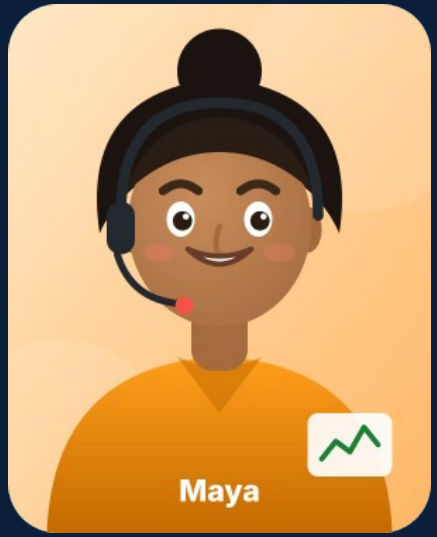
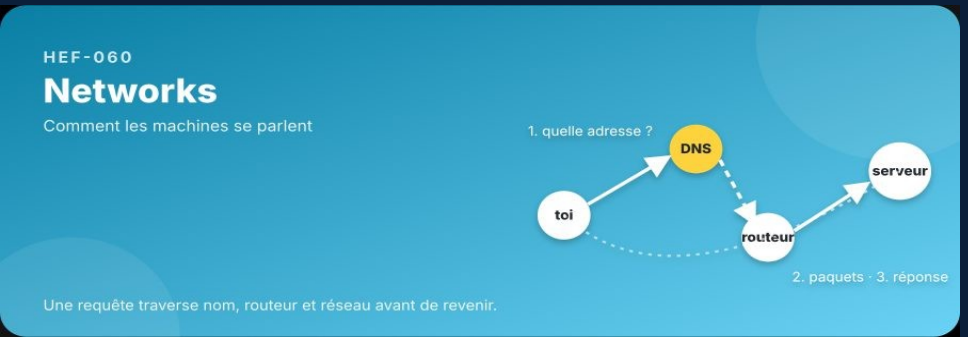
HEF-060 · MISSION 02 / 10

# Ethernet, adresses MAC et réseau local

*Sur un même réseau local, les machines se parlent grâce à leurs adresses MAC et à un switch qui apprend vite.*



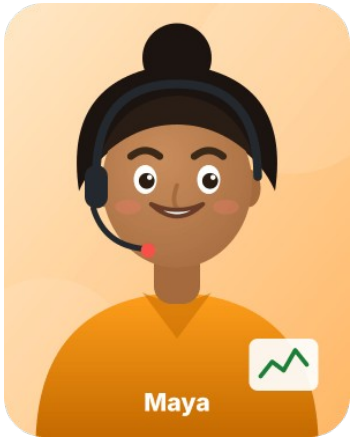

**Badge : Facteur du réseau local**

70 min · intermédiaire

# L'histoire

Au bureau, Kodi branche son ordinateur sur une prise murale. Il peut aussitôt joindre l'imprimante, le serveur de fichiers et le portable de Maya, sans rien configurer. Il se demande : « Comment mon ordinateur sait-il à qui envoyer chaque message ? »

*« Chaque carte réseau a une adresse unique, dit Sam, et un switch apprend où se trouve chacune. Ensuite il n'envoie le message qu'au bon endroit. »*




**Tu comprends comment fonctionne un réseau local : adresses MAC, trames, switch et ARP.**

Ta mission

# À la fin, je serai capable de…

1. décrire une adresse MAC et son format
2. expliquer le rôle d'une trame Ethernet
3. expliquer comment un switch apprend les adresses
4. expliquer le protocole ARP (adresse IP → adresse MAC)
5. distinguer hub, switch et domaine de diffusion (broadcast)

Défi d'échauffement

# Défi d'échauffement

Dans un grand open space, tu veux transmettre un papier à Léa. Option A : tu cries son prénom pour que tout le monde l'entende. Option B : tu sais où elle est assise et tu lui apportes le papier. Quand l'option A est-elle utile ? Comment un nouvel arrivant apprend-il où chacun est assis ?

