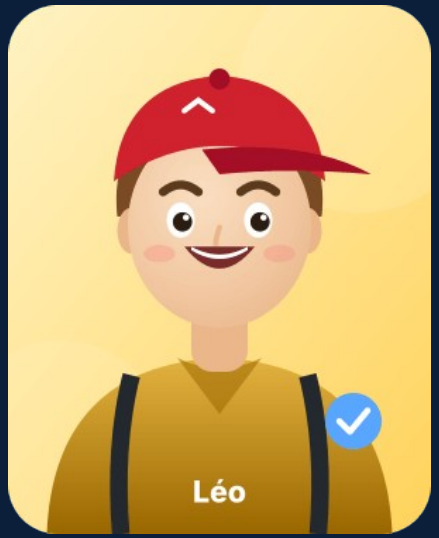
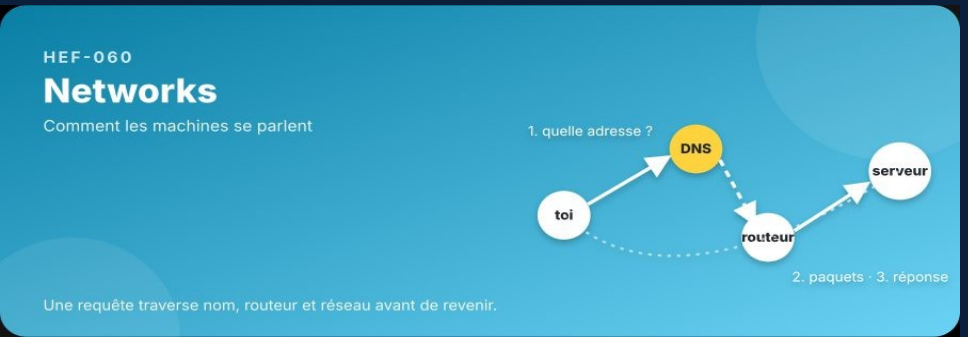
HEF-060 · MISSION 04 / 10

Le routage

Un routeur ne connaît pas internet : il connaît seulement le prochain pas à faire.



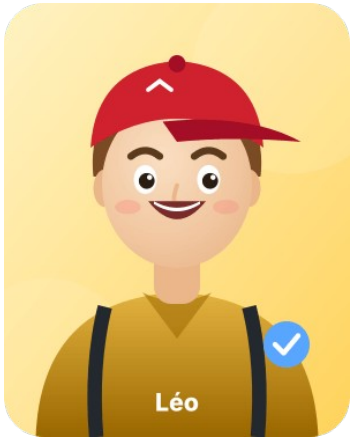


Badge : Aiguilleur

75 min · intermédiaire

L'histoire

Un client en Asie ne parvient plus à joindre HashBox. Un collègue exécute `traceroute` : la liste des machines traversées s'arrête au cinquième saut.



> *« Chaque routeur sur le chemin décide seul du prochain saut, dit Sam. Personne n'a la carte complète. Voyons comment ces décisions sont prises. »*

Tu comprends comment les paquets trouvent leur chemin à travers des centaines de réseaux.

Ta mission

À la fin, je serai capable de…

1. expliquer le rôle d'un routeur et d'une passerelle par défaut
2. lire une table de routage
3. appliquer la règle du plus long préfixe
4. expliquer le TTL et le fonctionnement de traceroute
5. distinguer routage statique et dynamique

Défi d'échauffement

À un carrefour sans carte, un panneau dit : « Paris → à gauche ; Lyon → tout droit ; autres destinations → à droite ». Tu veux aller à Marseille : quelle direction ? Que se passe-t-il s'il y a un panneau « Lyon et sa région → à droite » **et** « France → tout droit » ? Lequel suis-tu pour aller à Lyon ?

Le concept

Le routeur

Un **routeur** relie des réseaux. Pour chaque paquet qui arrive, il regarde l'**adresse IP de destination** et décide **par quelle interface et vers quel prochain saut** (*next hop*) l'envoyer. Il ne connaît pas le chemin complet.

