Netkit introduction & basic addressing

Network Infrastructures labs

Marco Spaziani Brunella

SAPIENZA
UNIVERSITÀ DI ROMA

Lecture details

- **Readings:**
  - Linux Network Administrators Guide
- **Lecture outline:**
  - Netkit introduction
  - Nomenclature
  - First lab
  - Basic network commands
  - Network debug tools

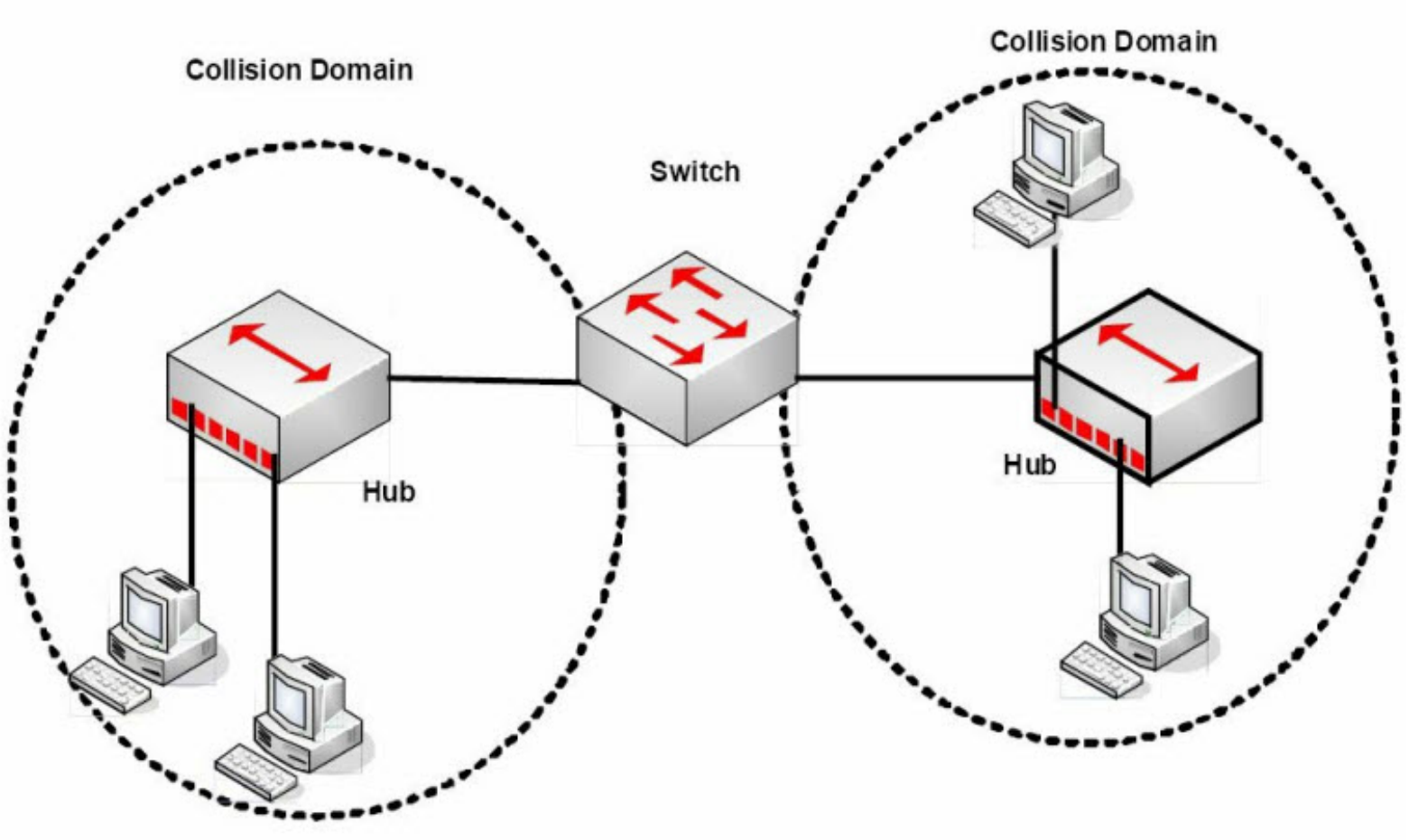
What is Netkit?