Le concept

# L'adresse MAC

Chaque carte réseau possède une **adresse MAC** (*Media Access Control*) de **48 bits**, écrite en six octets hexadécimaux : `3c:22:fb:12:ab:9f`.

- Les **3 premiers octets** identifient le fabricant (OUI).
- Les 3 derniers sont attribués par le fabricant.
- `ff:ff:ff:ff:ff:ff` est l'adresse de **diffusion** (broadcast) : tout le monde la lit.

## La trame Ethernet

Sur un réseau local, les données voyagent dans une **trame** :

Le concept

# La trame Ethernet (suite)

| Champ | Taille | Rôle |
|---|---|---|
| Adresse MAC destination | 6 octets | à qui |
| Adresse MAC source | 6 octets | de qui |
| Type | 2 octets | quel protocole est dedans (IPv4, IPv6, ARP) |
| Données (*payload*) | 46 à 1 500 octets | le paquet IP |
| Contrôle (FCS) | 4 octets | détecter les erreurs |

La taille maximale des données, **1 500 octets**, s'appelle la **MTU** (*Maximum Transmission Unit*).

## Hub, switch

- Un **hub** répète chaque trame sur tous ses ports : tout le monde entend tout (collisions, pas de confidentialité).

Le concept

# Hub, switch (suite)

- Un **switch** apprend quelle MAC se trouve derrière quel port, puis **n'envoie la trame qu'au bon port**.

**Apprentissage du switch** :

1. Une trame arrive sur le port 3 avec la MAC source `A` : le switch note `A → port 3`.
2. Si la MAC destination est connue : il envoie sur ce seul port.
3. Si elle est inconnue (ou si c'est du broadcast) : il envoie sur **tous les ports sauf celui d'arrivée** (*flooding*).

## ARP : de l'adresse IP à l'adresse MAC

Ta machine connaît l'**IP** du destinataire (192.168.1.20), mais pour envoyer une trame elle a besoin de sa **MAC**. Elle demande en diffusion :

Le concept

# ARP : de l'adresse IP à l'adresse MAC (suite)

> *« Qui a l'adresse 192.168.1.20 ? Répondez à 192.168.1.10. » (ARP request)*

La machine concernée répond avec sa MAC (ARP reply). Le résultat est gardé dans une **table ARP** (cache) pendant un moment.

## Domaine de diffusion

L'ensemble des machines qui reçoivent un broadcast forme un **domaine de diffusion**. Un switch ne le coupe pas ; un **routeur** le coupe. Les **VLAN** permettent de découper un switch en plusieurs réseaux logiques.

## Manipuler une adresse MAC en Python

Le concept

# Manipuler une adresse MAC en Python (suite)

```
def mac_vers_octets(mac):
    return bytes(int(x, 16) for x in mac.split(":"))

def est_diffusion(mac):
    return mac.lower() == "ff:ff:ff:ff:ff:ff"

def est_multicast(mac):
    return bool(mac_vers_octets(mac)[0] & 1)       # bit de poids faible du premie
   r octet

print(mac_vers_octets("3c:22:fb:12:ab:9f").hex())    # 3c22fb12ab9f
print(est_diffusion("ff:ff:ff:ff:ff:ff"), est_multicast("01:00:5e:00:00:01"))   #
   True True
```