La table de routage

Destination	Passerelle (prochain saut)	Interface
192.168.10.0/24	directement connecté	eth0
10.0.0.0/8	192.168.10.254	eth0
10.5.0.0/16	192.168.10.253	eth0
0.0.0.0/0	192.168.10.1	eth0

Le concept

La table de routage (suite)

La dernière ligne, `0.0.0.0/0`, est la **route par défaut** : « si rien d'autre ne correspond, envoie à cette passerelle ».

La règle du plus long préfixe

Plusieurs routes peuvent correspondre à une même destination. Le routeur choisit celle dont le **préfixe est le plus long** (la plus précise).

Pour la destination `10.5.7.9` : les routes `10.0.0.0/8`, `10.5.0.0/16` et `0.0.0.0/0` correspondent toutes. La plus spécifique est `10.5.0.0/16` : **c'est celle-ci qu'on utilise**.

Le TTL

Le concept

Le TTL (suite)

Chaque paquet IP porte un compteur **TTL** (*Time To Live*), souvent 64 au départ. Chaque routeur le **décrémente de 1**. À 0, le paquet est **détruit** et le routeur renvoie un message ICMP « temps dépassé ». Cela évite qu'un paquet tourne en boucle indéfiniment.

traceroute exploite cela : il envoie des paquets avec TTL = 1, 2, 3… Chaque routeur qui détruit un paquet répond, ce qui révèle le chemin.

```
1  192.168.10.1     1 ms     (ma passerelle)
2  10.255.0.1       5 ms     (réseau du fournisseur)
3  * * *                      (ce routeur ne répond pas)
4  203.0.113.1      28 ms
```

Routage statique et dynamique

Le concept

Routage statique et dynamique (suite)

	Statique	Dynamique
Configuration	à la main	protocoles qui échangent des routes
Exemples	route par défaut d'un poste	OSPF (réseau interne), BGP (entre opérateurs)
Adaptation aux pannes	non	oui

BGP est le protocole qui relie les milliers de réseaux autonomes qui forment internet.

Choisir une route en Python

```
import ipaddress

table = [
    ("192.168.10.0/24", "directement connecté"),
    ("10.0.0.0/8", "192.168.10.254"),
    ("10.5.0.0/16", "192.168.10.253"),
```

Le concept

Choisir une route en Python (suite)

```
    ("0.0.0.0/0", "192.168.10.1"),
]

def router(destination, table):
    dest = ipaddress.ip_address(destination)
    candidates = [(ipaddress.ip_network(r), p) for r, p in table if dest in ipaddr
   ess.ip_network(r)]
    reseau, passerelle = max(candidates, key=lambda c: c[0].prefixlen)
    return str(reseau), passerelle

print(router("10.5.7.9", table))      # ('10.5.0.0/16', '192.168.10.253')
print(router("10.9.1.1", table))      # ('10.0.0.0/8', '192.168.10.254')
print(router("8.8.8.8", table))       # ('0.0.0.0/0', '192.168.10.1')
```

Exemple pas à pas

Problème : simuler le voyage d'un paquet à travers trois routeurs, avec le TTL.

```
import ipaddress

class Routeur:
    def __init__(self, nom, table):
        self.nom = nom
        self.table = [(ipaddress.ip_network(r), saut) for r, saut in table]

    def prochain_saut(self, ip):
        dest = ipaddress.ip_address(ip)
        candidats = [(r, saut) for r, saut in self.table if dest in r]
        if not candidats:
            return None
        return max(candidats, key=lambda c: c[0].prefixlen)[1]

def envoyer(depart, reseau, destination, ttl=8):
    courant, chemin = depart, []
```

