Basics of Networking 2: the Internet Protocol

Daniel STAN

May 20, 2024

Éléments de réseau 2

Contrib and Speakers: Eric Milard, Nidà Meddouri, Elloh Adja, Suzana Dedefa, Christian Diaconu

Version: v1.0, feedback and remarks to: `daniel.stan@epita.fr`

Introduction: Outline

In this class, you will see what made the success of the Internet and how protocols build on top of each others.
But also: how to secure them.

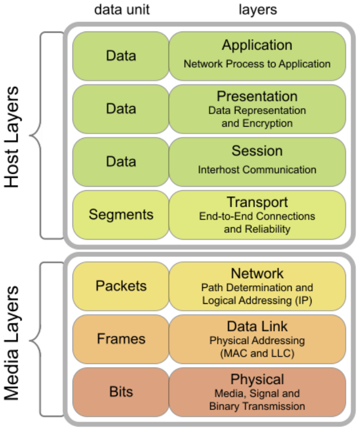


Figure: OSI Model (Credit: Wikipedia)

Role of Open Source Softwares

The Internet expansion is directly linked to the free and open source software expansion.

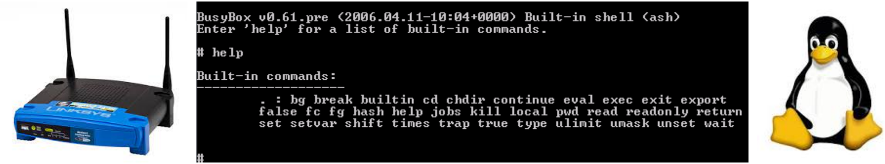
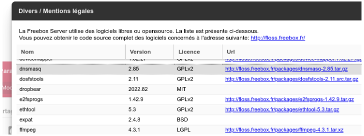


Figure: Example of the OpenWrt firmware (`https://en.wikipedia.org/wiki/OpenWrt#History`)

We will revisit classical concepts of networking through the use of **FOSS**[1].



Divers / Mentions légales

La Freebox Server utilise des logiciels libres ou opensource. La liste est présente ci-dessous.
Vous pouvez obtenir le code source complet des logiciels concernés à l'adresse suivante: http://floss.freebox.fr/

Nom	Version	Licence	Url
dnsmasq	2.85	GPLv2	http://floss.freebox.fr/packages/dnsmasq-2.85.tar.gz
dosfstools	2.11	GPLv2	http://floss.freebox.fr/packages/dosfstools-2.11.src.tar.gz
dropbear	2022.82	MIT	
e2fsprogs	1.42.9	GPLv2	http://floss.freebox.fr/packages/e2fsprogs-1.42.9.tar.gz
ethtool	5.3	GPLv2	http://floss.freebox.fr/packages/ethtool-5.3.tar.gz
expat	2.4.8	BSD	
ffmpeg	4.3.1	LGPL	http://floss.freebox.fr/packages/ffmpeg-4.3.1.tar.xz

Figure: Another example on a French ISP[2] modem

[1] Free and Open Source Software

[2] Internet Service Provider (🇫🇷 *FAI: Fournisseur d'Accès à Internet*)

Plan of the NET2 class

Themes:

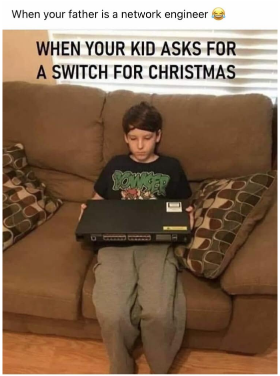
- IPv* subnetting and routing
- UDP and TCP
- Basics of Cryptography
- TLS
- Firewalling and NAT

4 Lab Classes (TP) with GNS3.

Final grade: 50% final exam (MCQ) + 50% Lab 1,2,3 grades[3].

[3] TP4 is not graded

Local Networks



Credit: @PR0GRAMMERHUM0R

Physical/Layer 1: Physical Interface

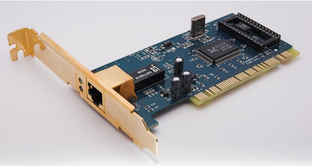


Figure: An Ethernet Gigabit PCI[4] Card

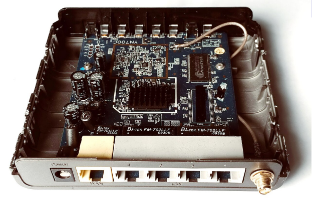


Figure: A Wireless (Home) Router, with its missing antenna

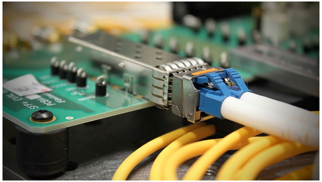


Figure: Optical Connection, through a SFP[5], or "Mini-GBIC"

Every physical medium has its specific communication protocol (wavelength, collision avoidance mechanisms, modulation, bandwidth, ...).

