IP subnetting, routing and ARP

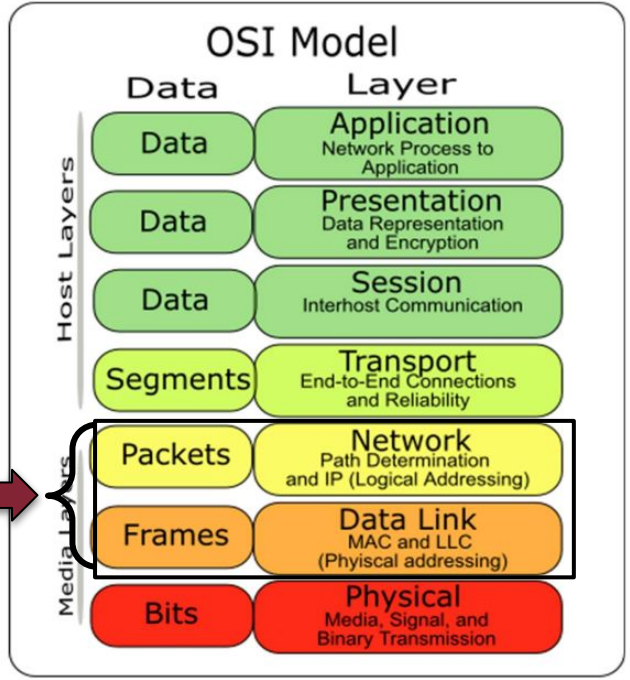
Network Infrastructures Lab Sessions

SAPIENZA
UNIVERSITÀ DI ROMA

Instructors:
- Pietro Spadaccino

Today on NI labs

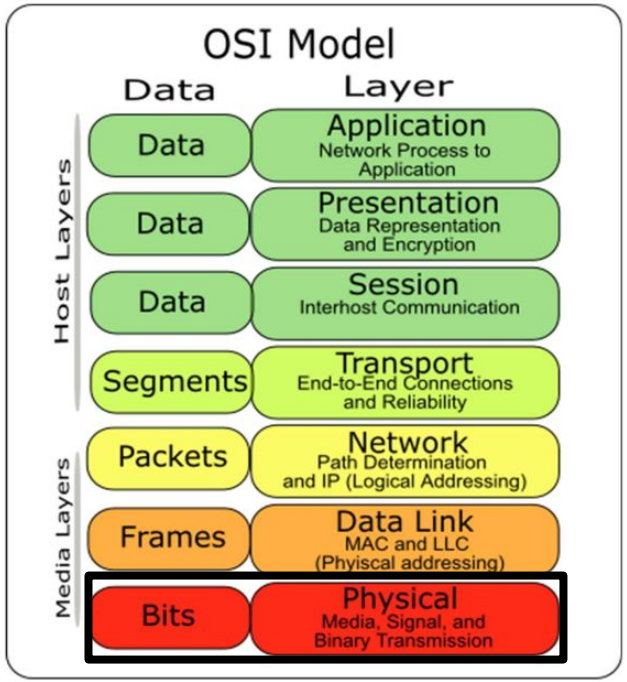
Understand what happens in Layer 2 and 3



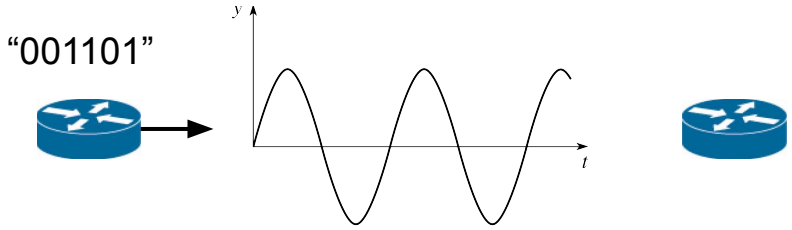
Today on NI labs

- Quick and dirty round-up of the OSI model
  - We will cover only aspects of interest for this course
- MAC and physical addressing
- IP subnetting & IP routing

Physical layer

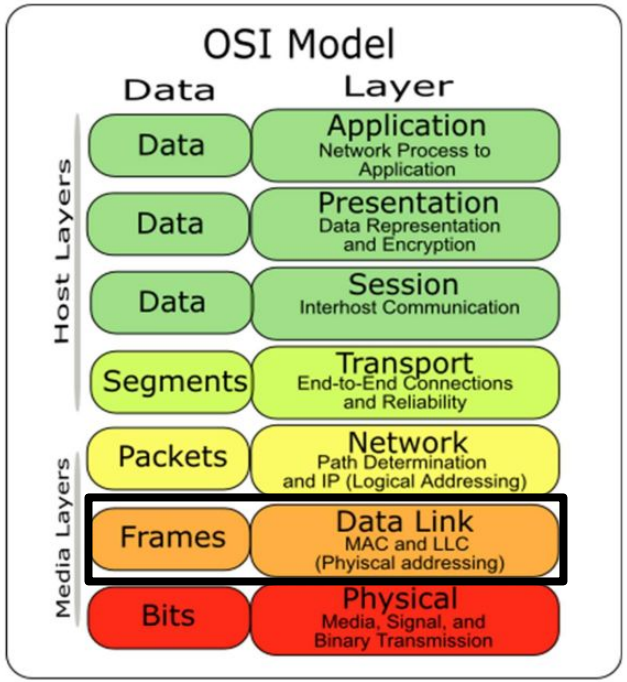


The Physical Layer (L1) is responsible to define how information is transmitted over a medium (copper wire, fiber optics, air, etc.) between two entities.



