

# IP Tables

- are used on various operating systems:
  - OSX, Solaris, FreeBSD and Linux
- learn once use many times

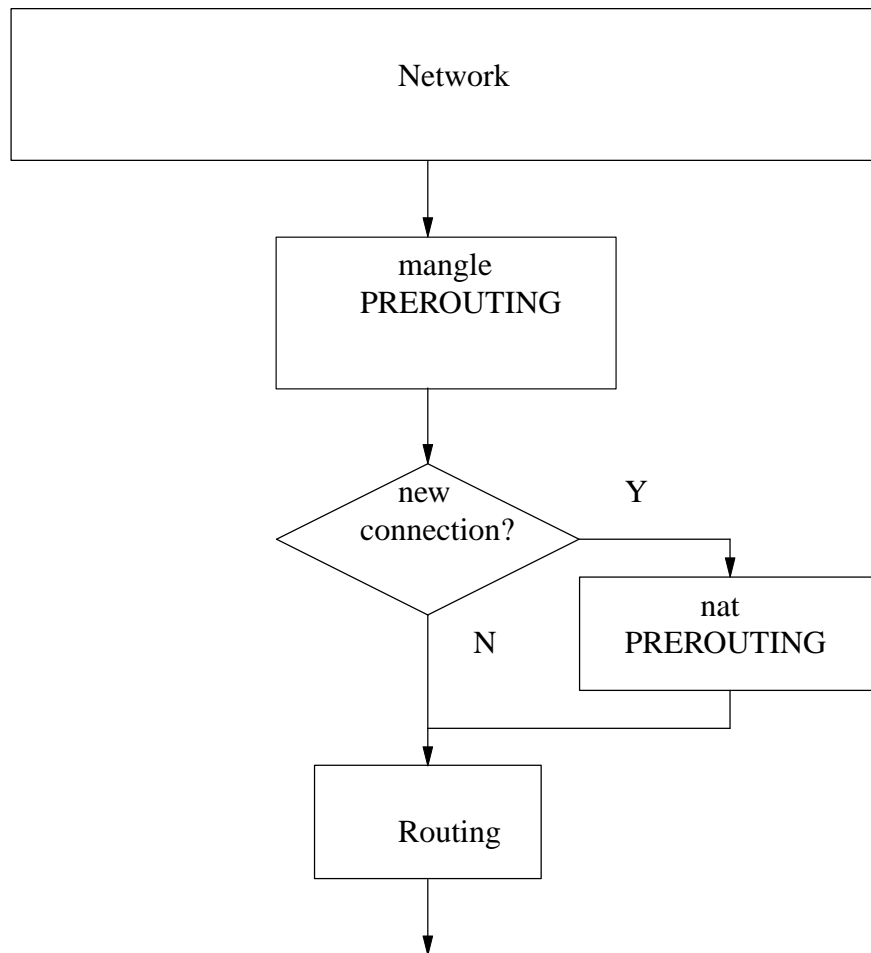
# IP Tables

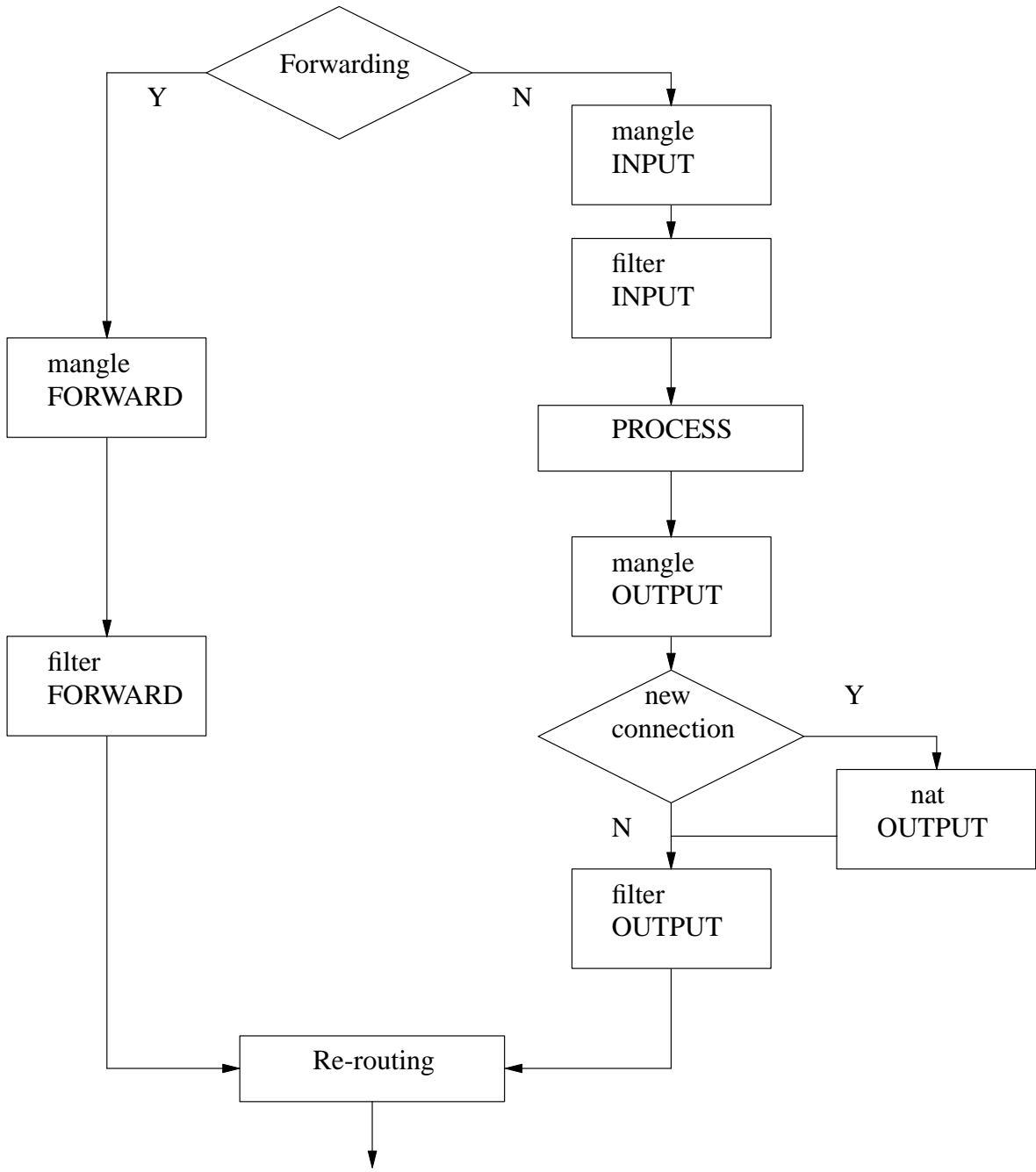
- is the name of the user space tool by which administrators create rules for the packet filtering and NAT modules
- the `netfilter` project is the kernel network infrastructure which implements a stateful firewall
  - `iptables` configures it in the user space

