Technische Universität Berlin

IPv4 III – Routing and Port Forwarding

Training for Afghan System Administrators

Routing of IP packets

Routing is the process of finding a route between hosts on a computer network. Whenever HOST_A (*source*) wants to send an IP packet to HOST_B (*destination*):

1) It will first check if the *destination* host is in the same IP network (using its own IP address and the subnet mask to calculate the network of the destination). Depending on this check:
   A) If the *destination* host is in the same network, the *source* host sends the packet directly to the *destination* host (and the *destination* host will answer directly)
   B) If the *destination* is NOT in the same network, the *source* sends the packet to a *router* according to its routing table. If there is no static route defined for the *destination* host's network, the *source* sends it to its *default gateway*
   C) If no static route or default gateway are defined that can handle the request, a routing error will occur and an error message will be sent to the *source*
2) The next node (router, gateway, firewall, etc) will then do the same calculation (as in step 1 above)
3) This will continue for as long until the packet reaches the router that is handling the network the *destination* host is located in. This router delivers the packet to the *destination* host
4) The destination now processes the request and then sends its answer (packet)
5) The whole process starts again, but in the opposite direction, until the router responsible for HOST_A is reached. The route the packets will travel back is probably, but not necessarily, the same!

Any Linux computer can be a router!

To turn any Linux computer into a router, all you need to do is write a routing table (at least add a default route), and activate the IP forwarding in the kernel. To see the current state:

- cat /proc/sys/net/ipv4/ip_forward
  OR
- sysctl net.ipv4.ip_forward
  # both commands will show the current state: 0 means forwarding is disabled / 1 means enabled

To enable forwarding (non-permanent / does not survive a reboot!):

- echo 1 > /proc/sys/net/ipv4/ip_forward
  OR
- sysctl -w net.ipv4.ip_forward=1
  # this overwrites the current value with 1 and enables forwarding

To change from disabled to enabled (permanently) the file /etc/sysctl.conf needs to be edited!

A firewall: just a router with a set of (iptables) rules!

Any Linux router can – if the necessary kernel options and modules ("drivers") are available – also become a firewall, by using the iptables utility which manipulates how the kernel deals with packets.

Iptables works by first defining rules (which can be entered on the command line or combined in a file), and then, when packets travel through the router, each packet is checked against all the rules. If it matches a rule, then an action is performed. The structure of an iptables rule is always the same:

iptables [-t <table-name>] <command> <chain-name> <matches><target>

- <table-name> Specifies which table the rule applies to. If omitted, the *filter* table is used
- <command> Specifies the action to perform, such as appending or deleting a rule
- <chain-name> Specifies the chain to edit, create, or delete
- <matches> Parameters and options that specify which packets match this rule
- <target> Specifies what to do with packets that match this rule

Iptables rules in detail

These are the common values for the components of an iptables rule

- Tables: filter, mangle, nat
- Commands: insert (-I), append (-A), delete (-D), replace (-R), list (-L), flush (-F), new chain (-N), policy (-P)
- Chains: INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING
- Matches: protocol (-p), source (-s), destination (-d), input interface (-i), output interface (-o), destination port (--dport), source port (--sport)
- Target: ACCEPT, REJECT, DROP, MASQUERADE, SNAT, DNAT, REDIRECT, LOG

Some example rules (might need other rules to function properly; see e.g. later slide for port forwarding):