Credit: https://en.wikipedia.org/wiki/Wireless_router#/media/File:Wireless_router,_internal_components_(LevelOne_WBR-6002).jpg
https://commons.wikimedia.org/wiki/File:SFP_board_2.jpg
https://fr.m.wikipedia.org/wiki/Fichier:Ethernet_NIC_100Mbit_PCI.jpg

[4] Peripheral Component Interconnect
[5] Small form-factor pluggable

Link/Layer 2: The Ethernet Frame (🇫🇷 *trame Ethernet*)

Basic **unit** of transmitted data.

MAC Addresses

Tells the type of

	Destination	Source	(Optional) 802.1Q tag	Ethertype	Payload	Checksum
Size (bytes)	6	6	4	2	42-1500	4

A bit of trivia:

- The maximal size of the payload is called the maximal transport unit (MTU).

Part of layer 1

- There are 7 bytes of preambles (10101010 repeated 7 times), then one byte of Start Frame Delimiter (10101011).
- There is also an InterPacket Gap (IPG) of 12 bytes afterwards.
- Layer 2 devices (switches) **do not look into** the payload (nor the ethertype).

On Linux: the `ip link` command

For short: `ip l`.

```
dstan@flan:~$ ip l
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group default ql
    link/ether 08:92:04:34:1a:a0 brd ff:ff:ff:ff:ff:ff
4: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DORMANT group default qlen 10
    link/ether c4:75:ab:a5:39:17 brd ff:ff:ff:ff:ff:ff
```

Device Name — MAC Address — MTU — Status (connected **and** started, or not)

NB: **lo** stands for **loopback**, a dummy interface.

Exercise: Recalls about hexadecimal

- 1 bit (b) $= 0$ or $1 = \boxed{2}$ possibilities;
- 1 byte (B) (🇫🇷 *octet*) $= 8$ bits $= \boxed{2^8 = 256}$ possibilities;
- 1 hexadigit $= 0, 1 \ldots 9, a, b, c, d, e, f = \boxed{16 = 2^4}$ possibilities;
- 2 hexadigits $= 16^2$ possibilities $= (2^4)^2 = 2^8$ possibilities.

Conclusion: 2 hexadigits $=$ 1 byte.

Ethernet MAC Address

MAC means **Medium Access Control** (here medium = the communication medium).

$$\overbrace{\underbrace{\mathsf{xx} : \mathsf{xx} : \mathsf{xx}}_{3\ \textbf{bytes} = \text{OUI}[6]} : \mathsf{xx} : \underbrace{\mathsf{xx}}_{1\ \textbf{byte} = 2\ \textbf{hexa}\text{digits}} : \mathsf{xx}}^{6\ \textbf{bytes} = 48\ \textbf{bits}}$$

A MAC Address is supposed to be **globally** (🇫🇷 *mondialement*) **unique**
⇝ every Manufacturer/Vendor is given **prefixes of 3 bytes** (OUI). It assigns suffixes **without collision** to every produced device.

Online DB: `https://maclookup.app` or Offline : `/usr/share/ieee-data/oui.csv` (package "`ieee-data`").

Other notations: **12:34:56:76:9A:BC**, **123456-769abc**, **1234.5676.9ABC** ...

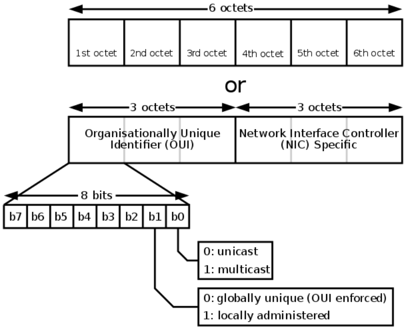
[6] Organizationally Unique Identifier

Non-unique addresses

Convention: Bit 2 of first byte is 1 *iff* address is **not globally unique**.

Some MACs are specifically **not unique**:

- Virtual Machine: who is the manufacturer ?
- Virtual Interfaces (tunnels, bridges, docker);
- Extra WiFi SSID for one AP;
- Randomized WiFi Address for Scanning (Privacy).
- Broadcast addresses (**ff:ff:ff:ff:ff:ff**).



Credit: Inductiveload, modified/corrected by Kju — SVG drawing based on PNG uploaded by User:Vtraveller. This can be found on Wikipedia here., CC BY-SA 2.5, `https://commons.wikimedia.org/w/index.php?curid=1852032`

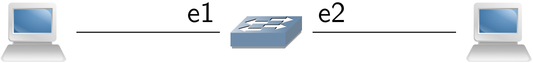
Exercise: Which ones are unique?

```
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group default ql
    link/ether 08:92:04:34:1a:a0 brd ff:ff:ff:ff:ff:ff
3: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether fa:16:3e:4a:19:8e brd ff:ff:ff:ff:ff:ff
    altname enp0s3
4: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DORMANT group default qlen 10
    link/ether c4:75:ab:a5:39:17 brd ff:ff:ff:ff:ff:ff
5: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default qlen 1
    link/ether 42:54:00:47:85:27 brd ff:ff:ff:ff:ff:ff
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:00:04:db:b5:0e brd ff:ff:ff:ff:ff:ff
12: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/none
45: enx08920483f152: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group def
    link/ether 08:92:04:83:f1:52 brd ff:ff:ff:ff:ff:ff
```