# Exemple pas à pas

**Problème** : simuler un switch à 4 ports qui apprend les adresses.

```
class Switch:
    def __init__(self, nb_ports):
        self.nb_ports = nb_ports
        self.table = {}                 # MAC -> port

    def recevoir(self, port, mac_src, mac_dst):
        self.table[mac_src] = port                          # apprentissage
        if mac_dst in self.table and mac_dst != "ff:ff:ff:ff:ff:ff":
            return [self.table[mac_dst]]                    # envoi ciblé
        return [p for p in range(self.nb_ports) if p != port]   # inondation

sw = Switch(4)
print(sw.recevoir(0, "A", "B"))    # B inconnue : [1, 2, 3]
print(sw.recevoir(1, "B", "A"))    # A connue : [0]
print(sw.recevoir(0, "A", "B"))    # B connue maintenant : [1]
print(sw.recevoir(2, "C", "ff:ff:ff:ff:ff:ff"))   # diffusion : [0, 1, 3]
```

# Exemple pas à pas (suite)

```
print(sw.table)
```

**Lecture** :

1. Au premier message A → B, le switch ne connaît pas B : il **inonde**. Il apprend que A est sur le port 0.
2. Quand B répond, le switch apprend que B est sur le port 1.
3. Ensuite, les échanges A ↔ B sont **ciblés**, les autres ports n'entendent plus rien.

**ARP en pratique** :

```
table_arp = {}

def resoudre(ip_cible, machines):
    if ip_cible in table_arp:
```

# Exemple pas à pas (suite)

```
        return table_arp[ip_cible], "cache"
    for ip, mac in machines.items():          # simulation de la diffusion ARP
        if ip == ip_cible:
            table_arp[ip] = mac
            return mac, "requête ARP"
    return None, "personne ne répond"

machines = {"192.168.1.20": "bb:bb:bb:bb:bb:02", "192.168.1.30": "cc:cc:cc:cc:cc:0
   3"}
print(resoudre("192.168.1.20", machines))
print(resoudre("192.168.1.20", machines))
print(resoudre("192.168.1.99", machines))
```

# À toi, avec aide

Un switch à 4 ports reçoit, dans cet ordre :

1. port 0 : A → B
2. port 1 : B → A
3. port 2 : C → A
4. port 0 : A → C
5. port 3 : D → ff:ff:ff:ff:ff:ff

Pour chaque trame, indique les ports sur lesquels le switch envoie, puis l'état de la table à la fin. Vérifie avec la classe `Switch`.

# Indices progressifs

## Indice 1

La première trame : B est inconnue → inondation (ports 1, 2, 3).

## Indice 2

Après la trame 2, le switch connaît A (port 0) et B (port 1).

## Indice 3

Une diffusion est toujours envoyée à tous les ports sauf celui d'arrivée.

# Réponse

```
class Switch:
    def __init__(self, n):
        self.n, self.table = n, {}
    def recevoir(self, port, src, dst):
        self.table[src] = port
        if dst in self.table and dst != "ff:ff:ff:ff:ff:ff":
            return [self.table[dst]]
        return [p for p in range(self.n) if p != port]

sw = Switch(4)
for port, s, d in ((0, "A", "B"), (1, "B", "A"), (2, "C", "A"), (0, "A", "C"), (3,
    "D", "ff:ff:ff:ff:ff:ff")):
    print(f"port {port} {s}→{d} : envoyé sur {sw.recevoir(port, s, d)}")
print(sw.table)
# [1,2,3] / [0] / [0] / [2] / [0,1,2] ; table : A:0, B:1, C:2, D:3
```

# À toi, en solo

Améliore le switch :

1. **Vieillissement** : une entrée de la table disparaît si elle n'a pas été vue depuis 300 secondes (utilise un temps simulé passé en paramètre).
2. **Déplacement** : si la MAC A apparaît sur un autre port, la table est mise à jour.
3. **Compteurs** : nombre de trames ciblées, inondées, diffusées.

Teste avec un scénario où une machine change de port et où une entrée expire.