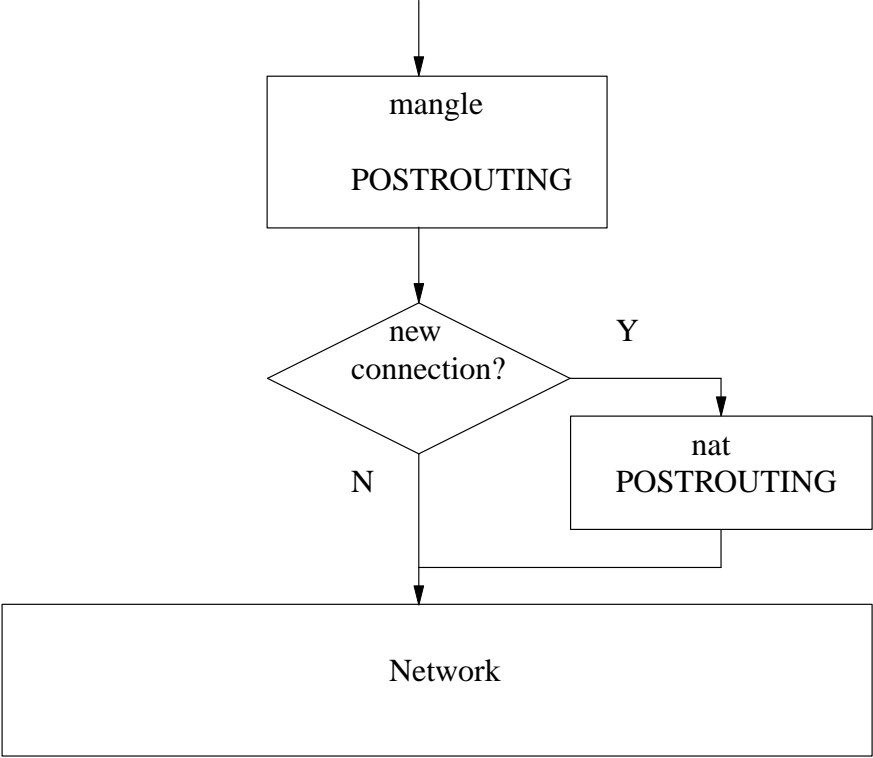
# IP Tables

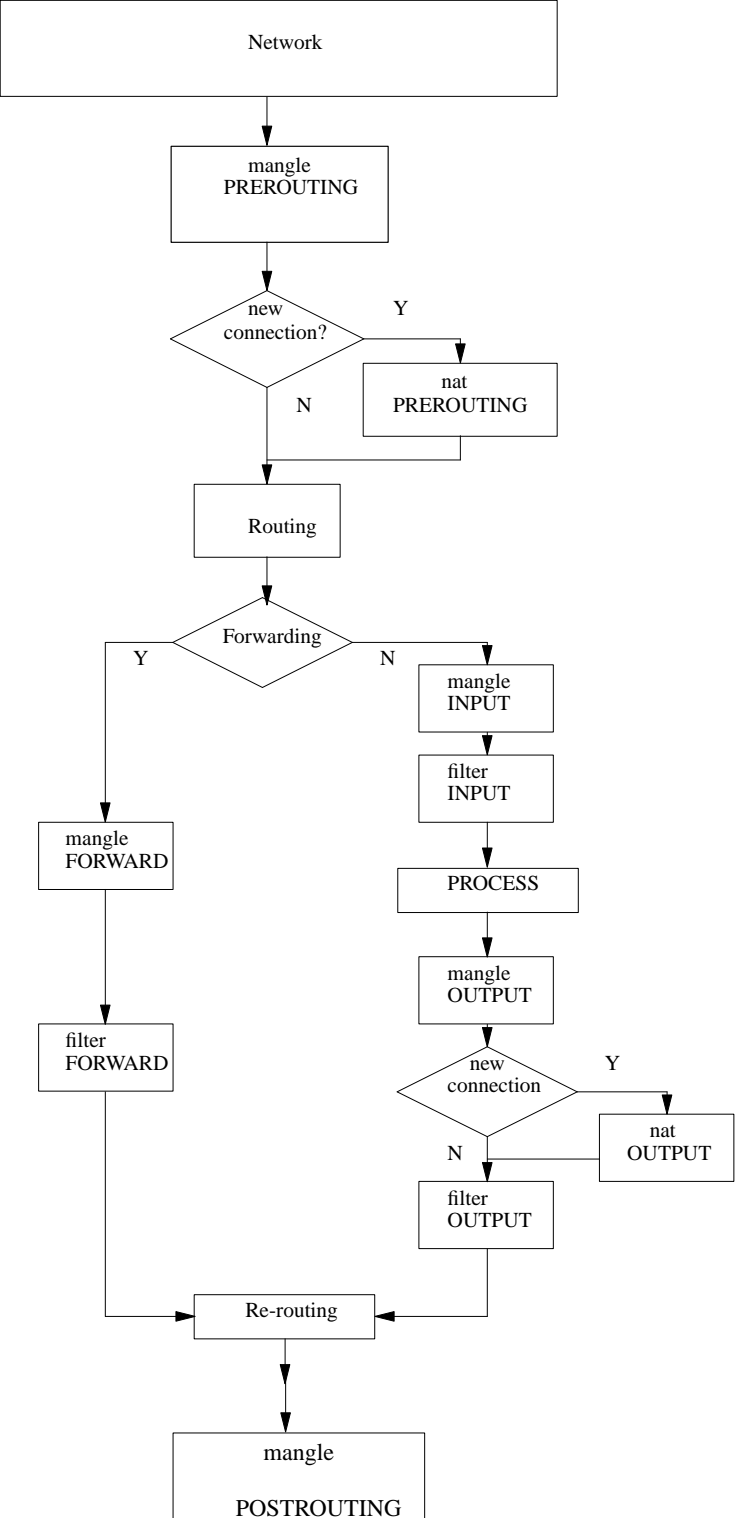
- internally iptables views the task of handling packets as one of passing packets through a list of rules
  