LAN switching (*commutation*)

A switch = 

A switch can connect computers on different ports:



Switches can be chained:

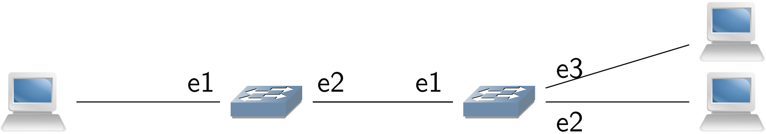


Figure: A typical cisco switch

Switches don't have a MAC address (they're just "intelligent wires")

Switching Table and Broadcast

Goal: send frames only to the correct destination.

How: map each known MAC address to a **switch** port.

Port	Mac Address of
e1	M1
e2	M2
e2	M3

Port	Mac Address of
e1	M1
e3	M3
e2	M2

Whenever a machine emits a frame **for the first time**, the receiving switch learns the mac address and the associated port.

⚠ Broadcast *may* still happen: if the table is full (*MAC flooding attack*), or for some destination addresses (broadcast on **ff:ff:ff:ff:ff:ff**), or for some other devices (hubs, WiFi hotspots ...).

IP Addressing

title-ip

IP/Layer 3: Motivation

“What if I change my network adapter?”

- Need to notify everyone who contacts my computer.
- Need to notify all **switches**.
- MAC addresses are hard to remember.

Solution: one address of 32 bits.

Notation: 4 decimal numbers separated by dots: **172.67.71.150**

Alternative Notation: plain 32 bit number (no one uses it)

```
dstan@flan:~$ ping 134744072
PING 134744072 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=111 time=13.1 ms
```

The Linux command: `ip address`

For short: `ip a`

```
dstan@flan:~$ ip -4 a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    inet 10.117.254.177/16 brd 10.117.255.255 scope global dynamic noprefixroute wlp0s20f3
       valid_lft 25565sec preferred_lft 25565sec
5: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
       valid_lft forever preferred_lft forever
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
12: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 10.11.0.15/32 scope global wg0
       valid_lft forever preferred_lft forever
```

NB: a machine could have more than one IP, even on the same interface.

Local IP network in CIDR notation

On a **local** network, one can talk with all the addresses having the same **prefix**:

172 .	17 .	0 .	1	/19
10101100	00010001	00000000	00000001	

prefix part host part

Example here:

- **172.17.31.1** is another IP on **the same network**;
- **172.17.13.37** is another IP on **the same network**;
- **172.17.32.2** is an IP on **another network**;
- **172.17.0.0** is the first IP of the range, it **cannot be used as an address**: this is the **prefix**.
- **172.17.31.255** is the last IP of the range, it **cannot be used as an address**: this is the **broadcast address**.

Terminology: **172.17.11.11/19** denotes an IP address with a prefix length of 19.
This address is part of the **172.17.0.0/19** (sub)net.

The first usable address of this net is **172.17.0.1**, the last one is **172.17.31.254**.

CIDR vs Masks: Practical Implementation

How to check if two IP addresses belong to the same subnet?

```c
int samesubnet(uint32_t ip1, uint32_t ip2, uint32_t preflen) {
  uint32_t mask = 0xffffffff; //                         111...111
  mask >>= (32 - preflen);    // shift to the right:         11..11
  mask <<= (32 - preflen);    // shift back to the left: 11..11000

  return (ip1 & mask) == (ip2 & mask); // bitwise and
}
```

Masks can be used instead of prefix lengths. Classical masks from **classes** of subnets:

- Class A = "/8" = **255.0.0.0**;
- Class B = "/16" = **255.255.0.0**;
- Class C = "/24" = **255.255.255.0**;

The notion of class is **legacy**, we use arbitrary prefix lengths nowadays (**C**IDR=**C**lassless).

Exercise: fill in the blanks

1. **192.168.42.1** belongs to subnet ☐ **/24** with mask ☐ and it contains ☐ usable addresses;
2. **8.229.151.87** belongs to ☐ **/12** with mask ☐ and it contains $2^{\square} - 2$ addresses.
3. ☐ is the first usable address of subnet **95.88.0.0/** ☐ with mask **255.255.252.0** and it contains ☐ usable addresses;
4. **62.129.255.254** is the last IP of ☐ with mask **255.255.224.0**;
5. A network mask, in IPv4 notation, can only contain 9 different numbers: ☐

IPv4 in Ethernet, the full picture

Ethernet Frame:

Destination	Source	802.1Q tag	Ethertype	Payload	Checksum
6B	6B	4B	2B: 0x0800	≤1500B	4B

IP Packet:

Version	...	Length	...	TTL	Protocol	Checksum	Source	Dest	Payload
0.5B	...	2B	...	1B	1B	2B	4B	4B	≤ 65535B

TCP = 6, UDP = 17, ICMP = 1

IP Addresses

- Source is before Destination;
- Checksum is computed without the header part;
- IP maximum payload is larger than Ethernet maximum payload;
- IP Packets may be carried by other protocols than Ethernet;

Address Resolution Protocol (ARP): IP → MAC?