- iptables -t filter -A INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT

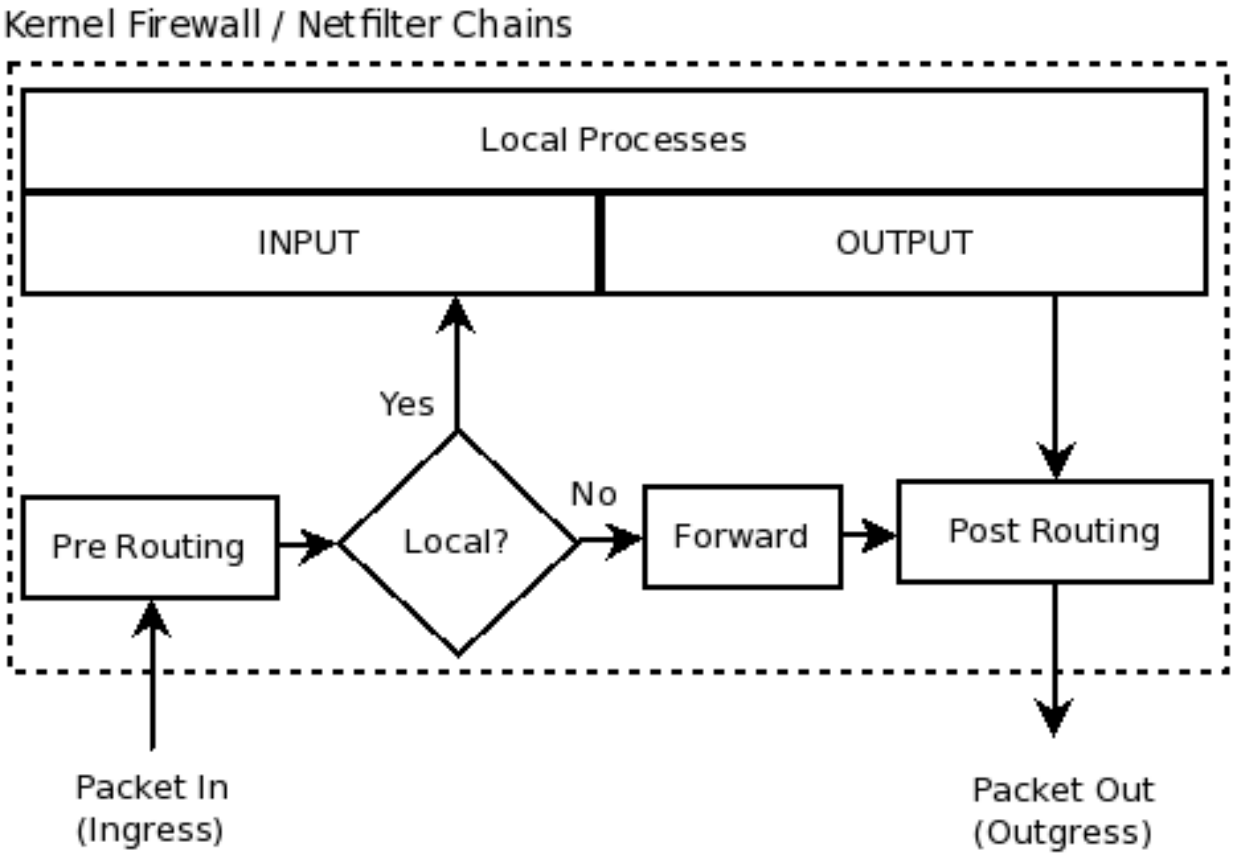
  # Allow TCP traffic on port 80 (http) that originated outside our network (state NEW)
- iptables -t nat -A POSTROUTING -s 192.168.0.1/24 -o ppp0 -j MASQUERADE

  # Create a NAT network that allows outgoing traffic via ppp0 from the 192.... network without knowing what IP the ppp0 interface has (dynamic IP)
- iptables -t nat -A POSTROUTING -s 192.168.0.1/24 -o eth0 -j SNAT

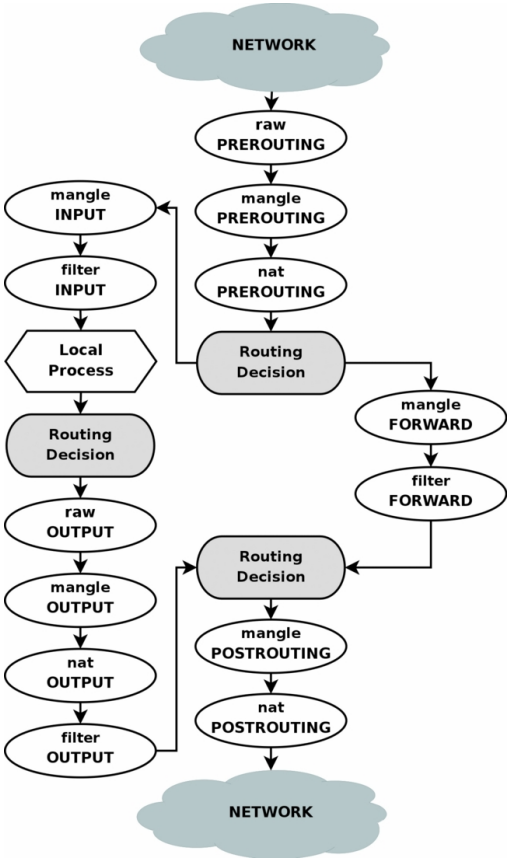
  # Same as above but with a fixed public IP on eth0 (faster because no IP lookup necessary)
- iptables -t nat -A PREROUTING -i ppp0 -p tcp --dport 80 -j DNAT --to-destination 192.168.0.1:80