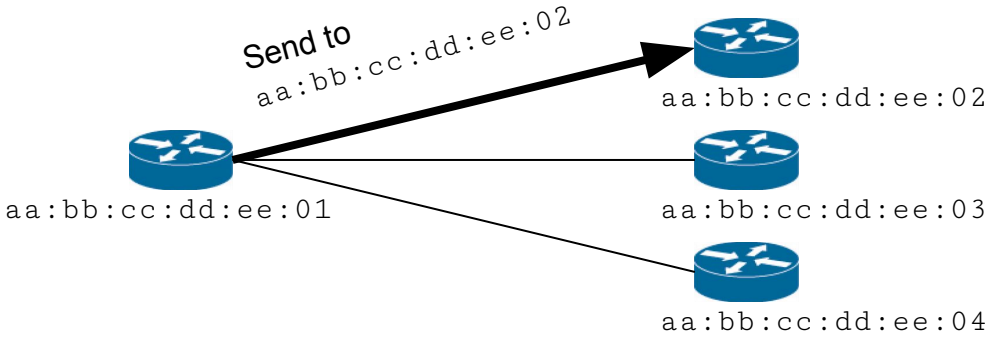
Physical layer

Link layer

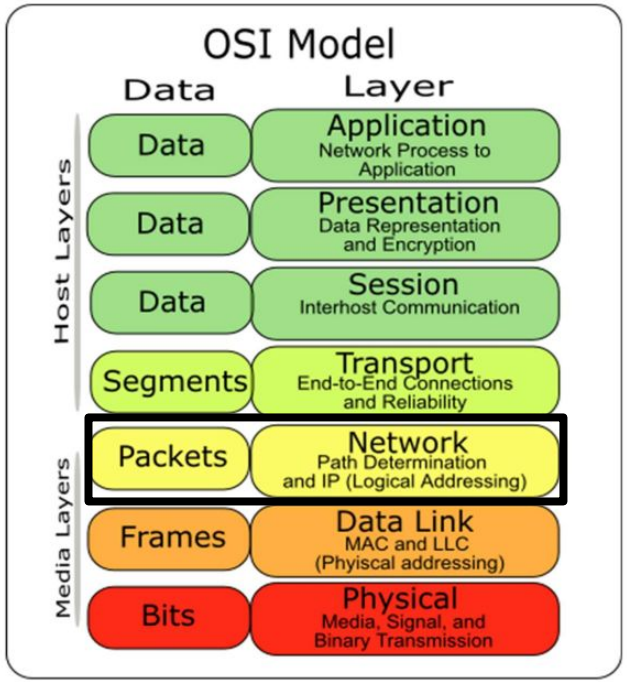


The Network Layer (L2) manages, among other things, physical addressing, i.e. addressing between **adjacent** nodes in the network.

Each network interface of every device has assigned a **unique identifier,** the *MAC* address.



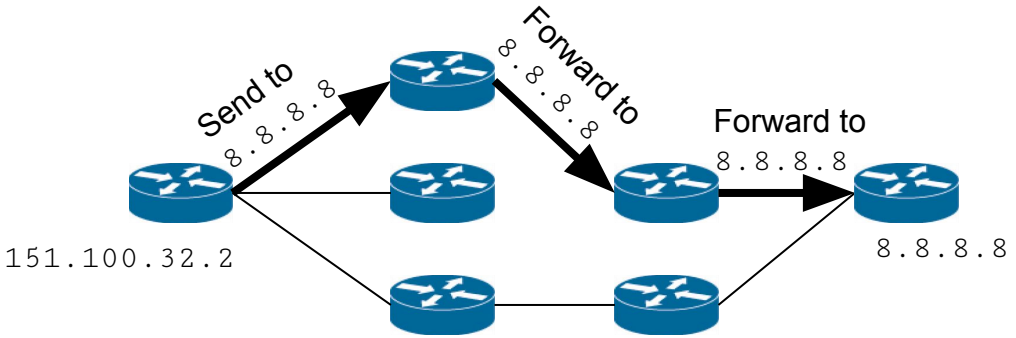
Network layer



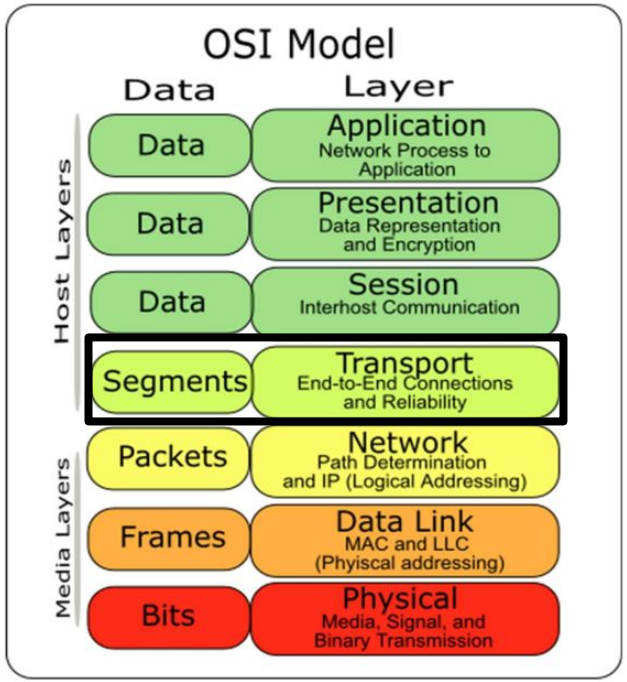
The Network Layer (L3) manages addressing between **distant** nodes, i.e. nodes that can't communicate directly.

Determines which path to take.

Each addressable node has a unique identifier, the *IP* address.