List known IP/MAC correspondance, on Linux: `ip neighbour` (or `ip n` for short)

```
dstan@kugel:~$ ip -4 neigh
192.168.0.10 dev eth0 lladdr b8:27:eb:51:6d:12 REACHABLE
192.168.0.1 dev eth0 lladdr 90:5c:44:76:79:74 REACHABLE
192.168.0.8 dev eth0 lladdr 0e:d9:fa:55:6d:fb STALE
192.168.0.2 dev eth0  FAILED
```

ARP requests are done on the fly by the kernel, when trying to talk with an IP on local network.
One can force an ARP request with: `arping`:

```
dstan@kugel:~$ sudo arping -I eth0 192.168.0.1
ARPING 192.168.0.1
60 bytes from 90:5c:44:76:79:74 (192.168.0.1): index=0 time=951.237 usec
```

⚠ ARP is an **Ethernet** protocol (Ethertype: 0x0806), `arping` ≠ `ping` (icmp protocol is an **IP protocol**). ⚠

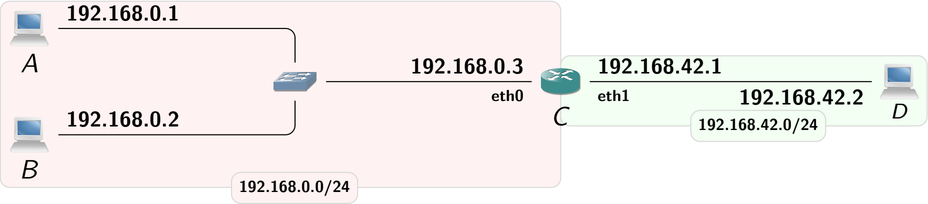
IP Routing

Routing Motivation: The *inter*net protocol

Ethernet with IP (ARP with broadcast): does not scale.
Solution: split the network into **sub networks**.

Routers Between Networks

Any machine connected **on two** (or more) networks, willing to **forward** packets.



- A can directly send a packet (src=**192.168.0.1**, dst=**192.168.0.2**) to B.
- A can directly send a packet (src=**192.168.0.1**, dst=**192.168.0.3**) to C.
- For A to send a packet to D, send a packet to C (src=**192.168.0.1**, dst=**192.168.42.2**). Then C sends the packet on its second interface (again, src=**192.168.0.1**, dst=**192.168.42.2**).

The same pair of IPs is used, but two different mac addresses of C are involved (if ethernet).

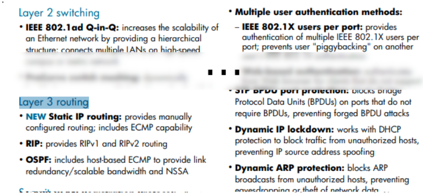
Routers in the wild

On Linux, just tell the kernel it should forward packets:

```
root@flan:~# echo "1" > /proc/sys/net/ipv4/ip_forward
```

 = . Some switches do routing:

Figure: A Cisco Sf300 switch ... and router!



Layer 2 switching

- **IEEE 802.1ad Q-in-Q:** increases the scalability of an Ethernet network by providing a hierarchical structure; connects multiple LANs on high-speed

...

Layer 3 routing

- **NEW Static IP routing:** provides manually configured routing; includes ECMP capability
- **RIP:** provides RIPv1 and RIPv2 routing
- **OSPF:** includes host-based ECMP to provide link redundancy/scalable bandwidth and NSSA

- **Multiple user authentication methods:**
  - **IEEE 802.1X users per port:** provides authentication of multiple IEEE 802.1X users per port; prevents user "piggybacking" on another

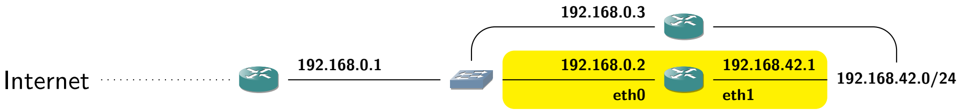
...

- **STP BPDU port protection:** blocks Bridge Protocol Data Units (BPDUs) on ports that do not require BPDUs, preventing forged BPDU attacks
- **Dynamic IP lockdown:** works with DHCP protection to block traffic from unauthorized hosts, preventing IP source address spoofing
- **Dynamic ARP protection:** blocks ARP broadcasts from unauthorized hosts, preventing eavesdropping or theft of network data ...

Figure: Datasheet of the *HP ProCurve *Switch* 6200yl-24G-mGBIC*

but dedicated hardware also exists...

Routing table



For each subnet, what interface to use and to what machine to talk to.