À toi, en solo · réponse

# Pistes de correction

```
class Switch:
    def __init__(self, n, duree_vie=300):
        self.n, self.vie = n, duree_vie
        self.table = {}                # mac -> (port, derniere_vue)
        self.stats = {"cible": 0, "inonde": 0, "diffusion": 0}

    def recevoir(self, maintenant, port, src, dst):
        self.table[src] = (port, maintenant)                    # apprend ou met
   à jour
        self.table = {m: (p, t) for m, (p, t) in self.table.items() if maintenant
   - t <= self.vie}
        if dst == "ff:ff:ff:ff:ff:ff":
            self.stats["diffusion"] += 1
        elif dst in self.table:
            self.stats["cible"] += 1
            return [self.table[dst][0]]
        else:
            self.stats["inonde"] += 1
        return [p for p in range(self.n) if p != port]
```

À toi, en solo · réponse

# Pistes de correction (suite)

```
sw = Switch(4)
print(sw.recevoir(0, 0, "A", "B"))        # inonde
print(sw.recevoir(1, 1, "B", "A"))        # ciblé : [0]
print(sw.recevoir(2, 3, "A", "B"))        # A a déménagé sur le port 3, B connue :
    [1]
print(sw.recevoir(400, 2, "C", "B"))      # B a expiré (vue à t=1) : inonde
print(sw.stats)
```

# Pièges classiques

| Piège | Pourquoi c'est un problème | Comment l'éviter |
|---|---|---|
| **Croire que le switch lit les IP** | Un switch classique travaille en couche 2 | MAC pour le réseau local, IP pour le routage |
| **Confondre hub et switch** | Un hub répète tout | Le switch cible grâce à sa table |
| **Oublier que les MAC changent à chaque saut** | Elles ne traversent pas un routeur | MAC = local, IP = bout en bout |
| **ARP non sécurisé** | Une machine peut mentir (usurpation ARP) | Réseaux segmentés, protections côté switch |

# Mini-boss

**Mini réseau local.** Simule un réseau avec 1 switch (6 ports) et 4 machines. Chaque machine a une IP, une MAC, une table ARP. Écris `envoyer(machine_src, ip_dst, message)` qui :

1. consulte la table ARP ; sinon envoie une requête ARP en diffusion via le switch ;
2. reçoit la réponse ARP et met à jour la table ;
3. envoie la trame de données via le switch ;
4. journalise chaque étape (ARP, inondation, ciblage).

Montre le journal pour deux envois consécutifs A → B : le second doit être plus court.

Mini-boss · réponse

# Pistes de correction

Un bon travail construit : une classe `Machine` (ip, mac, arp, port), un `Switch` avec `recevoir` (comme ci-dessus), une fonction `envoyer` qui appelle d'abord `resoudre` (diffusion via le switch, réponse ciblée), puis envoie la trame. Le journal du premier envoi contient : requête ARP (diffusion, inondation), réponse ARP (ciblée), trame de données (ciblée) ; le second : seulement la trame de données (ciblée), car l'ARP est en cache et le switch connaît les ports.

Quiz éclair

# Quiz éclair

**1.** Une adresse MAC est composée de :

A. 32 bits
B. 128 bits
C. 48 bits

**2.** Un switch apprend où se trouvent les machines en observant :

A. l'adresse MAC source des trames
B. l'adresse IP de destination
C. les ports des programmes

**3.** ARP sert à :

A. chiffrer les trames

Quiz éclair

# Quiz éclair (suite)

B. trouver l'adresse MAC correspondant à une adresse IP
C. attribuer des adresses IP

**4.** 🔁 *Révision mission 01* : L'encapsulation consiste à :

A. chiffrer les données
B. compresser le message
C. ajouter un en-tête à chaque couche

**5.** 🔁 *Révision mission 01* : Un routeur prend ses décisions à partir de :

A. l'adresse IP de destination
B. l'adresse MAC de l'émetteur
C. le contenu de la page web

Quiz éclair · réponse

# Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

# Je réfléchis

1. Dans quelle situation de ta vie « cries »-tu (diffusion) plutôt que de t'adresser à une personne précise ?
2. Pourquoi est-il pratique que le switch apprenne seul plutôt que de tout configurer à la main ?