Exemple pas à pas (suite)

```
    while True:
        chemin.append(courant)
        saut = reseau[courant].prochain_saut(destination)
        if saut == "livré":
            return chemin, "livré"
        if saut is None:
            return chemin, f"destination inaccessible depuis {courant}"
        ttl -= 1
        if ttl == 0:
            return chemin, "TTL expiré"
        courant = saut

reseau = {
    "R1": Routeur("R1", [("0.0.0.0/0", "R2")]),
    "R2": Routeur("R2", [("203.0.113.0/24", "R3"), ("0.0.0.0/0", "R1")]),
    "R3": Routeur("R3", [("203.0.113.0/24", "livré")]),
}

print(envoyer("R1", reseau, "203.0.113.50"))     # (['R1', 'R2', 'R3'], 'livré')
```

Exemple pas à pas (suite)

```
print(envoyer("R1", reseau, "198.51.100.7"))     # boucle R1 <-> R2 : TTL expiré
```

Lecture :

- Le premier paquet traverse R1 → R2 → R3 et arrive.
- Le second est destiné à une adresse que **personne ne connaît** : R1 l'envoie à R2 (route par défaut) qui le renvoie à R1 (route par défaut)… une **boucle de routage**. Le **TTL** finit par l'arrêter : sans lui, le paquet tournerait indéfiniment.

À toi, avec aide

Voici la table d'un routeur :

Destination	Saut
172.16.0.0/12	A
172.16.8.0/21	B
172.16.8.128/25	C
0.0.0.0/0	D

Pour chaque destination, dis quelle route est choisie : `172.16.8.200`, `172.16.9.1`, `172.20.0.1`, `192.0.2.5`. Vérifie avec le programme `router`.

À toi, avec aide · réponse

Indices progressifs

Indice 1

`172.16.8.0/21` couvre 172.16.8.0 à 172.16.15.255.

Indice 2

`172.16.8.128/25` couvre 172.16.8.128 à 172.16.8.255.

Indice 3

Le plus long préfixe gagne.

À toi, avec aide · réponse

Réponse

```
import ipaddress

table = [("172.16.0.0/12", "A"), ("172.16.8.0/21", "B"), ("172.16.8.128/25", "C"),
    ("0.0.0.0/0", "D")]

def router(ip):
    d = ipaddress.ip_address(ip)
    c = [(ipaddress.ip_network(r), s) for r, s in table if d in ipaddress.ip_netwo
   rk(r)]
    return max(c, key=lambda x: x[0].prefixlen)[1]

for ip in ("172.16.8.200", "172.16.9.1", "172.20.0.1", "192.0.2.5"):
    print(ip, "→", router(ip))
# C, B, A, D
```

À toi, en solo

traceroute simulé. Utilise la classe `Routeur` de l'exemple. Écris `traceroute(depart, reseau, destination)` qui envoie des paquets avec TTL = 1, 2, 3… et affiche, pour chaque TTL, quel routeur « répond » (celui où le TTL tombe à 0) jusqu'à la livraison. Teste sur le réseau de l'exemple, puis ajoute un routeur muet qui ne répond pas (affiche `*`).

À toi, en solo · réponse

Pistes de correction

```
import ipaddress

class Routeur:
    def __init__(self, nom, table, muet=False):
        self.nom, self.muet = nom, muet
        self.table = [(ipaddress.ip_network(r), s) for r, s in table]
    def prochain_saut(self, ip):
        d = ipaddress.ip_address(ip)
        c = [(r, s) for r, s in self.table if d in r]
        return max(c, key=lambda x: x[0].prefixlen)[1] if c else None

def chemin_ttl(depart, reseau, destination, ttl):
    courant = depart
    for _ in range(ttl):
        saut = reseau[courant].prochain_saut(destination)
        if saut == "livré":
            return courant, True
        if saut is None:
            return courant, False
```