Transport layer



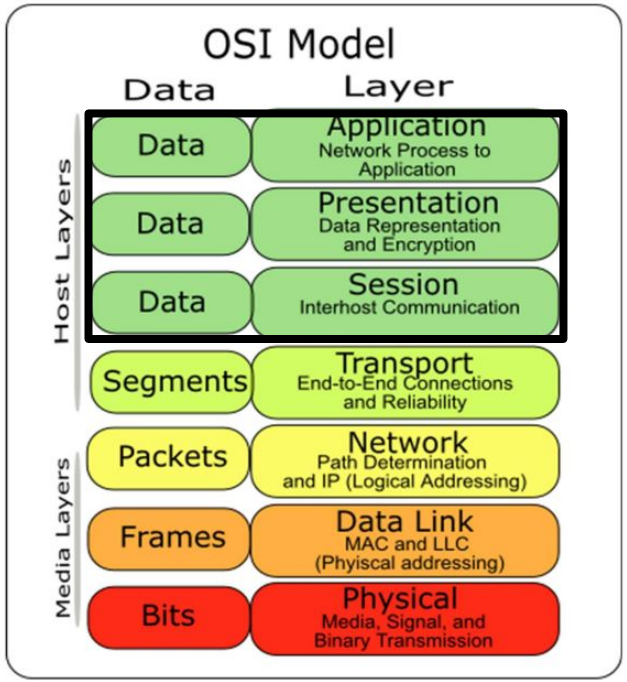
The Transport Layer (L4) transports data end-to-end adding **ports**, to multiplex the connections of a device.

TCP also adds packet acknowledgments, congestion control, counters to prevent out of order packets, etc…

Almost every well-known application protocol (HTTPS, MQTT, …) runs on top of the transport layer.

Netcat uses L4 protocols (TCP or UDP) to send arbitrary data between two hosts.

Application layers



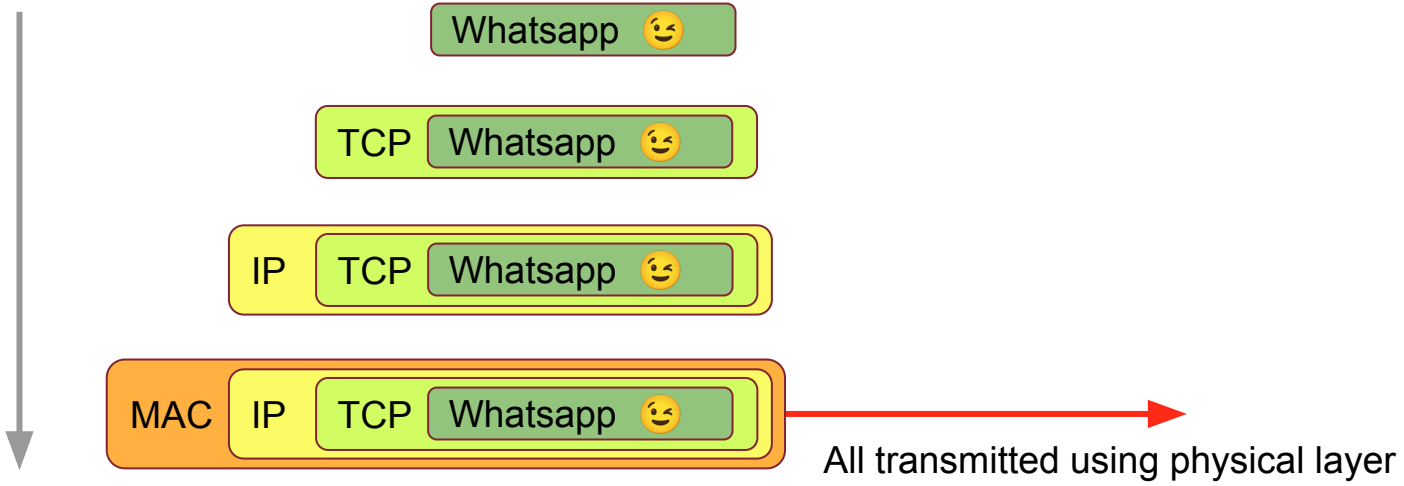
Application protocols (HTTP, FTP, MQTT) transmit user data end-to-end.

In between Transport and Application protocols, there could be protocols such as TLS/SSL to **secure** application packets.

Packet encapsulation

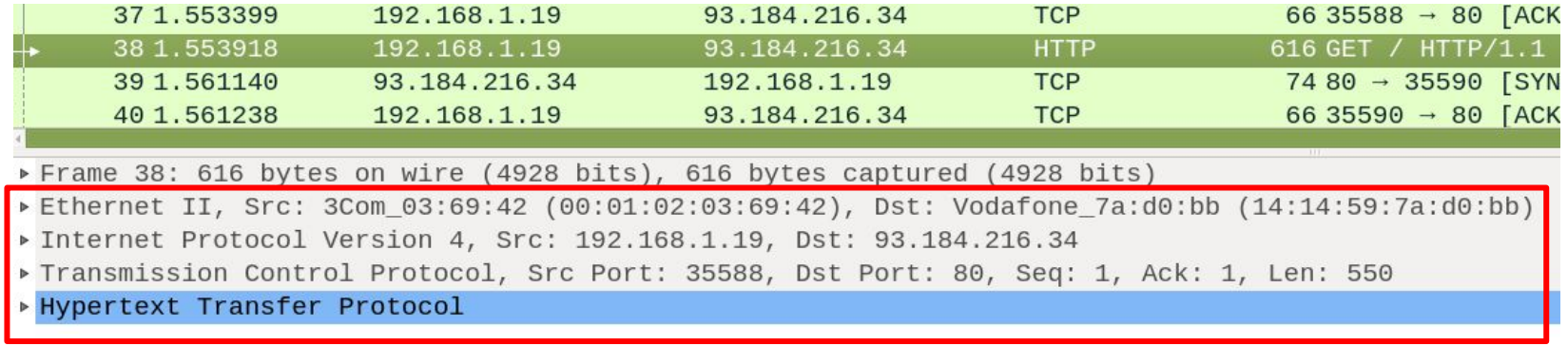
When sending a packet using a certain protocol in the stack layer, it gets encapsulated using all the lower layers!