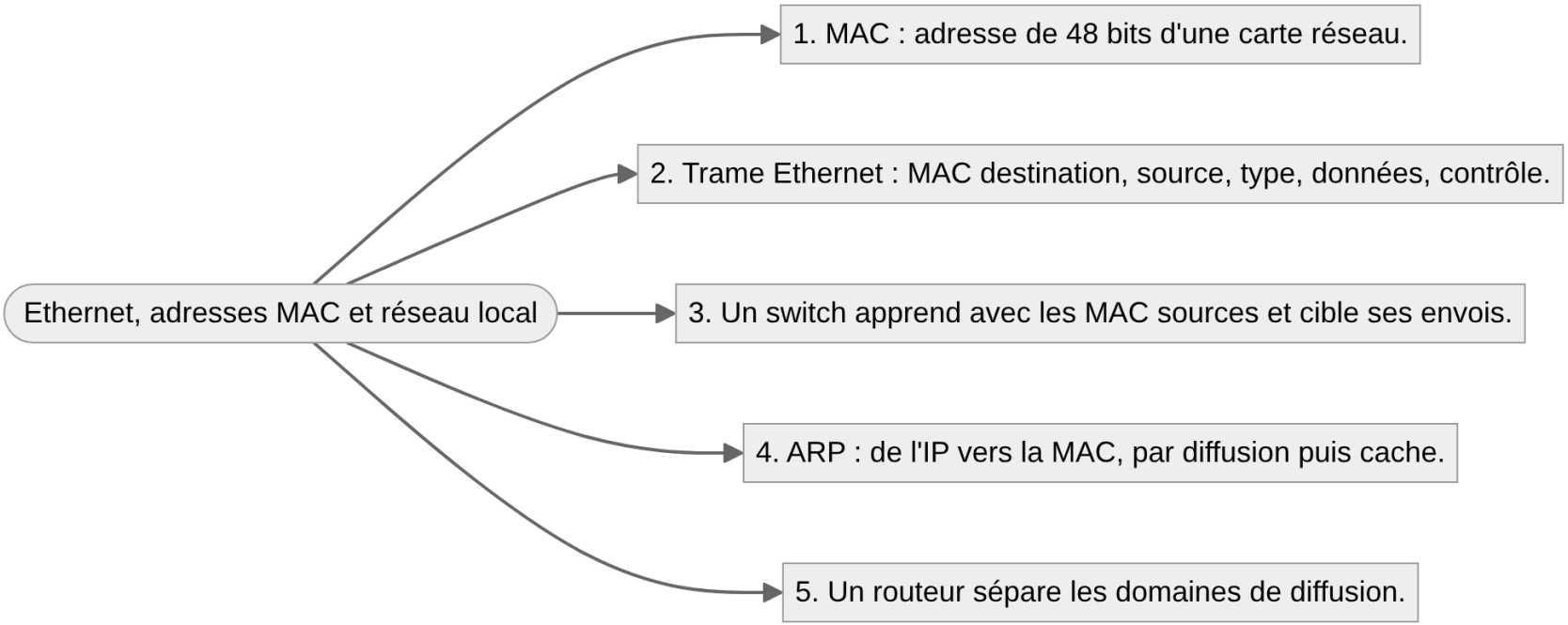
À retenir

# À retenir

1. MAC : adresse de 48 bits d'une carte réseau.
2. Trame Ethernet : MAC destination, source, type, données, contrôle.
3. Un switch apprend avec les MAC sources et cible ses envois.
4. ARP : de l'IP vers la MAC, par diffusion puis cache.
5. Un routeur sépare les domaines de diffusion.

À retenir

# Le schéma de la mission



MISSION ACCOMPLIE

# +100 XP

## Badge débloqué : Facteur du réseau local

Prochaine mission : Adresses IP et sous-réseaux

