Packets which aren't tracked will be dropped!



5.5 With Packet Filtering

Packet filtering and NAT don't interfere with each other.

- DNAT happens before routing and packet filtering
- SNAT happens after routing and packet filtering

`Real' network addresses visible.

- Packet filtering never sees masquerading.
- Packet filtering always sees port forwarding.

To add NAT to above example:

```
insmod ip_nat.o && insmod ip_nat_ftp.o
iptables -t nat -A POSTROUTING -o ppp0 -j SNAT --to-source $PPP_ADDR
```

5.6 Common Traps

Port-forwarding onto the same network: do SNAT as well.
Only use `MASQUERADE' target for dynamically-addressed interfaces

- Forgets all connections when interface goes down.

Unknown protocols will only work for unique src/dst/proto combinations

5.7 Summary

NAT gives you new power to screw over your network
NAT breaks end-to-end

Netfilter

Rusty Russell

NAT doesn't work on some protocols without helpers
NAT is very popular for home networks w/ dynamic dialups

6. Netfilter Extensions

6.1 Netfilter Extensions

This is the lowest layer of hooking into the kernel stack
You can write your own kernel modules which hook directly into the IP stack (or others)

```
/* Rusty's Dumb netfilter hook example */
#include <linux/config.h>
#include <linux/module.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>

/* The work comes in here from netfilter.c. */
static unsigned int
stupid_hook(unsigned int hook,
            struct sk_buff **pskb,
            const struct net_device *indev,
            const struct net_device *outdev,
            int (*okfn)(struct sk_buff *))
{
    if ((*pskb)->len == 200)
        return NF_DROP;

    return NF_ACCEPT;
}

static struct nf_hook_ops stupid_ops
= { { NULL, NULL }, stupid_hook, PF_INET, NF_IP_POST_ROUTING, 0 };

static int __init init(void)
{
    return nf_register_hook(&stupid_ops);
}

static void __exit fini(void)
{
    nf_unregister_hook(&linuxmag_ops);
}

module_init(init);
module_exit(fini);
```

6.2 Extensions In Userspace (cont.)

You can use the QUEUE target to pass packets to an asynchronous handler
The most common handler is James Morris' ip_queue handler

```
/* Stupidity in userspace */
#include <sys/types.h>
#include <limits.h>
#include <net/if.h>
```

Netfilter

Rusty Russell

```
#include <netinet/ip.h>
#include <linux/netfilter_ipv4.h>

#include "libipq/libipq.h"

int main(void)
{
    struct ipq_handle *h;
    unsigned char packet[65536];

    if ((h = ipq_create_handle(0)) == NULL)
        exit(1);

    /* Tell the queue to send packets up to size BUFSIZE */
    rval = ipq_set_mode(h, IPQ_COPY_META, 0);
    if (rval < 0)
        goto die;

    while ((rval = ipq_read(h, packet, sizeof(packet), 0)) >= 0) {
        if (ipq_message_type(packet) == IPQM_PACKET) {
            ipq_packet_msg_t *msg = ipq_get_packet(packet);
            int verdict = NF_ACCEPT;

            if (m->data_len == 200)
                verdict = NF_DROP;
            ipq_set_verdict(h, m->packet_id,
                           verdict, m->data_len, m->payload);
        } else {
            fprintf(stderr, "Received error message %d\n",
                    ipq_get_msgerr(packet));
            break;
        }
    }
die:
    ipq_destroy_handle(h);
    exit(1);
}
```

6.3 iptables Extensions

We've seen the limited REJECT target extension.
Let's write a 'length' match extension (James Morris)

```
#ifndef __IPT_LENGTH_H
#define __IPT_LENGTH_H

struct ipt_length_info {
    u_int16_t length;
};
#endif /* __IPT_LENGTH_H */

/* Kernel module to match packet length. */
#include <linux/module.h>
#include <linux/skbuff.h>

#include <linux/netfilter_ipv4/ipt_length.h>
#include <linux/netfilter_ipv4/ip_tables.h>
```

Netfilter

Rusty Russell

```
static int
match(const struct sk_buff *skb,
      const struct net_device *in,
      const struct net_device *out,
      const void *matchinfo,
      int offset,
      const void *hdr,
      u_int16_t datalen,
      int *hotdrop)
{
    const struct ipt_length_info *info = matchinfo;

    return skb->len == info->length;
}

static int
checkentry(const char *tablename,
          const struct ipt_ip *ip,
          void *matchinfo,
          unsigned int matchsize,
          unsigned int hook_mask)
{
    if (matchsize != IPT_ALIGN(sizeof(struct ipt_length_info)))
        return 0;
    return 1;
}

static struct ipt_match length_match
= { { NULL, NULL }, "length", &match, &checkentry, NULL, THIS_MODULE };

static int __init init(void)
{
    return ipt_register_match(&length_match);
}

static void __exit fini(void)
{
    ipt_unregister_match(&length_match);
}

module_init(init);
module_exit(fini);
```

6.4 iptables Extensions (cont.)