Example: You want to send a "😉" to your friend via Whatsapp (application layer)

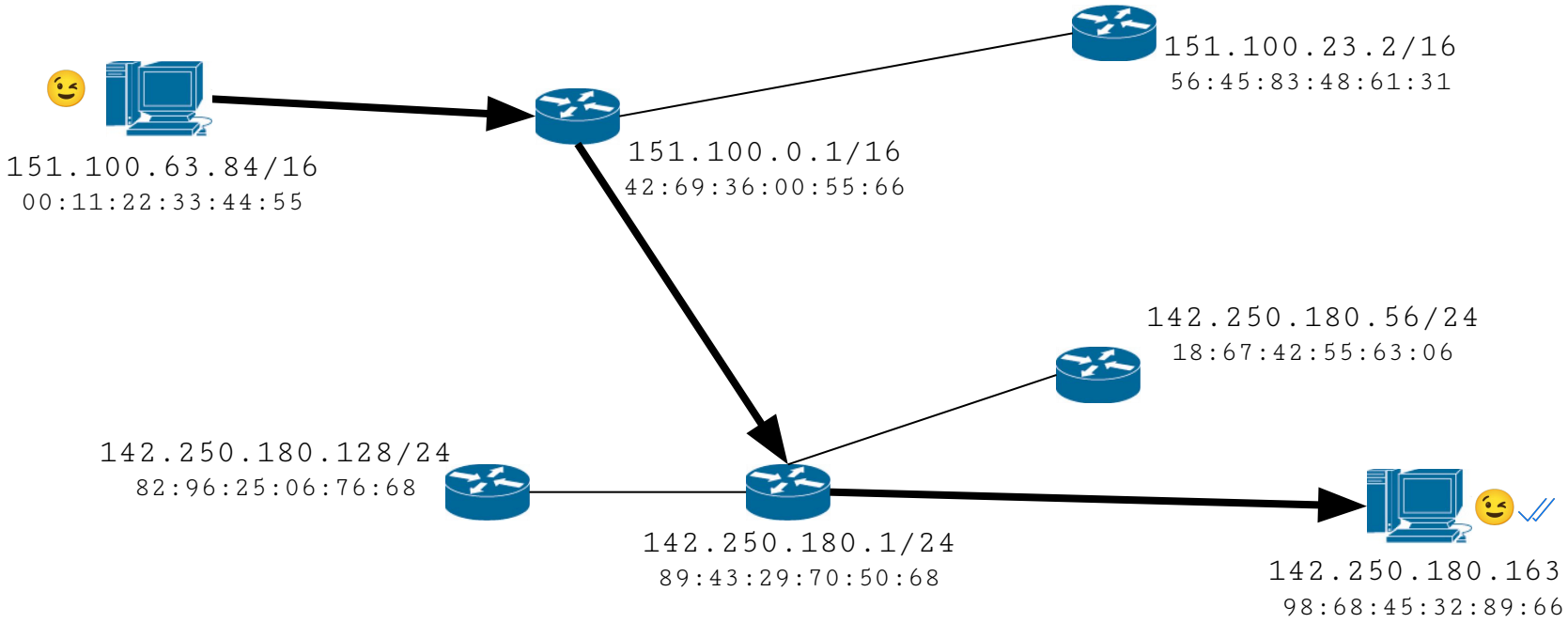


Packet encapsulation

Protocol encapsulation can be observed capturing packets with Wireshark



L2 and L3 addresses



Each node has (at least) one IP address (L3) *and* one MAC address (L2). **Why**?

L2 and L3 addresses

Each node has (at least) one IP address (L3) *and* one MAC address (L2). **Why**?

- MAC Addresses *uniquely* identify a network interface. Each manufacturer manages **its own pool of addresses**[1], and assigns one of them to each network interface. Constant throughout all the lifespan of the device
  - MAC addresses are used in physical addressing to uniquely identify a node in my neighbors
- IP addresses are assigned logically, in such a way to simplify the routing process. Each address belongs to a pool of addresses (subnet). May change over time.
  - IP addresses and subnetting enable routing

In the previous slide, we routed packets from `151.100.63.84` to `142.250.180.163`
It required 3 physical addressing, one for each “hop” in the routing

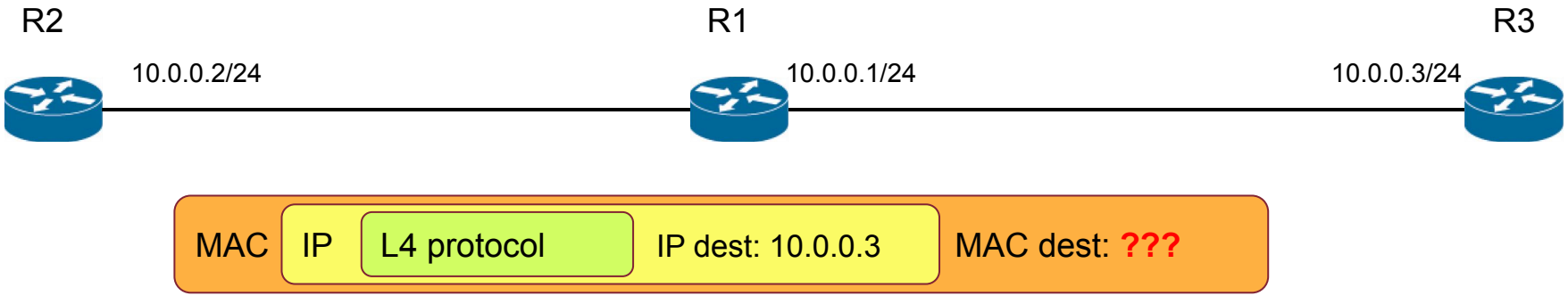
[1] https://macvendors.com/ finds the manufacturer of the network card given the MAC