À toi, en solo · réponse

Pistes de correction (suite)

```
        courant = saut
    return courant, False

def traceroute(depart, reseau, destination, max_sauts=10):
    for ttl in range(1, max_sauts + 1):
        qui, livre = chemin_ttl(depart, reseau, destination, ttl)
        repond = not reseau[qui].muet
        print(f"{ttl:2d}  {qui if repond else '*'}")
        if livre:
            break

reseau = {
    "R1": Routeur("R1", [("0.0.0.0/0", "R2")]),
    "R2": Routeur("R2", [("203.0.113.0/24", "R3"), ("0.0.0.0/0", "R1")], muet=True
   ),
    "R3": Routeur("R3", [("203.0.113.0/24", "livré")]),
}
traceroute("R1", reseau, "203.0.113.50")
```

À toi, en solo · réponse

Pistes de correction (suite)

Un * ne signifie pas forcément une panne : certains routeurs ignorent volontairement ces sondes.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Oublier la route par défaut	Destination inaccessible pour tout ce qui est hors réseau	Toujours une route `0.0.0.0/0` sur un poste
Boucle de routage	Deux routeurs se renvoient le paquet	Le TTL protège, corriger les tables
Confondre route et chemin	Un routeur ne connaît que le prochain saut	Penser « saut par saut »
Mauvais masque dans une route	Mauvaise correspondance, mauvais plus long préfixe	Vérifier avec `ipaddress`

Mini-boss

Mini-boss

Routage dynamique minimal. Simule un réseau de 6 routeurs reliés par des liens (graphe). Chaque routeur doit calculer sa table de routage vers les autres routeurs en connaissant la topologie complète (principe d'OSPF).

1. Représente le réseau comme un graphe pondéré (coût de chaque lien).
2. Pour chaque routeur, calcule avec l'algorithme de Dijkstra le prochain saut vers chaque destination.
3. Simule l'envoi d'un paquet de bout en bout avec TTL.
4. Coupe un lien, recalcule les tables et montre que le trafic contourne la panne.

Mini-boss · réponse

Pistes de correction

```
import heapq

def dijkstra(graphe, source):
    dist, premier = {source: 0}, {}
    tas = [(0, source, None)]
    vus = set()
    while tas:
        d, n, saut = heapq.heappop(tas)
        if n in vus:
            continue
        vus.add(n)
        if saut is not None:
            premier[n] = saut
        for voisin, cout in graphe[n].items():
            if voisin not in vus and d + cout < dist.get(voisin, float("inf")):
                dist[voisin] = d + cout
                heapq.heappush(tas, (d + cout, voisin, saut or voisin))
    return premier
```

Mini-boss · réponse

Pistes de correction (suite)

```
def tables(graphe):
    return {r: dijkstra(graphe, r) for r in graphe}

def envoyer(tables_, depart, arrivee, ttl=10):
    chemin, courant = [depart], depart
    while courant != arrivee and ttl > 0:
        courant = tables_[courant].get(arrivee)
        if courant is None:
            return chemin, "inaccessible"
        chemin.append(courant)
        ttl -= 1
    return chemin, "livré" if courant == arrivee else "TTL expiré"

graphe = {
    "A": {"B": 1, "C": 4}, "B": {"A": 1, "C": 2, "D": 5}, "C": {"A": 4, "B": 2, "D
   ": 1, "E": 3},
    "D": {"B": 5, "C": 1, "F": 2}, "E": {"C": 3, "F": 1}, "F": {"D": 2, "E": 1},
}
print(envoyer(tables(graphe), "A", "F"))
```

Mini-boss · réponse

Pistes de correction (suite)

```
del graphe["C"]["D"]; del graphe["D"]["C"]          # panne du lien C-D
print(envoyer(tables(graphe), "A", "F"))
```

Après la panne, les tables sont recalculées et le trafic passe par un autre chemin : c'est l'intérêt du routage dynamique.