"Netkit is a self-contained low cost solution for the emulation of computer networks. Within the Netkit environment each network device is implemented by a virtual machine, and interconnection links are emulated by using virtual collision domains (which you can see as virtual hubs)"

"Each virtual machine can be configured to have an arbitrary number of (virtual) network interfaces"

"Virtual machines use a customized filesystem which contains a standard Linux installation (at the time this man page is being written, it is a Debian distribution), which includes network oriented software such as routing daemons (RIP, OSPF, etc.), a bunch of servers (FTP, HTTP, etc.), firewalling utilities (iptables), diagnostic tools (ping, traceroute, tcpdump, etc.), and other applications"

Collision Domain vs Broadcast Domain



Network Interface Card

A network interface card (NIC) is the physical device in which you plug the cable (e.g. 802.3) or provides radio access to the medium (e.g. 802.11).

The NIC interacts with the OS via drivers.
The OS can virtualize NIC, meaning that can manage NIC that do not correspond to physical devices.
In linux (before systemd v197 adoption) the naming scheme for ethernet interfaces was ethX, where X is a number starting from 0.

Basic Netkit commands

In netkit, we will use basically 4 commands, altough there are many others:

```
vstart [options] VM-NAME #starts a netkit virtual
                         #machine called VM-NAME

vcrash [options] VM-NAME #crash a running netkit
                         #virtual machine called VM-NAME

lstart #starts a netkit laboratory sequentially

lstart -p0 #laboratory parallel startup

lcrash #crash a netkit laboratory
```

Virtual Machine Demo

VM Info

In the virtual machine we are the "root" user so our home directory is in /root .
In / there is a folder called "hosthome" wich is mounted on the home of the user who launched the VM.
This is very useful if you want to pass files to the VM.
Single virtual machines are not that interesting, since they do not create a network ☺

Netkit Laboratories

In order to simulate "complex" networks, netkit has laboratories, wich specifies how many machines we have on the virtual network and how them are connected toghether.
The basic ingredients of a laboratory are:

- Lab direcotry
- VM direcotries
- lab.conf

The Lab directory is where all the files of the lab are contained, such as lab.conf and VM directories.
For every VM on the virtual network, we must have a folder named as the VM inside the Lab directory.

Nomenclature

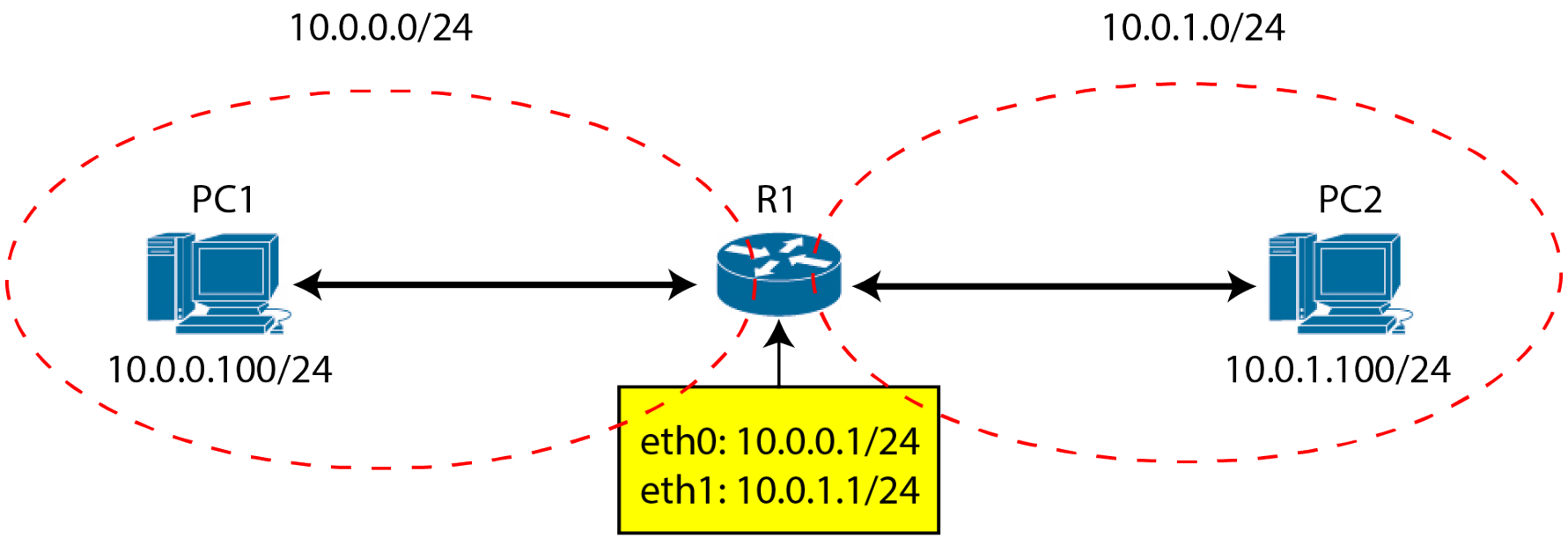
We will use the convention for IP addresses and subnet with the /:

192.168.0.1/24

In general, we will use the following rules to interpret topology schemes:

- a.b.c.0/m to define the subnet
- a.b.c.1/m to define the subnet gateway

lab_0


10.0.0.0/24
10.0.1.0/24
PC1
R1
PC2
10.0.0.100/24
10.0.1.100/24
eth0: 10.0.0.1/24
eth1: 10.0.1.1/24

VM directory

In order to create the above topology, lets create a folder "lab_0".
Inside lab_0, we have to create the folders related to the nodes:

- pc1
- pc2
- r1

These folders are mounted inside the / of every single VM: every file inside the VM folder will be mapped(create or replace existing files) inside the filesystem of the VM.

lab.conf

In order to start properly the lab, we have to create a "lab.conf" file inside the lab_0 folder.
This file specifies the connection of the VMs interfaces to collision domains and more (*man lab.conf* for details).

```
pc1[0]=A #set eth0 of pc1 to collision domain A
r1[0]=A  #set eth0 of r1 to collision domain A

r1[1]=B  #set eth1 of r1 to collision domain B
pc2[0]=B #set eth0 of pc2 to collision domain B
```

Starting the lab

We are ready to start the lab.
In lab_0 directory:

```
user@localhost:$ lstart
```

It will pop up 3 VM shells, pc1 - pc2 - r1.
In every single VM, inside the / directory, there is an additional folder, named "hostlab", wich points to the folder in wich the lab is (in our case lab_0).

iproute2

In order to set up interface, we will use a commands wich are part of the *iproute2* suite. These commands replace the old ones (such as ifconfig):

Legacy utility	Obsoleted by	Note
`ifconfig`	`ip addr`, `ip link`, `ip -s`	Address and link configuration
`route`	`ip route`	Routing tables
`arp`	`ip neigh`	Neighbors
`iptunnel`	`ip tunnel`	Tunnels
`nameif`	`ifrename`, `ip link set name`	Rename network interfaces
`ipmaddr`	`ip maddr`	Multicast
`netstat`	`ip -s`, `ss`, `ip route`	Show various networking statistics

The commands on the first column should be considered deprecated and no more maintained, so let's not use them :)

ip link

The *ip link* command let us show and manage (setting up and down) network interfaces:

```
ip link show #shows the details of every available iface

ip link set eth0 up #brings up eth0 interface

ip link set eth0 down #brings down eth0 iface
```

For our lab_0, we need to bring up eth0 of pc1 and pc2 and we need to bring up also eth0 and eth1 of r1.

Assign IPv4 addresses to interfaces

Now that the ifaces are up, we are can assign IPv4 addresses to them. The command to assing IPv4 addresses to ifaces is *ip adress*:

```
ip address show #show information about addresses
                #assigned to interfaces

ip address add a.b.c.d/m dev ethX
#assigns a.b.c.d address with
#submask m to device ethx
```

With these commands, we now assign addresses to every single host as depicted on the topology picture.