ARP basics

ARP - *Address Resolution Protocol*

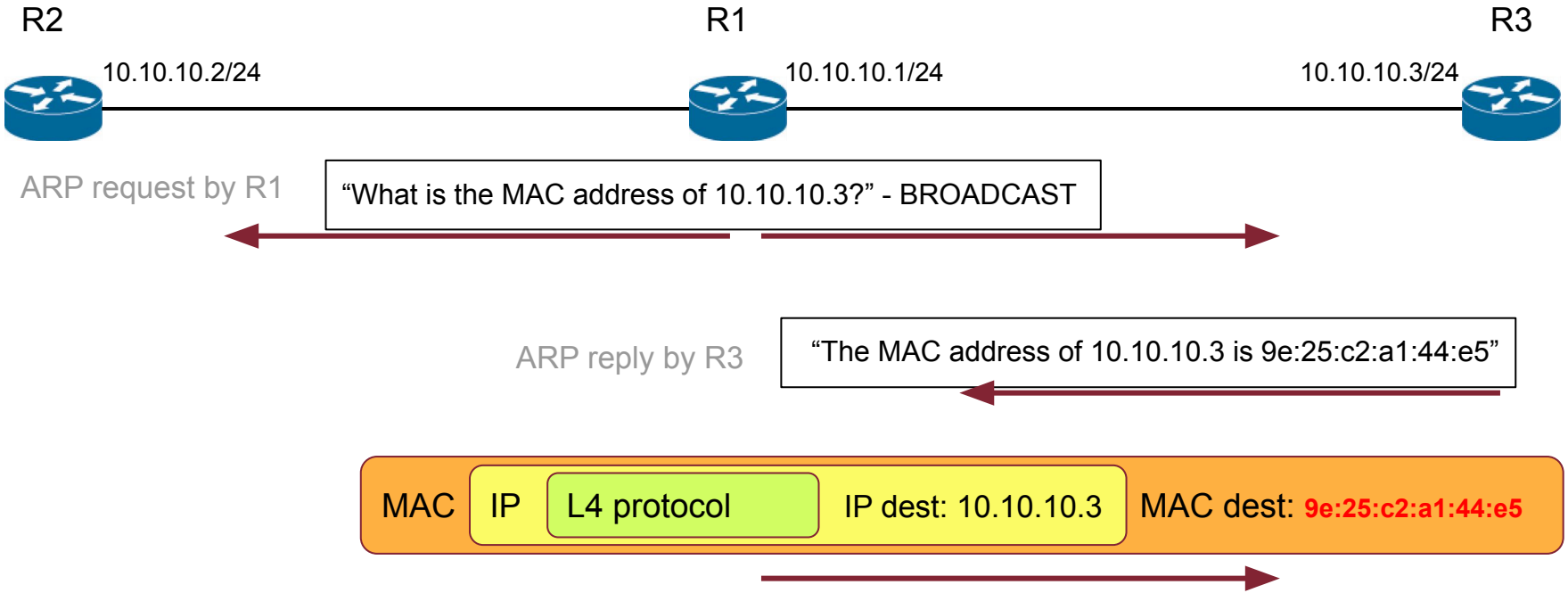
ARP is used to discover the link layer MAC address associated to a given internet layer IP address

Applications are required to specify the destination IP address to which send/receive data, they know nothing about MAC addresses



ARP basics

Capture in file arp_cap.pcap



ARP basics

The MAC address collected by ARP are stored in a **cache**, accessible with `ip neigh` command

```
192.168.1.1 dev wlp1s0 lladdr 14:14:50:7a:d0:bb DELAY
192.168.1.7 dev wlp1s0 lladdr 1e:80:eb:d2:28:e8 REACHABLE
192.168.1.250 dev wlp1s0 lladdr e4:4f:01:00:31:08 REACHABLE
192.168.1.8 dev wlp1s0 lladdr 44:08:0b:7e:68:22 REACHABLE
```

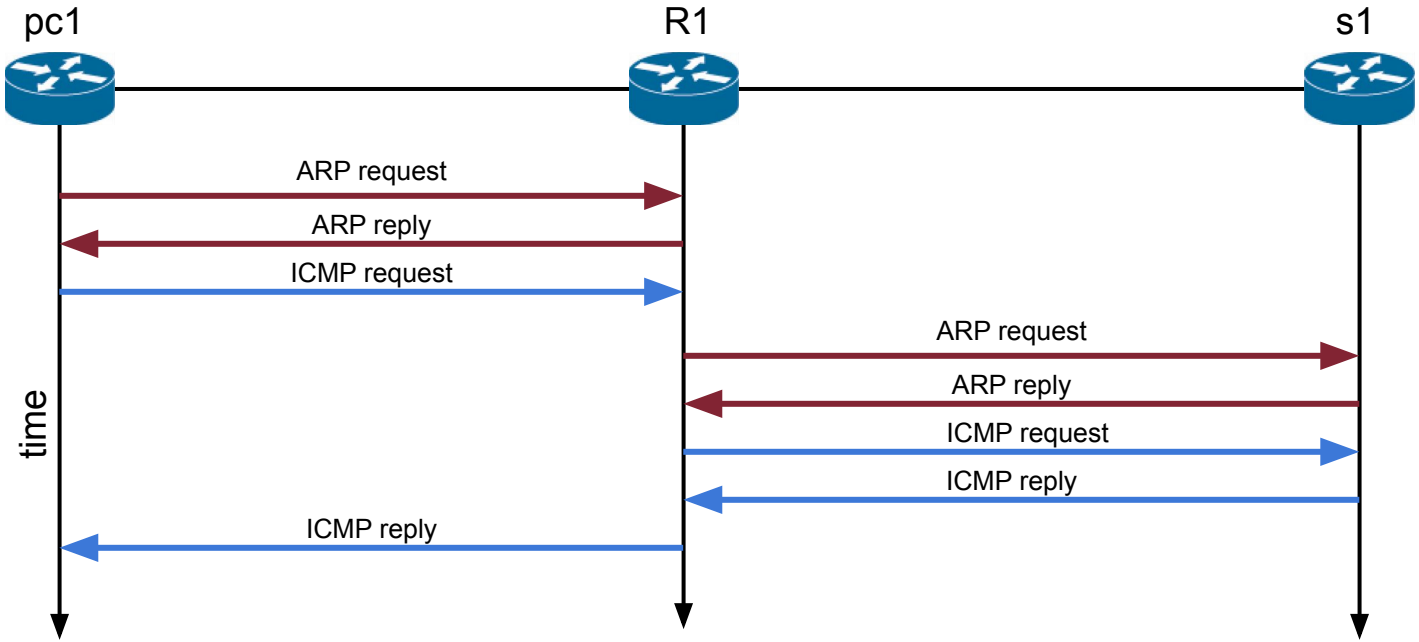
Each ARP entry has a state[1]. Transitions are regulated by timers[2]

