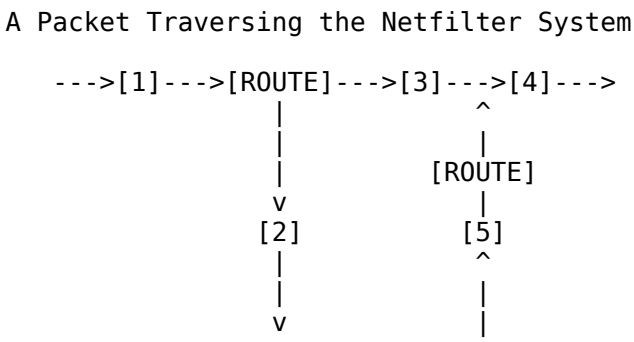
Next Previous Contents

3. Netfilter Architecture

Netfilter is merely a series of hooks in various points in a protocol stack (at this stage, IPv4, IPv6 and DECnet). The (idealized) IPv4 traversal diagram looks like the following:



On the left is where packets come in: having passed the simple sanity checks (i.e., not truncated, IP checksum OK, not a promiscuous receive), they are passed to the netfilter framework's NF_IP_PRE_ROUTING [1] hook.

is this the truely routing process or just the forwarding process. If routing, who c

Next they enter the routing code, which decides whether the packet is destined for another interface, or a local process. The routing code may drop packets that are unroutable.

from this description, it is possibly a forwarding process.

If it's destined for the box itself, the netfilter framework is called again for the NF_IP_LOCAL_IN [2] hook, before being passed to the process (if any).

If it's destined to pass to another interface instead, the netfilter framework is called for the NF_IP_FORWARD [3] hook.

The packet then passes a final netfilter hook, the NF_IP_POST_ROUTING [4] hook, before being put on the wire again.

The NF_IP_LOCAL_OUT [5] hook is called for packets that are created locally. Here you can see that routing occurs after this hook is called: in fact, the routing code is called first (to figure out the source IP address and some IP options): if you want to alter the routing, you must alter the `skb->dst' field yourself, as is done in the NAT code.

3.1 Netfilter Base

Now we have an example of netfilter for IPv4, you can see when each hook is activated. This is the essence of netfilter.

Kernel modules can register to listen at any of these hooks. A module that registers a function must specify the priority of the function within the hook; then when that netfilter hook is called from the core networking code, each module registered at that point is called in the order of priorites, and is free to manipulate the packet. The module can then tell netfilter to do one of five things:

1. NF_ACCEPT: continue traversal as normal.
2. NF_DROP: drop the packet; don't continue traversal.
3. NF_STOLEN: I've taken over the packet; don't continue traversal.
4. NF_QUEUE: queue the packet (usually for userspace handling).
5. NF_REPEAT: call this hook again.

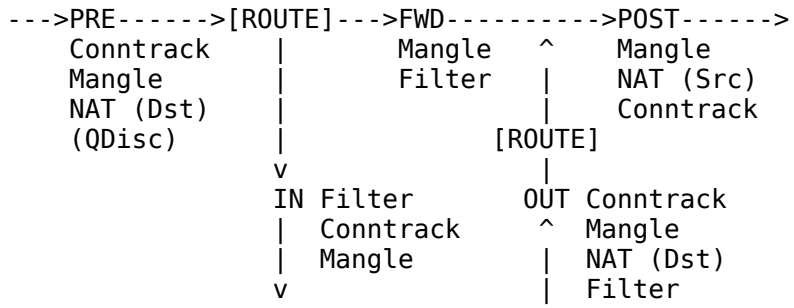
The other parts of netfilter (handling queued packets, cool comments) will be covered in the kernel section later.

Upon this foundation, we can build fairly complex packet manipulations, as shown in the next two sections.

3.2 Packet Selection: IP Tables

A packet selection system called IP Tables has been built over the netfilter framework. It is a direct descendent of ipchains (that came from ipfwadm, that came from BSD's ipfw IIRC), with extensibility. Kernel modules can register a new table, and ask for a packet to traverse a given table. This packet selection method is used for packet filtering (the `filter' table), Network Address Translation (the `nat' table) and general pre-route packet mangling (the `mangle' table).

The hooks that are registered with netfilter are as follows (with the functions in each hook in the order that they are actually called):



Packet Filtering

This table, `filter', should never alter packets: only filter them.

One of the advantages of iptables filter over ipchains is that it is small and fast, and it hooks into netfilter at the NF_IP_LOCAL_IN, NF_IP_FORWARD and NF_IP_LOCAL_OUT points. This means that for any given packet, there is one (and only one) possible place to filter it. This makes things much simpler for users than ipchains was. Also, the fact that the netfilter framework provides both the input and output interfaces for the NF_IP_FORWARD hook means that many kinds of filtering are far simpler.

Note: I have ported the kernel portions of both ipchains and ipfwadm as modules on top of netfilter, enabling the use of the old ipfwadm and ipchains userspace tools without requiring an upgrade.

NAT

This is the realm of the `nat' table, which is fed packets from two netfilter hooks: for non-local packets, the NF_IP_PRE_ROUTING and NF_IP_POST_ROUTING hooks are perfect for destination and source alterations respectively. If CONFIG_IP_NF_NAT_LOCAL is defined, the hooks NF_IP_LOCAL_OUT and NF_IP_LOCAL_IN are used for altering the destination of local packets.

This table is slightly different from the `filter' table, in that only the first packet of a new connection will traverse the table: the result of this traversal is then applied to all future packets in the same connection.

Masquerading, Port Forwarding, Transparent Proxying

I divide NAT into Source NAT (where the first packet has its source altered), and Destination NAT (the first packet has its destination altered).

Masquerading is a special form of Source NAT: port forwarding and transparent proxying are special forms of Destination NAT. These are now all done using the NAT framework, rather than being independent entities.

Packet Mangling

The packet mangling table (the `mangle' table) is used for actual changing of packet information. Example applications are the TOS and TCPMSS targets. The mangle table hooks into all five netfilter hooks. (please note this changed with kernel 2.4.18. Previous kernels didn't have mangle attached to all hooks)

3.3 Connection Tracking

Connection tracking is fundamental to NAT, but it is implemented as a separate module; this allows an extension to the packet filtering code to simply and cleanly use connection tracking (the `state' module).

3.4 Other Additions

The new flexibility provides both the opportunity to do really funky things, but for people to write enhancements or complete replacements that can be mixed and matched.

Next Previous Contents