  # Enable port forwarding for packets from ppp0 on port 80 to host 192.168.0.1 on port 80 (http)

How packet filtering with iptables works (simplified)



How packet filtering with iptables works (detailed)



The problem of IPv4 depletion

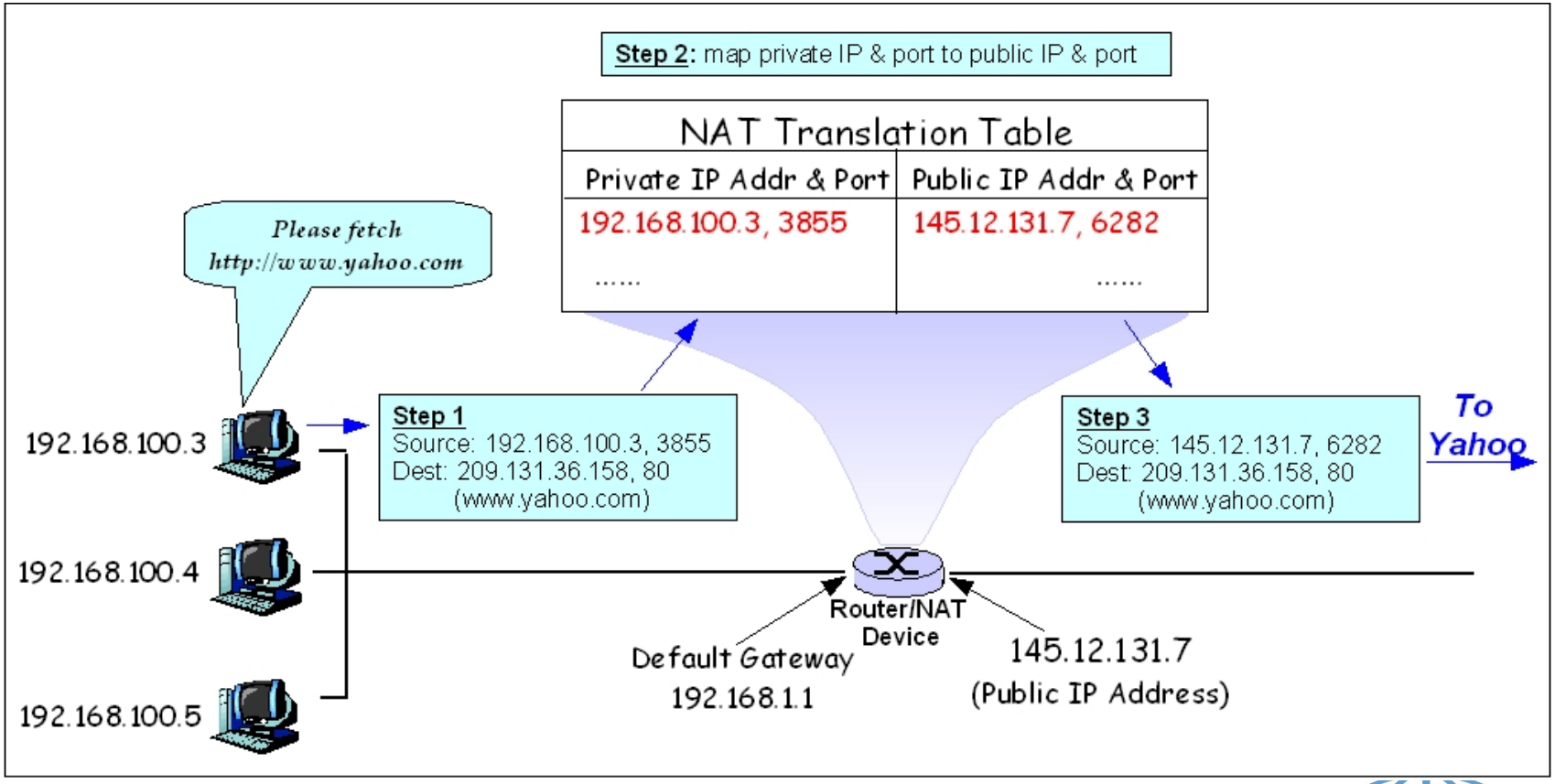
In theory, if every computer connected to the internet had its own unique address, then exchanging information between any of them would be very easy, and would work exactly as described on the previous slide. But it is not as simple as that, because there are not enough addresses!!!