Whenever we want to send a packet to some IP address, we first lookup its MAC address in the ARP cache. If there isn't: generate ARP request

[1] https://elixir.bootlin.com/linux/v5.14.7/source/include/uapi/linux/neighbour.h#L56 [2] https://elixir.bootlin.com/linux/v5.14.7/source/net/core/neighbour.c#L1022

ARP basics

When pc1 pings (ICMP request) s1 for the first time, we observe an ARP request and reply for each hop



ARP request

```
5 15.981663 9a:71:c0:f7:4… Broadcast      ARP      42 Who has 10.0.0.3? Tell 10.0.0.1
6 15.981712 9e:25:c2:a1:4… 9a:71:c0:f7:4… ARP      42 10.0.0.3 is at 9e:25:c2:a1:44:e5
```

```
▸ Frame 5: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▸ Ethernet II, Src: 9a:71:c0:f7:40:52 (9a:71:c0:f7:40:52), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)
```

Broadcast MAC address!

"What is the MAC address of 10.0.0.3? Tell to 10.0.0.1"

```
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 9a:71:c0:f7:40:52 (9a:71:c0:f7:40:52)
    Sender IP address: 10.0.0.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.0.3
```

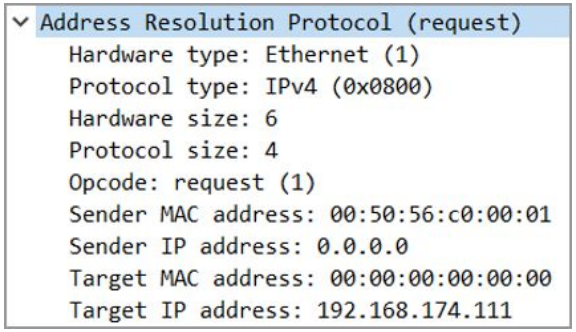
Sender IP and MAC address. Target IP address (which I know) and MAC address (unknown)

ARP reply

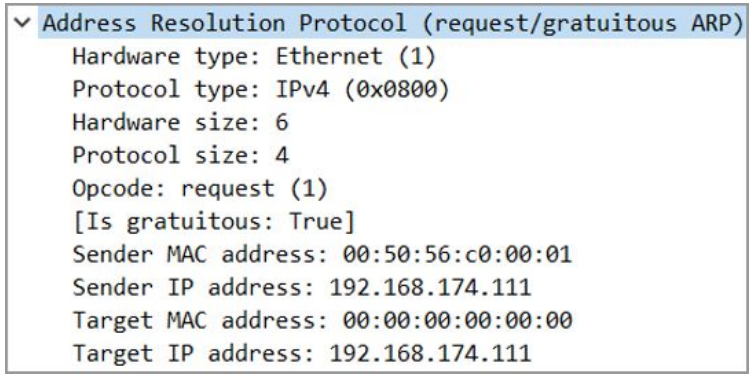
“The MAC address of 10.0.0.3 is 9e:25:c2:a1:44:e5”

```
▾Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: 9e:25:c2:a1:44:e5 (9e:25:c2:a1:44:e5)
  Sender IP address: 10.0.0.3
  Target MAC address: 9a:71:c0:f7:40:52 (9a:71:c0:f7:40:52)
  Target IP address: 10.0.0.1
```

Other ARP packets



ARP Probe
Checks if an IP address is in use in the network.
If yes, the owner sends back a Reply



ARP Announcement
Announce in the network which IP address is using the sender

https://www.practicalnetworking.net/series/arp/arp-probe-arp-announcement/

Security considerations - Spoofing

In theory: MAC addresses are assigned by the manufacturer and are immutable.
In practice: the linux command to change MAC address is:

```
ip link set eth0 address 00:11:22:33:44:55
```

⟶ ARP alone is not secure (ARP spoofing, ARP cache poisoning, ...)

Some WiFi and Ethernet networks may use MAC-based authentication.

