Systèmes et Réseaux (S6)
/ L3 Miage
Cours Réseaux / 2019-20

CM 4 :
Couche Réseau

D'après le cours de Bruno Martin et les slides du livre "Computer Networking: A Top Down Approach, 6th edition, Jim Kurose, Keith Ross, Addison-Wesley, March 2012"

Ramon APARICIO-PARDO
Ramon.Aparicio-Pardo@unice.fr

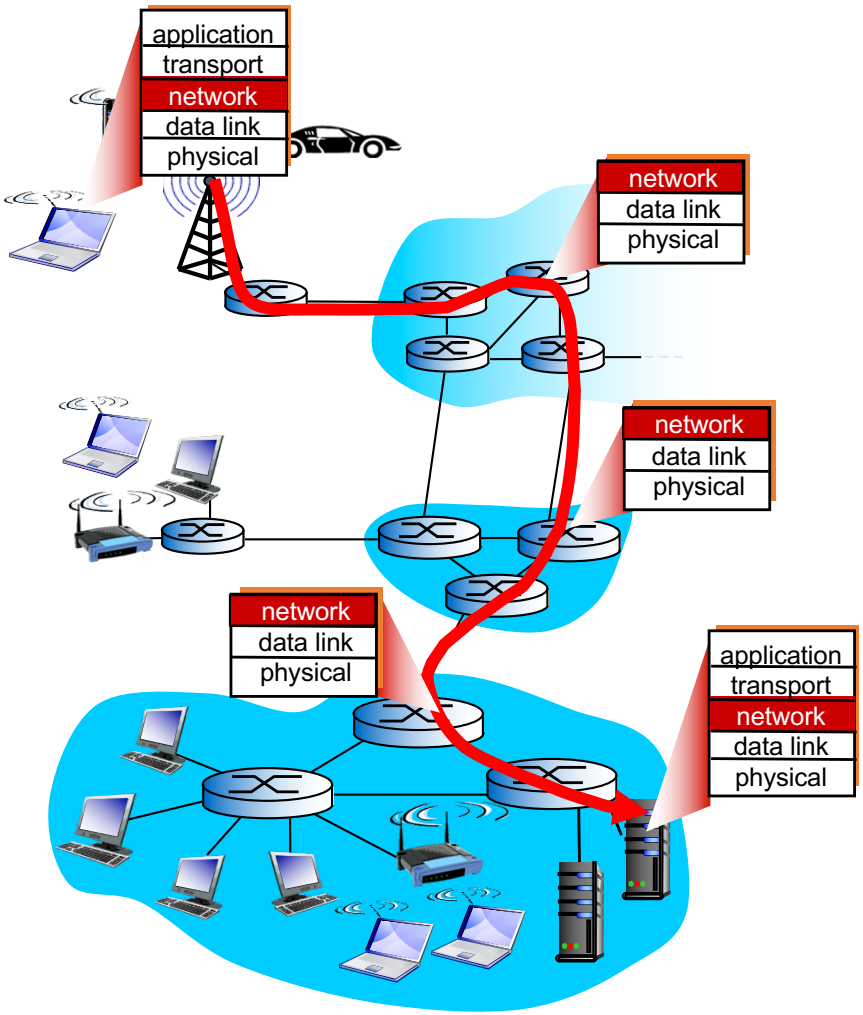
02/02/2020

PLAN CM 2

1. COUCHE RESEAU
2. IP : INTERNET PROTOCOL
3. ROUTAGE

Services / fonctions de la couche réseau

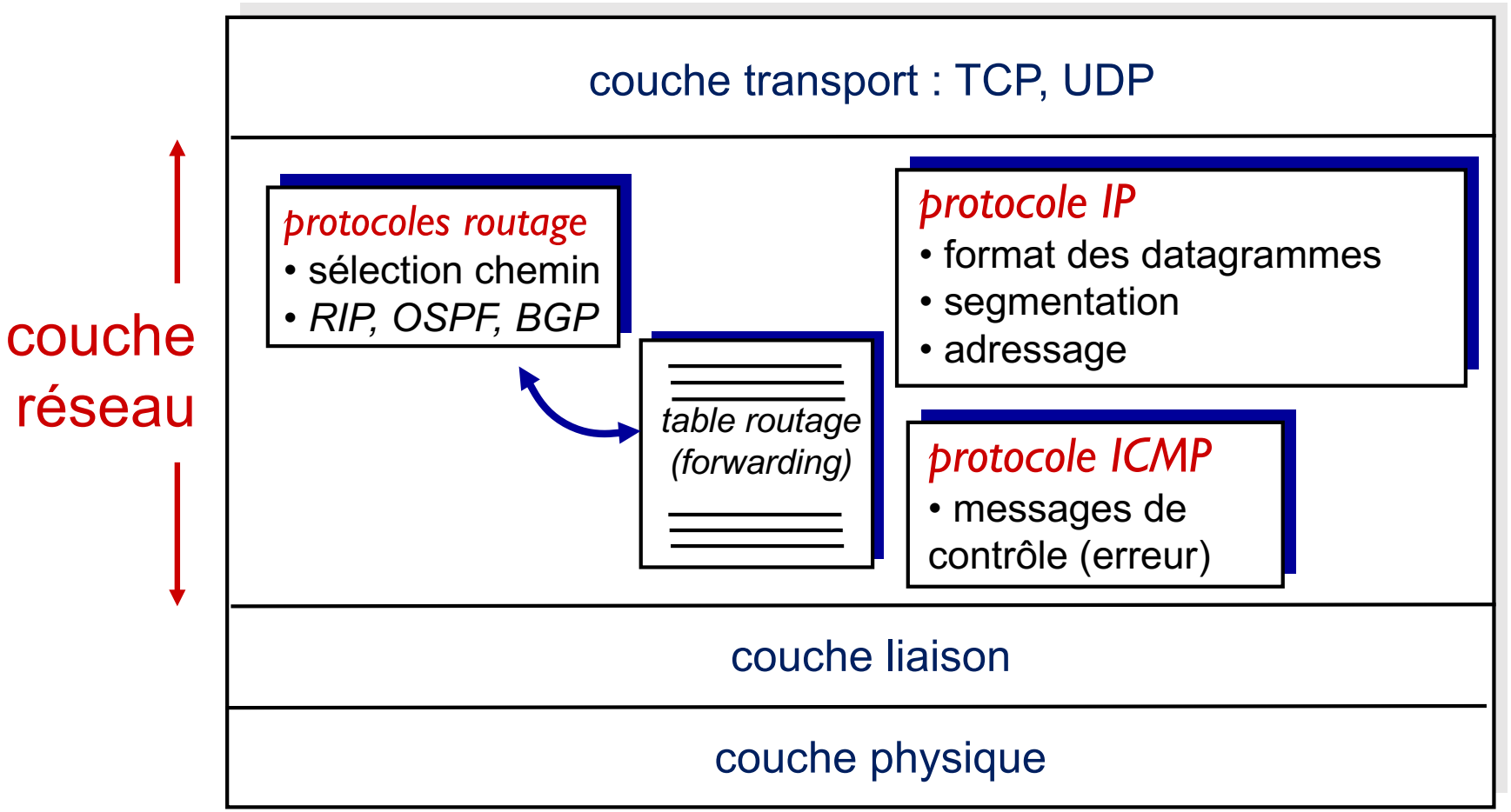
- Les protocoles de couche réseau sont dans toutes les machines (hôtes, routeurs)
- **Segmentation** : Typiquement dans les bouts
  - *Emetteur* encapsule les *segments* de la couche de transport en datagrammes
  - *Récepteur* réassemble les *datagrammes* en *segments*, qui sont délivrés à la couche de transport
- **Routage et *forwarding*** : Typiq. dans les routeurs
  - Ils examinent les champs d'en-tête des datagrammes IP qui les traversent.
  - Ils acheminent ces datagrammes IP en fonction de l'en-tête (adresses réseau)
- **Interconnexion des réseaux**
  - La couche détermine comment les données traversent les réseaux (sous-réseaux)
  - La couche gère les passages entre différents réseaux