Target	Router	Interface	Source	Metric
default	**192.168.0.1**	-	-	-
192.168.0.0/24	-	**eth0**	**192.168.0.2**	
192.168.42.0/24	-	**eth1**	**192.168.42.1**	1
192.168.42.0/24	**192.168.0.3**	**eth0**	**192.168.0.2**	10

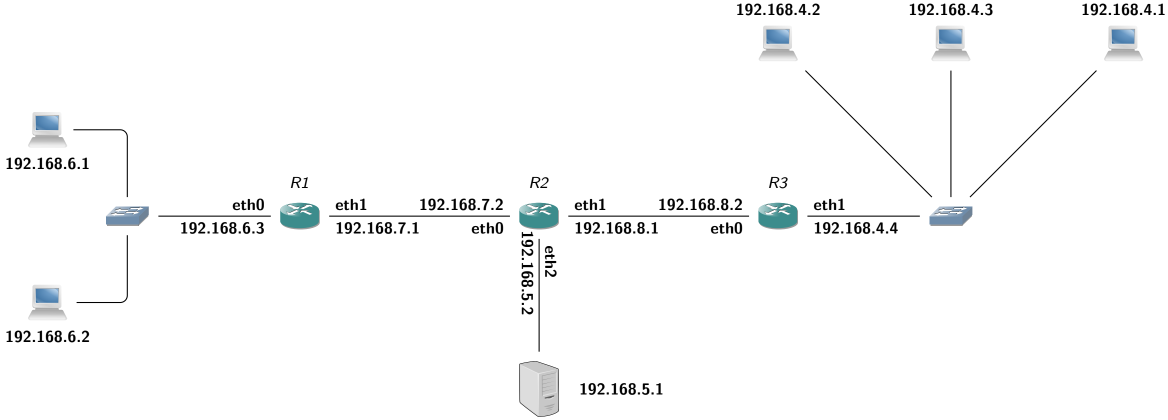
Default route

Local links

Fallback route

- Interface and source IP need to be filled only for local networks;
- Some systems use masks instead of CIDR notation;
- If two rules apply, use the **most specific** one;
- Two rules are as specific, use the one with the smallest metric;
- **default** is a shortcut for **0.0.0.0/0**, sometimes called *gateway* (🇫🇷 *passerelle*)

Exercise: Compute Routing Table for R1



Routing Table on Linux: `ip route` command

For short: `ip r`

```
dstan@kugel:~$ ip r
default via 192.168.0.1 dev eth0 proto dhcp metric 100
10.8.0.0/24 via 10.8.0.2 dev tun-vpn
10.8.0.2 dev tun-vpn proto kernel scope link src 10.8.0.1
192.168.0.0/24 dev eth0 proto kernel scope link src 192.168.0.4 metric 10
```

NB: Every time an **address** is added ("`ip a`"), the corresponding local **route** is automatically added:

```
dstan@flan:~$ sudo ip address add 192.168.42.1/24 dev wlp0s20f3
[sudo] password for dstan:
dstan@flan:~$ ip route
...
192.168.42.0/24 dev wlp0s20f3 proto kernel scope link src 192.168.42.1
```

Routing on the Internet

- Routing one packet is **fast**
- Computing the routing table is **tedious**
- By hand: **error-prone**
- On the Internet: **distributed algorithm** are used.
- NB: slow convergence (order of magnitude: min, hrs).

An autonomous system (AS) is provided with globally unique network prefixes, and is in charge of announcing them on the Internet (usually using BGP).
Inside an AS, the administrator chooses an architecture, and split the assigned prefixes however they want (static or dynamic routing, for example with OSPF[7]).

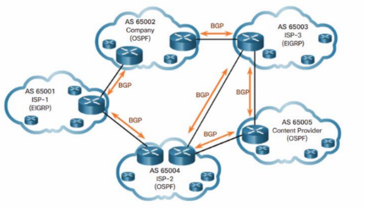


Figure: Interconnected Systems

NB: one can explore the BGP[8] interconnections (*peering*) with an online tool such as `https://bgp.he.net/`. Start for example with AS 212489 (EPITA), who owns **91.243.117.0/24**.

[7] Open Shortest Path First
[8] Border Gateway Protocol

Exercise: Subnetting (part 1)

1. A French university, that shall remain anonymous, received the IP subnet **138.231.0.0/16**.
   How many addresses does this prefix contain?

2. Every teaching department, research lab or other entity is assigned (by the IT dept) a different (sub-)subnet. For example:
   - The computer science department: **138.231.81.0/24**;;
   - The subnet of the university websites: **138.231.176.0/24**;
   - The subnet of the student dorms (🇫🇷 *Le "Crous"*) **138.231.136.0/21**;
   - To connect all these subnets, another internal subnet is used: **138.231.132.0/24**.

   How many addresses were allocated to the student dorms?

3. Draw the decision tree, reading the third byte of an IP address and deciding whether it belongs to **138.231.136.0/21**.

Exercise: Subnetting (part 2)

4. The student council (in charge of the dorms' network) needs to keep an handful number of addresses for its own servers (100 machines). Give a sufficient, but as **small as possible**, subnet to contain the servers.

5. Draw the decision tree reading the last 11 bits of an IP address and decides whether it belongs to the subnet of the previous question.