- Any newly connected device is redirected to a login based (i.e., any new device with an unseen MAC address)
- By cloning (spoofing) the MAC address of a device which is already logged in, an attacker could bypass the login screen

Solution: network administrator should implement defenses against ARP attacks [1]

[1] https://en.wikipedia.org/wiki/ARP_spoofing#Defense

Security considerations - Tracking and localization

MAC addresses are unique values for every network interface of any device

→ MAC addresses could be used to track a device through space

Example: Wi-Fi based localization on smartphones

1. Devices send to their manufacturer the MAC addresses of near Wi-Fi access points
2. There exist databases[1] [2], even open ones, containing millions of MAC addresses with their position
3. By cross-referencing the MAC addresses sent by the device and the positions of known MAC addresses, you get a rough estimate of the device position

[1] https://www.mylnikov.org/archives/1170

[2] https://wigle.net/

Security considerations - Tracking and localization

MAC addresses are unique values for every network interface of any device

→ MAC addresses could be used to track a device through space

Researchers in [1] have decoded the encrypted diagnostic traffic sent by Android and iOS devices, in particular the traffic exchanged when a device turns on the localization services

For example, iPhones [1] with GPS localization enabled logs the MAC address of **all nearby devices** (connected to the same wifi) and send it to their servers with precise location

→ Someone knows your location without your consent!

A possible solution is MAC address **randomization**, implemented in most platforms but:

- it may be disabled by default
- I can randomize my MAC address as much as I want. But if then if my device tells to the manufacturer which MAC it currently has, goodbye randomization

[1] https://www.scss.tcd.ie/doug.leith/apple_google.pdf, presented at SecureComm 2021

IP subnetting

An **IPv4 address** is a 4 bytes-long identifier, expressed in dotted notation, from 0.0.0.0 to 255.255.255.255

- example: 192.168.1.100

A **subnet** or **prefix** is a logical division of IP addresses. It is a pool of IP addresses.

Examples:

- Subnet **192.168.1.0/24** - Contains addresses from 192.168.1.0 to 192.168.1.255
- Subnet **10.50.0.0/16** - Contains addresses from 10.50.0.0 to 10.50.255.255
- Subnet **10.10.10.4/31** - Contains addresses from 10.10.10.4 to 10.10.10.5

The number after the / is the netmask, the number of bits of the subnet

IP subnetting

Which IP addresses belong to a subnet?

- An IP belongs to a subnet if its first `netmask` bits match the subnet

Example, consider subnet 10.8.160.0/19, **which addresses does it contain?**

1.	IP Address decimal	10	8	160	0
2.	Netmask /19	11111111	11111111	11100000	00000000
3.	IP Address binary	00001010	00001000	10100000	00000000
4.	Lowest IP	00001010	00001000	10100000	00000000
5.	Highest IP	00001010	00001000	10111111	11111111

Fixed (Network ID) Variable (Host ID)

The subnet 10.8.160.0/19 contains IP addresses from 10.8.160.0 to 10.8.191.255

In total 2^{13} = 8192 distinct addresses

IP subnetting

How many IP addresses belong to a subnet?

Example: consider the subnet of the previous slide 10.8.160.0/19. We have 19 fixed bits (network ID) and 13 variable bits (host ID). Number of addresses is 2^{13}

HOWEVER, we don't assign two addresses:

1. The lowest address 10.8.160.0 indicates the subnet itself
2. The highest address 10.8.191.255 is the broadcast address

So the total number of assignable addresses is $2^{13} - 2 = 8190$

HOWEVER x2, by convention, all the 2 addresses of /31 subnets are assignable.
E.g. in a subnet 10.0.0.0/31 both 10.0.0.0 and 10.0.0.1 are valid

IP subnetting

To which subnet does an IP belong?

In other words, given an IP address and a netmask, how to find its subnet?

- You can obtain the subnet by applying the bitwise AND of the address and the netmask

Example: considering an IP address 10.8.162.52 and knowing that its subnet has netmask /19, **which is its subnet**? Steps:

1.	IP Address decimal	10	8	162	52
2.	IP Address to binary	00001010	00001000	10100010	00110100
3.	Netmask* /19	11111111	11111111	11100000	00000000
4.	Subnet binary	00001010	00001000	10100000	00000000
5.	Subnet to decimal	10	8	160	0

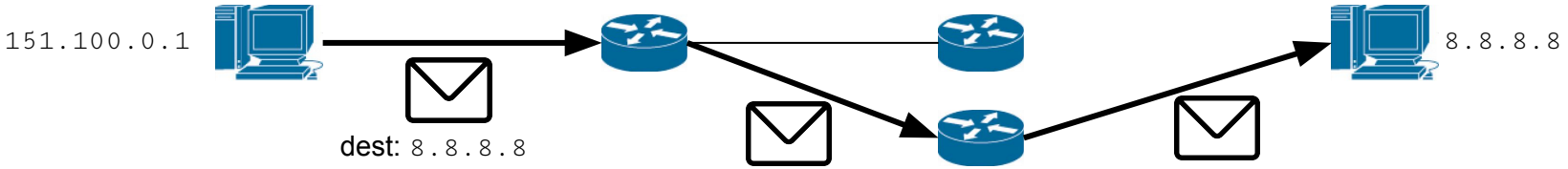
Address 10.8.162.52/19 belongs to subnet 10.8.160.0/19

* another way of expressing the netmask is in decimal notation. In this case it would be 255.255.224.0

IP routing

Why care about subnets? They enable routing!

Routing is the process of forwarding a packet towards its final destination



The nodes doing routing are routers. For each incoming packet they:

1. Lookup in a **Routing Table** and decide which is the next hop (another router or the final node)
2. Send the packet on right interface to the next hop

Routing table

Output of `ip route` command

Destination Subnet	Interface	Metric
Default	eth0, via 10.50.0.1	100
10.2.128.0/18	wlp1s0	600
10.50.0.0/16	eth0	100
169.254.0.0/16	eth0	1000

Generally, each entry in the routing table indicates how to reach a certain **subnet**

- not one specific IP address

Routing table

- Route to remote subnet via `next_hop_ip`:

  ```
  ip route add a.b.c.d/m via next_hop_ip
  ```

  - Detail: note that in the routing table, we store the *interface* to which forward a packet. But we did not specify an interface:
    - `next_hop_ip` is **itself routed recursively** to find its target interface!
    - If I don't know how to route to `next_hop_ip` i get *Network Unreachable*

- Route to local subnet which is directly reachable through eth0 :

  ```
  ip route add a.b.c.d/m dev eth0
  ```

- You can also specify both:
  - `ip route add a.b.c.d/m via next_hop_ip dev eth0`

Forwarding decision

How to forward a packet based on the routing table?