Ping

We can test connection between two hosts using the *ping* command:

```
ping a.b.c.d #sends ICMP packets to a.b.c.d
             #and wait for reply
```

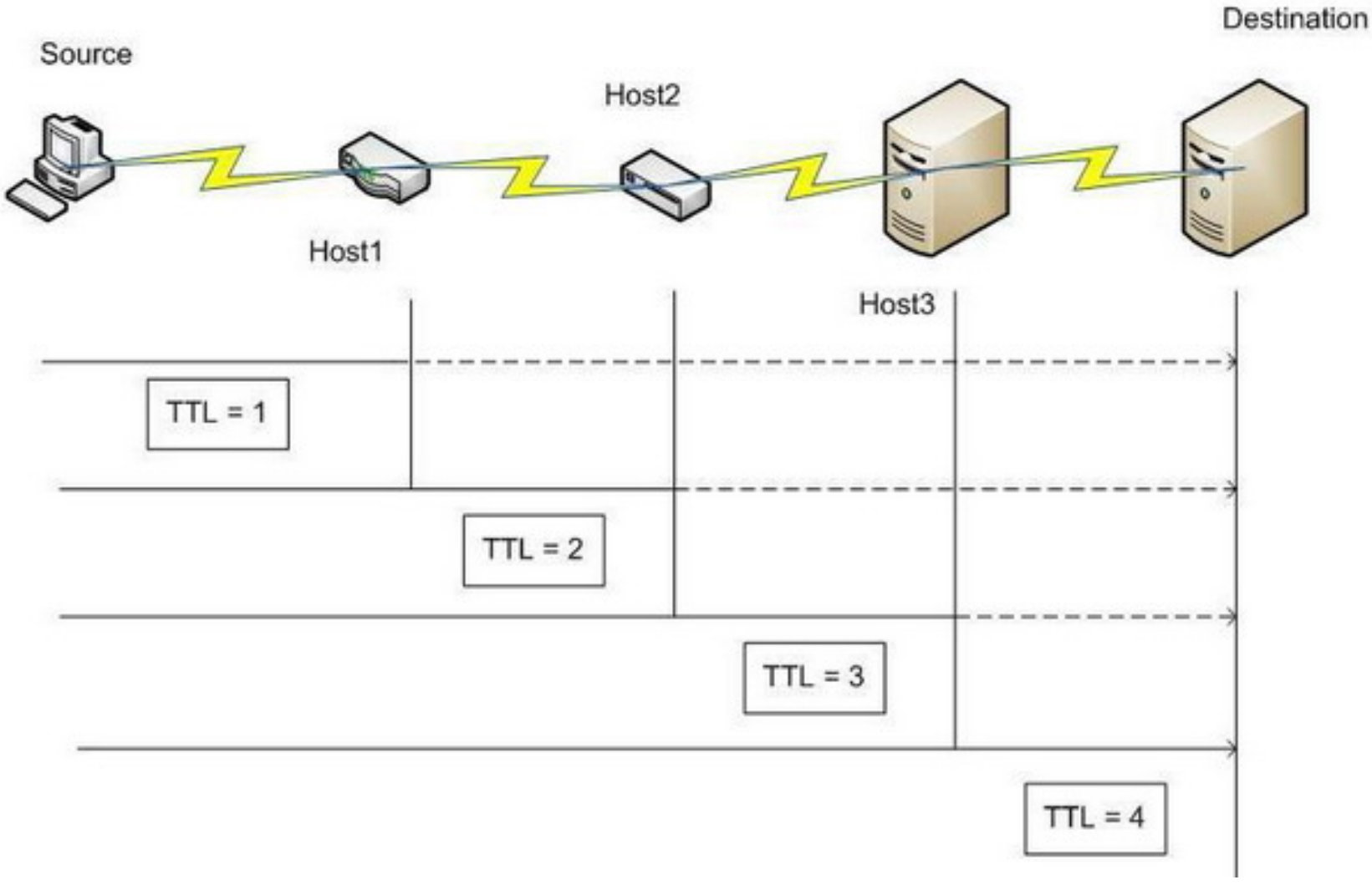
It gives us information about Round Trip Time and Time To Live (useful for traceroute) for a connection between 2 hosts.