6. Infer from the previous question the different subnets to allocate to the student devices such that they **don't overlap** with the server subnet.

7. Additionally to **138.231.136.0/21**, the student council also gets the next prefix **138.231.144.0/21**. Can these two prefixes be summarised as a single subnet?

Exercise: Subnetting (part 3 – Epilogue)

NB: In the past, universities commonly got assigned **/16** addresses ("class B") by their Regional Internet Registry (RIR)

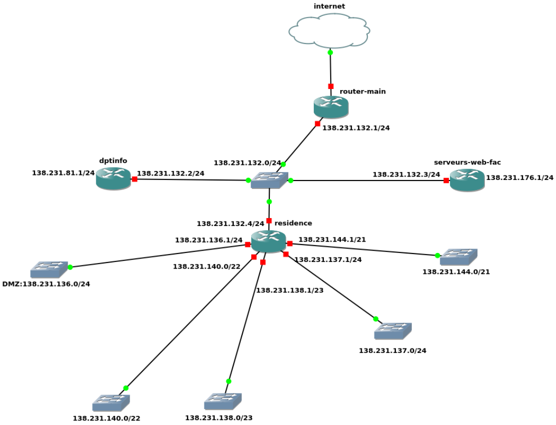


Figure: Sketch of the University network, as seen in GNS3

Routing Summary

- Routing is **fast** but computing routing tables is **slow** (BGP protocol, not discussed here).
- **Subnetting**: complex if we want to optimize used addresses $\rightsquigarrow$ **less granularity**.
- IP addresses are globally unique ressources assigned by a **central authority**:

$\xrightarrow{\text{assign prefixes}}$ RIR $\xrightarrow[\text{and prefixes}]{\text{assigns AS numbers}}$ LIR $\xrightarrow{\text{give addresses}}$ customers

, , . . .

Figure: Administrative assignement of IP addresses worldwide (Internet Assigned Numbers Authority, Regional Internet Registries, Local Internet Registries)

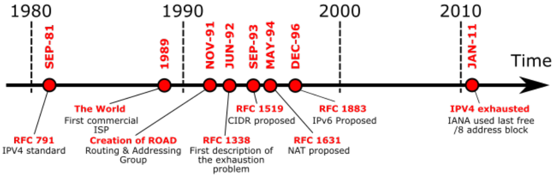
IPv6

Source: `https://www.worldipv6launch.org/downloads/`

Motivations: IPv4 Depletion

IPs should be unique to communicate with one another.
IPv4 is 4 bytes, ie 2^{32} addresses (4 billions). In reality, way less because of:

- **Reserved** subnets (non-routable, multicast, etc).
- **Granularity** due to subnetting/routing.
- **Hoarding**.



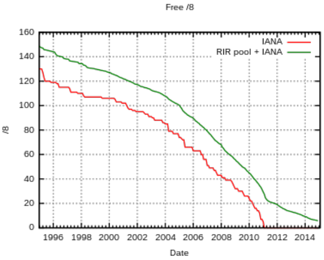


Figure: IPv4 exhaustion timeline (Credit: `https://en.wikipedia.org/wiki/IPv4_address_exhaustion`)

How to overcome the shortage

- **More granularity**: switching from classes (/8,/16 or /24), to **Classless** Inter-Domain Routing (CIDR) with any prefix length;
- Use of private addresses for personnal computer: **192.168.0.0/16** (used for /24 networks), **10.0.0.0/8** (for /8 nets), **172.16.0.0/12** (for /16).
- These IPs are not globally unique: they can only be used on a local network, not on the Internet: we need **Network Address Translation** (NAT) $\rightsquigarrow$ **more complex routers** (last lectutre).

And still, this is **not sufficient**.

How to overcome the shortage (con't)

IPv6 = IPv4 but with **128 bits**

Backward compatible on upper and lower layers:

- $\uparrow$ For **users**: Protocols on top of IPv4 (TCP, UDP, see next lecture) also work on top of IPv6;
- $-$ For **network engineers**: Subnetting and routing work in the same way;
- $\downarrow$ For **Ethernet**: IPv6 uses layer 2 protocols in a similar fashion (ARP $\rightarrow$ NDP[9])

We will just discuss some nice features of IPv6 now.

[9] Neighbor Discovery Protocol

IPv6 Addresses

Here is an address:

$$\overbrace{\underbrace{2a}_{\text{1 byte}}00:\underbrace{1450}_{\text{2 bytes}}:4007:081a:0000:0000:0000:0003}^{\text{16 bytes} = 8 \times 8 \times 2 = \text{128 bits}}$$

Some shortcuts:

- Trailing zeroes (most significant hex) in each group can be omitted:
  2a00:1450:4007:~~0~~81a:~~000~~0:~~000~~0:~~000~~0:~~000~~3 $\rightarrow$ **2a00:1450:4007:81a:0:0:0:3**
- One continuous sequence of zeroes can ommited: **2a00:1450:4007:81a:~~0:0:0~~:3** $\rightarrow$ **2a00:1450:4007:81a::3**

IPv6 on Linux

Parallel network stack. `ip -6 a` command.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state UNKNOWN qlen 1000
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UP qlen 10
    inet6 2a01:e0a:507:3290:173d:55db:8010:72a4/64 scope global temporary
       valid_lft 86256sec preferred_lft 84152sec
    inet6 2a01:e0a:507:3290:e8c7:50b9:f194:e829/64 scope global dynamic n
       valid_lft 86256sec preferred_lft 86256sec
    inet6 fe80::b55c:170b:d157:e7b/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

NB: it is very common to have more than one IP address, even on the same device.