Let's call *A* the address of an incoming packet we want to route.

For each subnet *S* entry in the routing table, check whether *A* belongs to *S*:

- **Only one subnet matches:** forward the packet there
- **More than one subnet matches:** Longest prefix match
- **No subnet matches:** Is there a "default" entry?
  - Yes: forward it using the default gateway
  - No: *Network Unreachable* error. Translates to "I don't know how to reach that subnet"

Longest Prefix Match

What to do when more than one matching occurs in the routing table?

I don’t know the exact subnet of the destination address of the packet: i don’t have the netmask, only the IP address

→ Forward the packet to the matching subnet with the longest netmask

Example:

Address 192.168.1.100 matches with both 192.168.1.0/24 and 192.168.0.0/16.

However, the address matches with the first 16 bits of 192.168.0.0/16 and the first 24 bits of 192.168.1.0/24. Therefore the packet is forwarded towards 192.168.1.0/24.

Routing table - local and non local destinations

The routing table must differentiate between hosts that are **directly** reachable and hosts that aren't.

Example output of **`ip route show`**

```
default via 192.168.1.1 dev eth0
10.50.0.0/16 via 192.168.1.32 dev eth0
192.168.1.0/24 dev eth0 scope link
```

Here, hosts in subnet 192.168.1.0/24 are **directly** reachable through dev eth0.

Hosts in subnet 10.50.0.0/16 are **not** directly reachable, but they can be reached **via** 192.168.1.32

- i.e. in order to reach 10.50.0.0/16 I need to forward packets to 192.168.1.32

All other hosts (default) can be reached via 192.168.1.1

Why care? Because depending on the destination being local or not, it changes **ARP** requests being done.

Routing table - local destinations

```
default via 192.168.1.1 dev eth0
10.50.0.0/16 via 192.168.1.32 dev eth0
192.168.1.0/24 dev eth0 scope link
```

Example **local** destination:

Suppose that I have a packet to forward for host A, with destination IP_A 192.168.1.100.

It is **directly** reachable through dev eth0, so I can perform an ARP request to find the MAC address of 192.168.1.100, MAC_A

I can now send the packet on dev eth0, having as L2 address MAC_A and as L3 address IP_A

Routing table - non local destinations

```
default via 192.168.1.1 dev eth0
10.50.0.0/16 via 192.168.1.32 dev eth0
192.168.1.0/24 dev eth0 scope link
```

Example **non-local** destination:

Suppose that now host A, has destination IP_A 10.50.1.3.

It is reachable through via host B 192.168.1.32, so I can perform an ARP request to find the MAC address of 192.168.1.32, MAC_B

I can now send the packet on dev eth0, having as L2 address MAC_B and as L3 address IP_A

Routing example

Address to route:

151.100.25.63

Routing table contains the following entries:

A. 151.64.0.0/10
B. 151.96.0.0/11
C. 151.0.0.0/8
D. 151.112.0.0/12
E. 151.104.25.0/28

Which one would you chose?

Routing example

Subnets in routing table

Address to route	151.100.25.63	10010111	01100100	00011001	00111111
1)	151.64.0.0/10	10010111	01000000	00000000	00000000
2)	151.96.0.0/11	10010111	01100000	00000000	00000000
3)	151.0.0.0/8	10010111	00000000	00000000	00000000
4)	151.112.0.0/12	10010111	01110000	00011001	00000000
5)	151.104.25.0/28	10010111	01101000	00011001	00000000

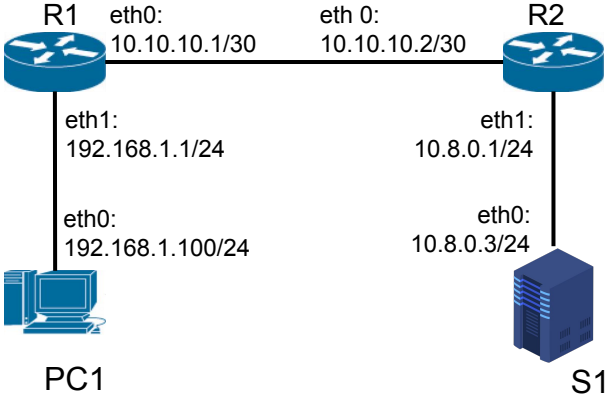
4), 5) DON'T MATCH the address

1), 2), 3) MATCH the address, however 2) matches the longest prefix, hence is selected

Proposed exercise: unreachable_destinations

In this lab we try to observe the network behavior in case of errors. Depending on the nature of the error we observe a different behavior, useful when debugging.

Launch unreachable_destinations lab, already configured, available now in the shared drive.



1. From pc1 try to ping an unreachable **local** destination, e.g. 192.168.1.7, in the same subnet of pc1:
   a. Capture on eth0 pc1
   b. Example: tcpdump -i et0 -w /shared/cap.pcap
2. From pc1 try to ping an unreachable **remote** destination, e.g. 10.8.0.20, in the same subnet of s1
   a. For this, capture packets on interface eth0 and eth1 of R2
3. From pc1 try to ping a non-existent **remote** destination, e.g. 172.16.0.1
   a. Capture on eth0 pc1
4. Observe with wireshark which ARP packets are exchanged in the various collision domains. Which errors do you observe?

tip: In kathará, if you need more than one terminal for a machine, e.g. pc1, you can: open a terminal on your computer, go into the lab folder, type `kathara connect pc1`