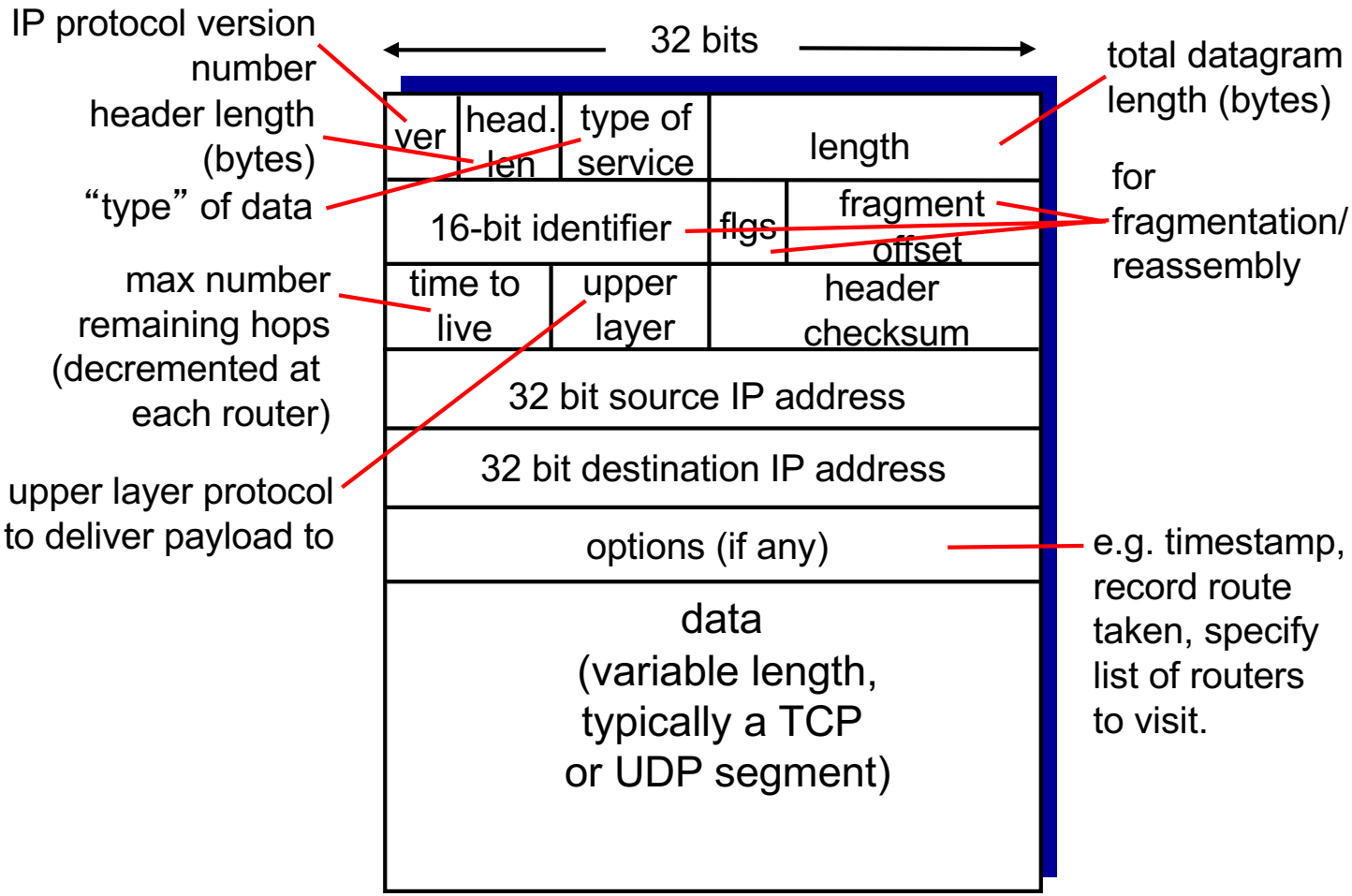
Services / fonctions de la couche réseau