IPv6 Subnets and Routes

IPv6 routing works exactly as IPv4: `ip -6 r`

```
::1 dev lo proto kernel metric 256 pref medium
2a01:cb04:1cb:3f00::/64 dev wlp0s20f3 proto ra metric 600 pref medium
fe80::/64 dev wlp0s20f3 proto kernel metric 1024 pref medium
default via fe80::aecf:7bff:fe14:cd30 dev wlp0s20f3 proto ra metric 600 p
```

- CIDR notation: no-one uses masks (except implementation)
- ✓For our sanity: the prefix length is always multiple of 4 (why?)
- ✓For our sanity again: no prefix larger than 64 (except for /128 for a point to point connection)
- Usually, an end customer of an ISP offers *at least* one /64. [10]

[10] RFC7368: "it is highly preferable that the ISP offers at least a /56"

Exercise: IPv6 alternative notation – opt

Prefix **/96** + IPv4 address (**32 bits**) = **128 bits** = one valid address.

Example (Exercise)

- **2001:0db8:6464::/96** + **140.82.121.3** → **2001:db8:6464::8c52:7903**
- **fd42:0:0:0:0:feed::/96** + **190.239.176.209** → **fd42::feed:beef:b0d1**
- **fd42::/96** + **115.49.115.49** → **fd42::7331:7331**

Usage: transition mechanisms like (6rd, nat64, and others).
The notation *prefix*:*ipv4 address* is allowed: One can write **fd42::feed:115.49.115.49** instead of **fd42::feed:7331:7331**.

IPv4 in an IPv6: the NAT64 example (read "NAT six-to-four") – opt

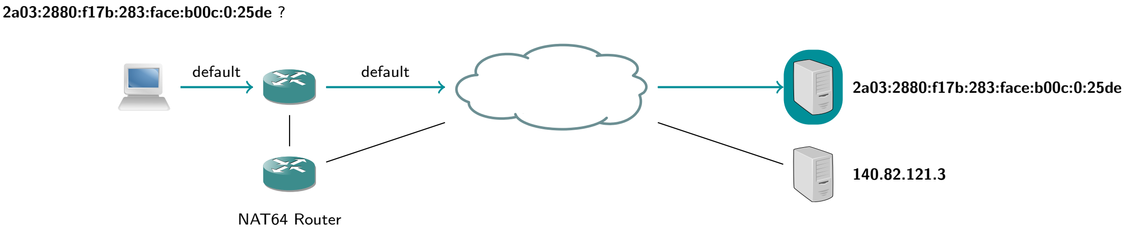


Figure: NAT64 Example: how to reach an IPv4-only server from an IPv6-only connection?

IPv4 in an IPv6: the NAT64 example (read "NAT six-to-four") – opt

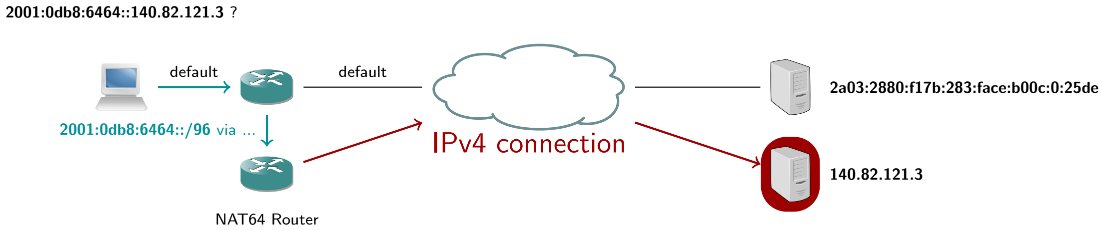


Figure: NAT64 Example: how to reach an IPv4-only server from an IPv6-only connection?

IPv4 in an IPv6: the NAT64 example (read "NAT six-to-four") – opt

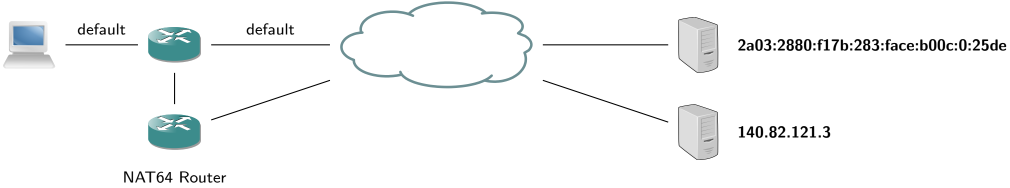


Figure: NAT64 Example: how to reach an IPv4-only server from an IPv6-only connection?