traceroute

Another useful way of testing connection between two hosts is to traceroute the connection via the *tracert* command:

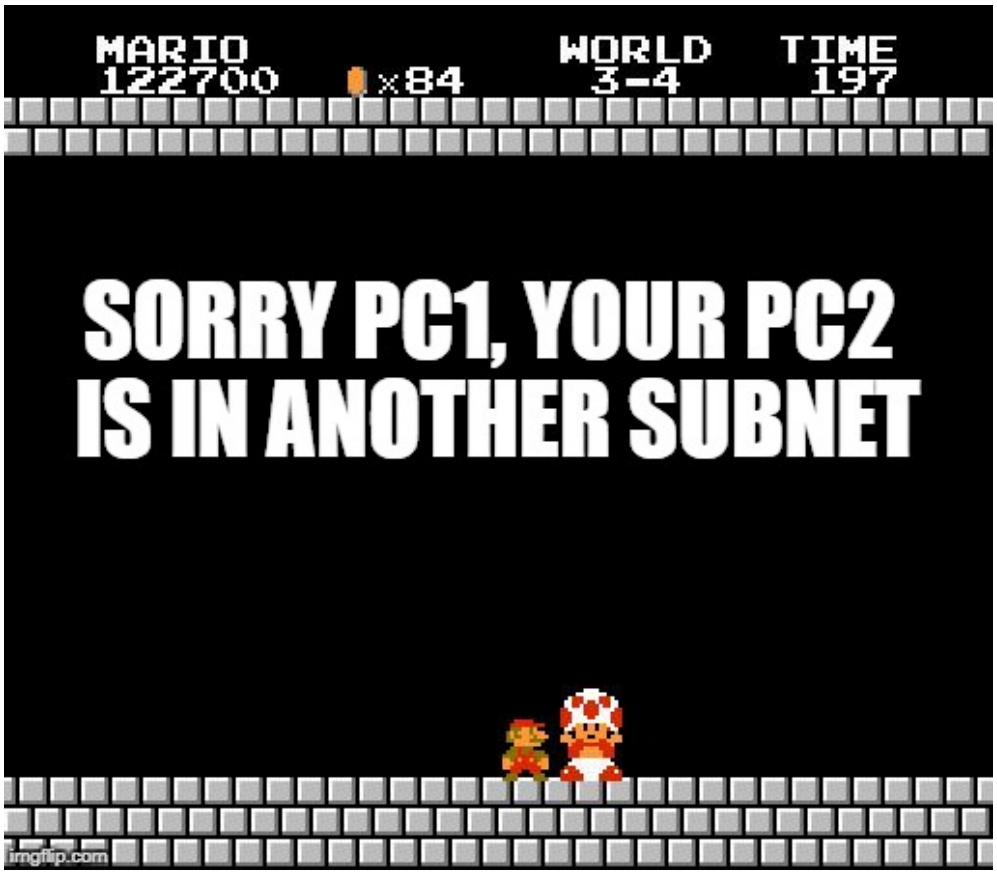
```
traceroute a.b.c.d #finds the number and addresses
                   #of hops to reach a.b.c.d
```

This command exploit the fact that every packet has a TTL field which is X when the packet leaves the NIC.
Every hop decrease by one the TTL field. When the TTL reaches 0, the host that expired the TTL sends back to the sender an ICMP error packet. The packet contains the IP address of the host who expired the TTL. By sending packets with TTL starting from 1, *tracert* can discover the path taken from the packets to reach a particular destination.


Source
Host2
Destination
Host1
Host3
TTL = 1
TTL = 2
TTL = 3
TTL = 4

Static routes

If we try to ping pc1 $\rightarrow$ pc2, we get "network unreachable". Why?! Because pc1 does not now how to reach pc2, cause he only has the route to his subnet which is 10.0.0.0/24, but pc2 is on 10.0.1.0/24.