In the address space of IPv4 (RFC 791; 1981), there are a theoretical maximum of 2^32 (4.294.967.296) addresses. However, with some parts of this address space being reserved (for private networks, multicast, future use, etc), the real number of public IP addresses available is 3,706,452,992.

This number might sound big, but since the late 1980s it was clear, that the number is MUCH too small to handle the amount of network enabled devices worldwide. This was the reason for the invention of IPv6, which offers an theoretical address space of 2^128 addresses (340282366920938463463374607431768211456 or ~ 340 undecillion), minus the reserved spaces

But IPv6 (RFC 2460; 1998) adoption was / is very slow, so that in the meantime, a technique called network address translation (NAT) is used, to handle a much larger amount of devices than IPv4 addresses

Network address translation



Creating a NAT router with iptables

To turn a router into a NAT router, we only need a few iptables rules. Assuming out router has 2 network cards, eth0 being the public one and eth1 being the internal one, then:

- iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
  # this will enable NAT / masquerading for the external network card eth0
- iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
  # this will allow all outgoing traffic from the internal network eth1 via the eth0
- iptables -A FORWARD -i eth0 -o eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
  # this will allow all incoming traffic on eth0 to be forwarded to eth1 but only if the connection was initiated from the inside first (e.g. it will forward the answer to a request from inside our NAT network)

The problem with network address translation

If connections are initiated *inside* the LAN, e.g. when your browser requests www.google.com, the router which handles the NAT knows that your browser requested the google page. So when google finally sends the answer, the router knows which computer the request came from and will send the answer to it.

But what if you want to run a server, e.g. a webserver inside your NAT network? When a foreign host outside of your NAT network wants to connect to your webserver, all it has is the public IP of the router that handles the NAT. So it will send a request to that IP and destination port 80 (http). But the router does not know what to do with this packet, because on the router itself, there is no webserver running (= port 80 is not listening / open), so it will drop the packet and close the connection. In effect, your webserver is not reachable from the outside world :(

Port forwarding (DNAT) to the rescue!

By configuring the router's netfilter table / iptables rules you can instruct the router to pass on packets which are meant for servers inside the the NAT network. This happens on a *per port* base. In the case of our webserver inside our NAT network, we need to tell the router to pass on incoming requests to port 80 (http) to the webserver inside our NAT network with the source IP and port unchanged, and our webserver will send its answer to that address / port. This can also be done with the iptables utility:

```
iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 80 -j DNAT --to-destination
192.168.1.2:80
iptables -A FORWARD -p tcp -d 192.168.1.2 --dport 80 -j ACCEPT
```

If the router is configured to pass on all traffic on all ports to a host inside our NAT network, this is called a demilitarized zone (DMZ) host. Demilitarized because all packet filter (=firewall) functionality on the router is bypassed, and it appears as if the DMZ host is directly connected to the public IP of the router (the router itself becomes “invisible” to the outside).

Further reading

- http://www.iptables.info/en/iptables-contents.html
- http://www.netfilter.org/
- http://www.howtogeek.com/177621/the-beginners-guide-to-iptables-the-linux-firewall/
- http://www.thegeekstuff.com/2011/06/iptables-rules-examples/
-