We need a shared library to supplement command line:

```
/* Shared library add-on to iptables to add packet length matching support. */
#include <stdio.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <getopt.h>
#include <iptables.h>
#include <linux/netfilter_ipv4/ipt_length.h>

/* Function which prints out usage message. */
```

Netfilter

Rusty Russell

```
static void
help(void)
{
    printf("length v%s options:\n"
           "--length length    Match this packet length\n",
           NETFILTER_VERSION);
}

static struct option opts[] = {
    { "length", 1, 0, 'l' },
    {0}
};

/* Initialize the match. */
static void
init(struct ipt_entry_match *m, unsigned int *nfcache)
{
    *nfcache |= NFC_UNKNOWN;
}

/* Function which parses command options; returns true if it
   ate an option */
static int
parse(int c, char **argv, int invert, unsigned int *flags,
      const struct ipt_entry *entry,
      unsigned int *nfcache,
      struct ipt_entry_match **match)
{
    struct ipt_length_info *info = (struct ipt_length_info *)(*match)->data;
    int len;

    switch (c) {
        case 'l':
            if (*flags)
                exit_error(PARAMETER_PROBLEM,
                           "length: `--length' may only be "
                           "specified once");
            len = string_to_number(argv[optind-1], 0, 0xFFFF);
            if (len == -1)
                exit_error(PARAMETER_PROBLEM,
                           "length invalid: `%s'\n", s);
            *flags = 1;
            break;

        default:
            return 0;
    }
    return 1;
}

/* Final check; must have specified --length. */
static void
final_check(unsigned int flags)
{
    if (!flags)
        exit_error(PARAMETER_PROBLEM,
                   "length: You must specify `--length'");
}
```

Netfilter Rusty Russell

```
/* Common match printing code. */
static void
print_length(struct ipt_length_info *info)
{
    printf("%u ", info->length);
}

/* Prints out the matchinfo. */
static void
print(const struct ipt_ip *ip,
      const struct ipt_entry_match *match,
      int numeric)
{
    printf("length ");
    print_length((struct ipt_length_info *)match->data);
}

/* Saves the union ipt_matchinfo in parsable form to stdout. */
static void
save(const struct ipt_ip *ip, const struct ipt_entry_match *match)
{
    printf("--length ");
    print_length((struct ipt_length_info *)match->data);
}

struct iptables_match length
= { NULL,
    "length",
    NETFILTER_VERSION,
    sizeof(struct ipt_length_info),
    sizeof(struct ipt_length_info),
    &help,
    &init,
    &parse,
    &final_check,
    &print,
    &save,
    opts
};

void _init(void)
{
    register_match(&length);
}
```

Using it is now simple a matter of

```
iptables -m length --length 100 -j DROP
```

6.5 Other Extensions

Both the connection tracking code and NAT code can be taught about new transport protocols (eg. PPTP)

Both can also have `helpers' to assist with particular application protocols (eg. ftp, irc, etc).

Netfilter

Rusty Russell

6.6 Summary

The framework is very extensible

Even without altering the core code, alot of flexibility is given at various levels

Documentation is fairly good (netfilter hacking HOWTO)

Flexibility should increase design life

7. Netfilter: Useful URLs

Netfilter home pages:

- <http://www.samba.org/netfilter>
- <http://netfilter.kernelnotes.org>
- <http://netfilter.filewatcher.org>

`ip' tool:

- `iproute' deb and rpm
- <ftp://ftp.inr.ac.ru/> (lists ip-routing/ mirrors)

Advanced Routing HOWTO

- <http://www.ds9a.nl/2.4Routing>

Netfilter documentation:

- <http://www.samba.org/netfilter/unreliable-guides>
- <http://netfilter.kernelnotes.org/unreliable-guides>
- <http://netfilter.filewatcher.org/unreliable-guides>