Hall of shame of popular IPv4-only services: `https://whynoipv6.com/`

Link local address

Every interface has at least one IPv6 address to talk on the local network: It can be computed from the MAC address. This is following the EUI64 computation:

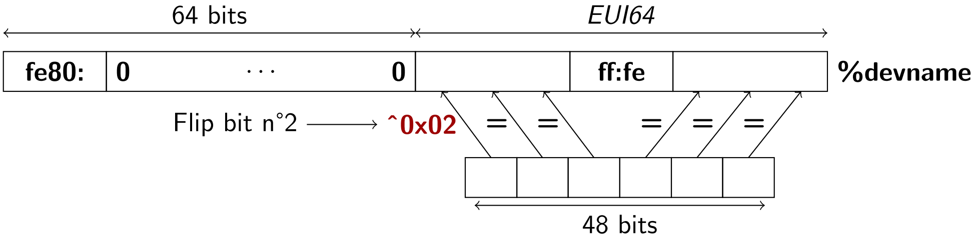


Figure: Formula to compute the EUI64

NB:

- Very convenient to connect to a freshly plugged in device.
- For privacy reasons, some prefer to generate a **random suffix**.

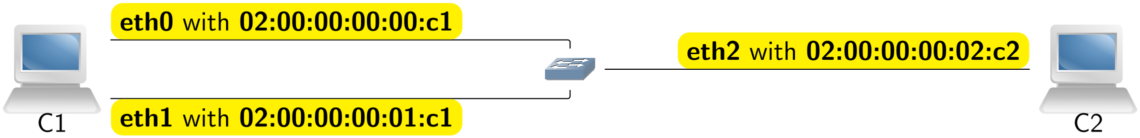
Exercise: Compute the link local addresses of the following network devices

```
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group default ql
    link/ether 08:92:04:34:1a:a0 brd ff:ff:ff:ff:ff:ff
3: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether fa:16:3e:4a:19:8e brd ff:ff:ff:ff:ff:ff
    altname enp0s3
4: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DORMANT group default qlen 10
    link/ether c4:75:ab:a5:39:17 brd ff:ff:ff:ff:ff:ff
5: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default qlen 1
    link/ether 42:54:00:47:85:27 brd ff:ff:ff:ff:ff:ff
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:00:04:db:b5:0e brd ff:ff:ff:ff:ff:ff
12: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/none
45: enx08920483f152: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group def
    link/ether 08:92:04:83:f1:52 brd ff:ff:ff:ff:ff:ff
```

The devicename is part of the link-local address

⚠ The %devicename is part of the address ⚠

Example with a machine with two network devices:



- **C1 sends packets to C2 via first device:** `ping fe80::ff:fe00:2c2%eth0`
- **C1 sends packets to C2 via second device:** `ping fe80::ff:fe00:2c2%eth1`
- **C2 sends packets to C1's first device:** `ping fe80::ff:fe00:c1%eth2`
- **C2 sends packets to C1's second device:** `ping fe80::ff:fe00:1c1%eth2`

Auto Configuration

How to automatically configure a computer freshly connected to a network?

- IPv4: DHCP protocol.
  (Client broadcasts a DISCOVER, a server OFFERs, Client REQUEST then Server ACK).
- IPv6: Router Advertisement:
  (The router broadcasts a RA).

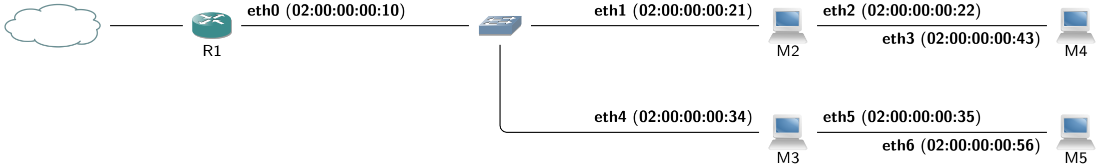
A rather typical setup: a network is provided by a /64 address. The gateway broadcasts a message **advertising for the network**. Every computer picks an address (suffix) in the range. This suffix can be:

- The **EUI64** ("**ff:fe**") suffix computed before.
- A random 64 bits suffix.

For **privacy reasons**, the outbound connections use the random address while the EUI64 address *may* be used for inbound connections only.
NB: RA is simpler than DHCP, but a DHCPv6 protocol exists for complex setups.

Exercise: IPv6 Typical Network

Assume your Internet provider assigned you the prefix **2001:db8:f00:f00::/60**. By default, the first **/64** network is used when its *box is plugged in, but additionnal subnets can be used by any computer. Choose a subnetting scheme and assign IP addresses to each interface.



Summary

Today, we have seen how to:

- Compute a Mask from a CIDR notation and vice-versa
- Split a network into subnets of the desired sizes
- Design static routing schemes and set them up on machines
- Same but in IPv6
- Compute an EUI64 based address from a MAC and vice versa