- it uses three tables to separate out the different categories of packets
  - filter
  - NAT
  - mangle
  
- initially tables are set to allow packets to enter and leave a system

# Tables and chains









## Filter table

- is responsible for filtering (blocking or permitting a packet to proceed)
- every packet passes through the filter table
- *is the default table if no other is specified*

## Filter table

- it contains the following predefined chains, and any packet will pass through one of them:
  - chain INPUT: all packets destined for this system go through this chain (hence sometimes referred to as LOCAL\_INPUT)
  - chain OUTPUT: all packets created by this system go through this chain (aka. LOCAL\_OUTPUT)
  - chain FORWARD: all packets merely passing through the system (being routed) go through this chain

## NAT Table

- is responsible for setting up the rules for rewriting packet addresses or ports
- the first packet in any connection passes through this table:
  - any verdicts here determine how all packets in that connection will be rewritten

## NAT Table

- it contains the following predefined chains:
  - chain PREROUTING Incoming packets pass through this chain before the local routing table is consulted, primarily for DNAT (destination-NAT)
  - chain POSTROUTING Outgoing packets pass through this chain after the routing decision has been made, primarily for SNAT (source-NAT)
  - chain OUTPUT Allows limited DNAT on locally-generated packets

## Mangle table

- is responsible for adjusting packet options, such as quality of service
- all packets pass through this table, as it is designed for advanced effects, it contains all the possible predefined chains:
  - chain PREROUTING: all packets entering the system in any way, before routing decides whether the packet is to be forwarded (FORWARD chain) or is destined locally (INPUT chain)
  - chain INPUT: all packets destined for this system go through this chain
  - chain FORWARD: all packets merely passing through the system go through this chain
  - chain OUTPUT: all packets created by this system go through this chain
  - chain POSTROUTING: all packets leaving the system go through this chain



## IP tables

- in addition to the built-in chains, the user can create any number of user-defined chains within each table, which allows them to group rules logically
  
- each chain contains a list of rules
  - when a packet is sent to a chain, it is compared against each rule in the chain in order
  
- the rule specifies what properties the packet must have for the rule to match
  - such as the port number or IP address

## IP tables

- if the rule does not match then processing continues with the next rule
- if, however, the rule does match the packet, then the rule's target instructions are followed (and further processing of the chain is usually aborted)
- some packet properties can only be examined in certain chains
  - eg the outgoing network interface is not valid in the INPUT chain
- some targets can only be used in certain chains, and/or certain tables (for example, the SNAT target can only be used in the POSTROUTING chain of the nat table)

## Rule targets

- the target of a rule can be the name of a user-defined chain or one of the built-in targets ACCEPT, DROP, RETURN, DNAT, SNAT and MASQUERADE
  - there are others but we will not cover these here
  
- can think of a target in the same way as a subroutine
  
- ACCEPT
  - tells netfilter to accept the packet
    - eg. input can ACCEPT packet
    - output can allow a packet to travel outwards