Gateway

In order to let pc1 and pc2 communicate, we have two ways:

- Define the "default" route (gateway)
- Specify how to reach pc2 subnet from pc1 and viceversa

The difference between the two approaches will be clearer on bigger topology, but basically by defining a gateway we tell to the host "give all the packets that are not for your subnet to the gateway, he knows how to route them".
In our case the gateway is r1.
With the second approach, we tell to the host "give all the packets that are for a specific subnet to a specific host, he knows how to route them".

ip route

In order to manage routes, we use the *ip route* command:

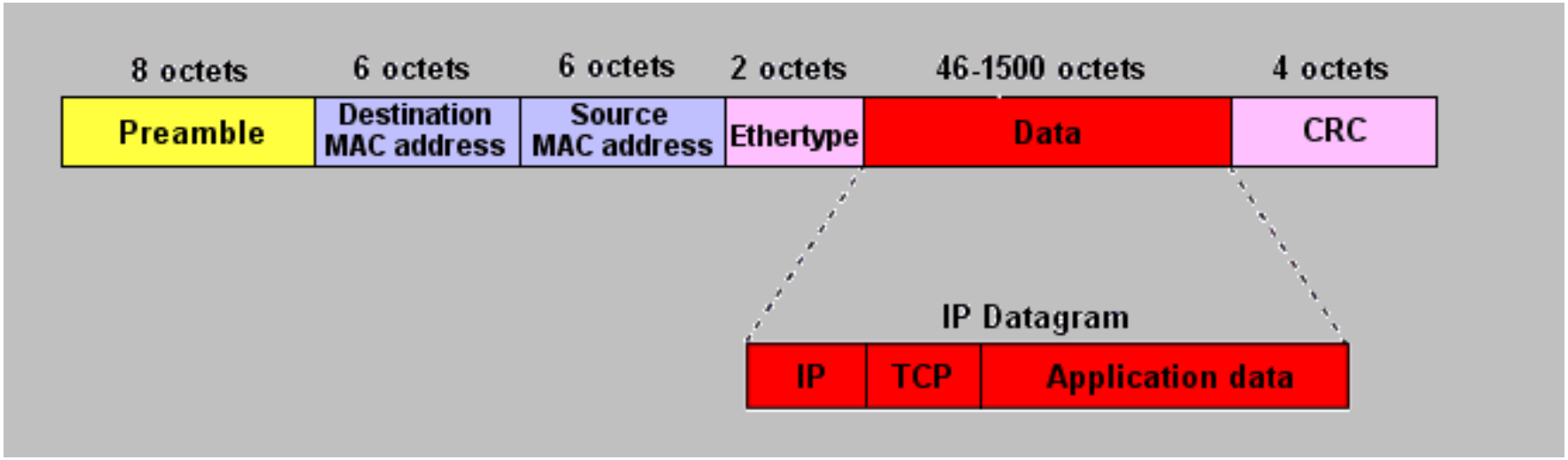
```
ip route show #show the available routes on the host

ip route add a.b.c.d/m via next_hop_ip
#adds the route to subnet a.b.c.d/m via the next_hop_ip
```

There is a special subnet, called "default", wich is 0.0.0.0/0, that enables you to define the gateway.