Quiz éclair

Quiz éclair

1. Un routeur choisit parmi plusieurs routes correspondantes celle qui a :

A. le plus long préfixe
B. le plus court préfixe
C. le plus petit numéro

2. La route par défaut est :

A. 127.0.0.1
B. 0.0.0.0/0
C. 255.255.255.255

3. Le TTL sert à :

A. chiffrer le paquet

Quiz éclair

Quiz éclair (suite)

B. compresser le paquet
C. empêcher un paquet de tourner indéfiniment

4. 🔁 *Révision mission 03* : Le masque 255.255.255.192 correspond à :

A. /26
B. /24
C. /28

5. 🔁 *Révision mission 03* : Un réseau /24 contient combien d'hôtes utilisables ?

A. 256
B. 254
C. 255

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quand tu voyages, suis-tu un plan complet ou des panneaux « prochain saut » ?
2. Pourquoi est-il robuste qu'aucun routeur n'ait la carte complète d'internet ?

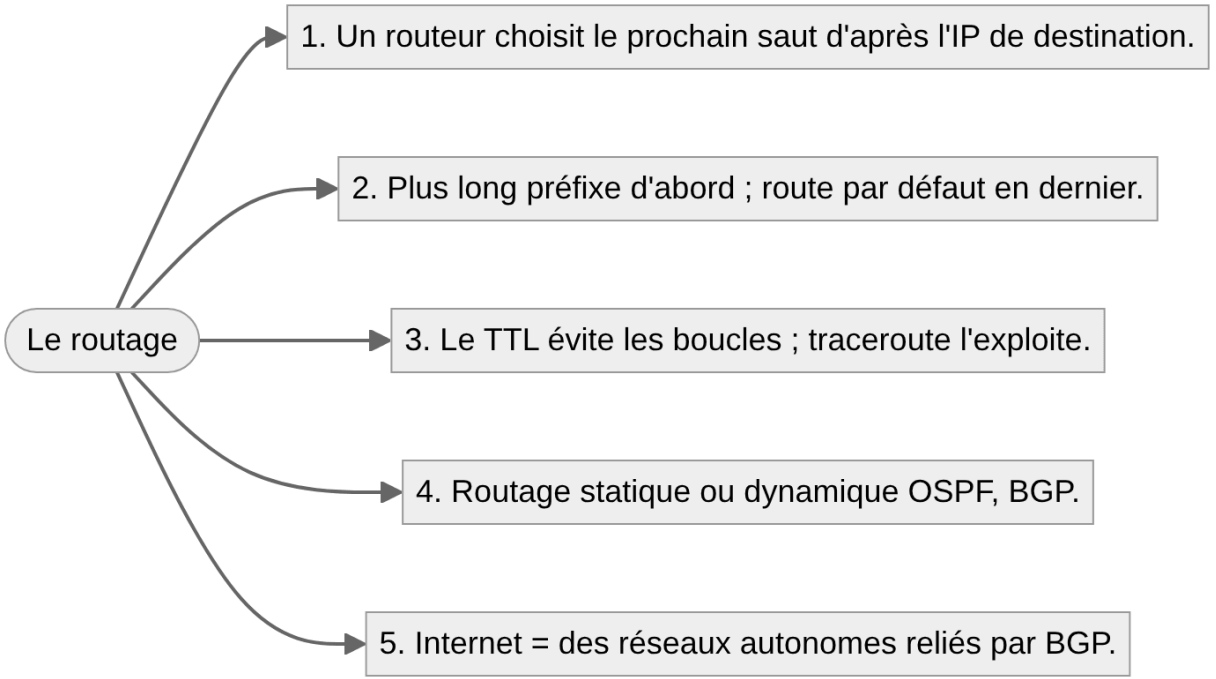
À retenir

À retenir

1. Un routeur choisit le prochain saut d'après l'IP de destination.
2. Plus long préfixe d'abord ; route par défaut en dernier.
3. Le TTL évite les boucles ; traceroute l'exploite.
4. Routage statique ou dynamique (OSPF, BGP).
5. Internet = des réseaux autonomes reliés par BGP.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Aiguilleur

Prochaine mission : TCP et UDP : deux façons de parler