## Rule targets

- DROP drop the packet and do not send an error report to the originator
- REJECT drop the packet and do send an error report to the originator
- DNAT packet's destination address (and optionally port) are rewritten for network address translation
  - the `--to-destination` flag must be supplied to indicate the destination to use
  - it is only valid in the `OUTPUT` and `PREROUTING` chains within the `nat` table

## Rule targets

- SNAT target causes the packet's source address (and optionally port) to be rewritten for network address translation
  - the `--to-source` flag must be supplied to indicate the source
  - it is only valid in the `POSTROUTING`
  
- MASQUERADE
  - is a special, restricted form of SNAT for dynamic IP addresses, such as most Internet service providers provide for modems or DSL
  - rather than change the SNAT rule every time the external interface IP changes it examines the external interface and automatically SNATs the packets

## Port forward input from eth0 -> 192.168.0.2:80

```
# iptables -A PREROUTING -t nat -i eth0 -p tcp \  
--dport 80 -j DNAT --to 192.168.0.2:80  
# iptables -A FORWARD -p tcp -m state \  
--state NEW -d 192.168.0.2 --dport 80 -j ACCEPT
```

- eth0 faces the outside world
- the first rule says that all incoming tcp connections arriving destined for port 80 should be sent to the internal machine 192.168.0.2 (also on port 80)

## Port forward input from eth0 -> 192.168.0.2:80

- the second rule, after the DNATting, the packet will be handled by the FORWARD chain
- easy to restrict access to our web server by modifying the first rule:

```
# iptables -A PREROUTING -t nat -i eth0 -p tcp \  
--source 193.63.148.0/24 \  
--dport 80 -j DNAT --to 192.168.0.2:80
```

- which says only the class C addresses 193.63.148.0/24 can connect to our web server

## iptables examples

- allow 2 telnet connections per client host

```
# iptables -A INPUT -p tcp --syn --dport 23 -m connlimit \  
--connlimit-above 2 -j REJECT
```

## iptables examples

- using DNAT for all protocols *and ports* on one IP

```
# iptables -t nat -A PREROUTING -d 1.1.1.1 \  
-j DNAT --to-destination 2.2.2.2
```

- in this example, all packets arriving on the router with a destination of 1.1.1.1 will depart from the router with a destination of 2.2.2.2

## iptables examples

- using DNAT for a single port

```
# iptables -t nat -A PREROUTING -p tcp -d 10.10.20.99 \  
--dport 80 -j DNAT --to-destination 10.10.14.2
```

- simulating full NAT with SNAT and DNAT

```
# iptables -t nat -A PREROUTING -d 205.254.211.17 \  
-j DNAT --to-destination 192.168.100.17  
# iptables -t nat -A POSTROUTING -s 192.168.100.17 \  
-j SNAT --to-destination 205.254.211.17
```

## iptables examples

- tells iptables to drop packets with the record-route flag

```
# iptables -A input -m ipv4options --rr -j DROP
```

- will drop packets with the timestamp flag

```
# iptables -A input -m ipv4options --ts -j DROP
```

- the `recent` option in `iptables` allows you to dynamically create a list of IP addresses and then match against that list in a few different ways

## iptables examples

- you can create a *badguy* list out of people attempting to connect to port 139 on your firewall and then DROP all future packets from them without considering them

```
# iptables -A FORWARD -m recent --name badguy --rcheck \  
--seconds 60 -j DROP  
# iptables -A FORWARD -p tcp -i eth0 --dport 139 \  
-m recent --name badguy --set -j DROP
```

## iptables examples

- `--rcheck` tests to see if the source address is currently in the list
- `-A` append one or more rules to the end of the selected chain `-i` interface card (in our examples always `eth0`)
- `-p` protocol (tcp in our examples)
- `-j` jump to target, what it must do if the packet matches it
  - the target can be another chain or one of the builtin targets

## Further reading

- manual page for iptables
- [Debian administration](http://www.debian-administration.org/articles/73) `<http://www.debian-administration.org/articles/73>`