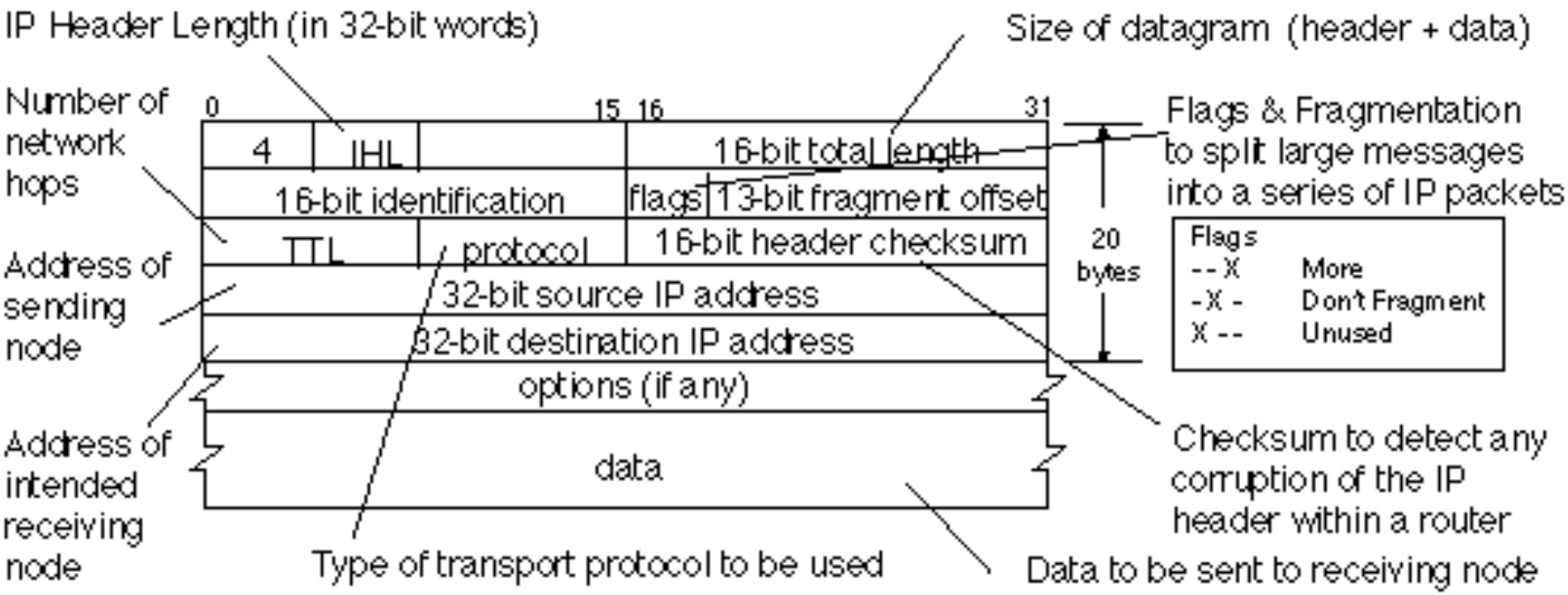
Packet inspection

Linux provides us tools to generate, capture and inspect packets.
We can create TCP/UDP packets via the *nc* (netcat) command.
This packets generated can be "captured" via the *tcpdump* command and stored to a "pcap" (packet capture) file.
This file can be later open to be graphically inspected with the *wireshark* tool.

Ethernet Frame



IP Frame


IP Header Length (in 32-bit words)
Size of datagram (header + data)
Number of network hops
Address of sending node
Address of intended receiving node
0
15 16
31
4
IHL
16-bit total length
16-bit identification
flags
13-bit fragment offset
TTL
protocol
16-bit header checksum
32-bit source IP address
32-bit destination IP address
options (if any)
data
20 bytes
Flags & Fragmentation to split large messages into a series of IP packets
Flags
-- X More
-X - Don't Fragment
X -- Unused
Checksum to detect any corruption of the IP header within a router
Type of transport protocol to be used
Data to be sent to receiving node

TCP Segments & UDP Datagrams

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

netcat

Netcat allows us to send data over TCP and UDP protocols. In order to use that, we must specify the protocol we ant to use, the port we want to use and if we want to send or "listen" on that particular port:

```
nc ip_addr port_num     #connects via TCP to ip_addr
                        #trough port_num via TCP

nc -u ip_addr port_num     #Send UDP datagrams
                           #to ip_addr trough port_num

nc -l -p port_num          #listen for TCP connections
                           #on port_num

nc -u -l -p port_num          #listen for UDP datagrams
                              #on port_num
```

tcpdump

tcpdump allows us to capture packets on a physical interface (not only TCP packets as the name may depict!)

```
tcpdump -ni iface_name -w filename.pcap
#capture packets on iface_name withouth resolving
#ip addresses and save captured packets on filename.pcap
```

Has a lot more configuration parameters $\rightarrow$ check on the manual.

Wireshark

After capturing packets with *tcpdump*, we can open the pcap file with *wireshark* to view the packets and their contents.