- **IP (Internet Protocol)** : Protocol sur lequel pivote toute la couche réseau
- IP fournit un service
  - **non fiable** : fiabilité assurée par couches au dessus (notamment, TCP)
    - S'il y a un soucis, IP rejette les données et émet un datagramme ICMP à la source
  - **sans connexion** : datagrammes gérés indépendamment: peuvent suivre différentes routes
- Ce que **IP** *fait* :
  - définit le format des datagrammes
  - s'occupe de la segmentation et réassemblage des datagrammes
  - établit le plan d'adressage utilisé pour le *forwarding* (réexpédition d'un datagramme pour un interface)
- Ce que **IP** *ne fait pas* :
  - ne fournit pas des mécanismes de contrôle (détection d'erreurs) → ICMP
  - ne sélectionne pas le chemin (*routing*) des datagrammes → RIP, OSPF, BGP

Services / fonctions de la couche réseau

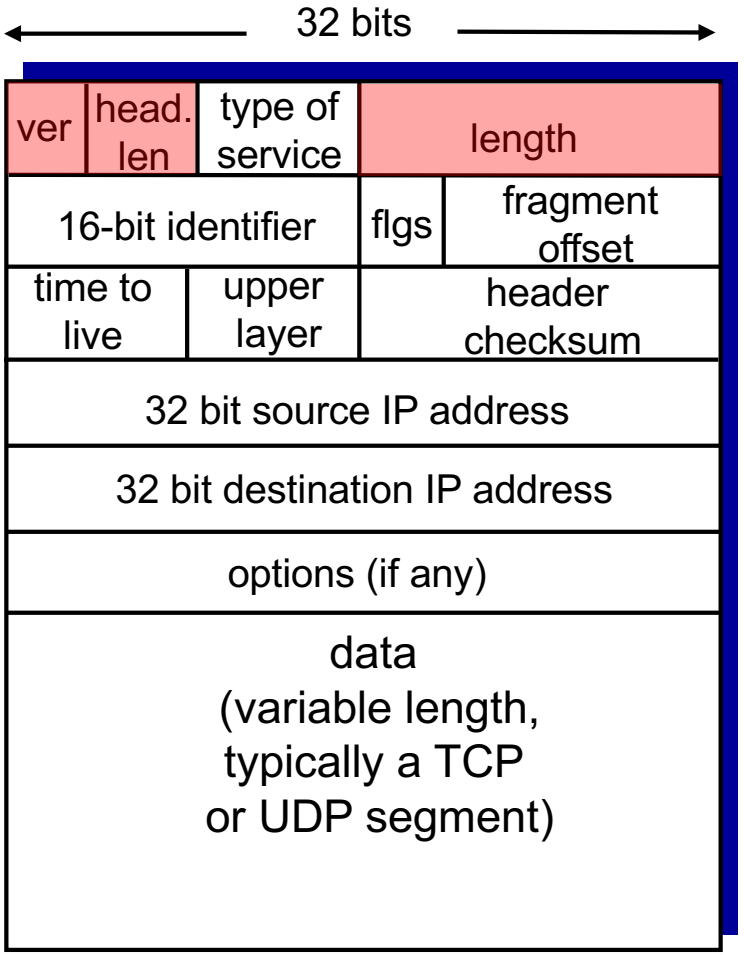


Format de datagramme IP



En-tête IP

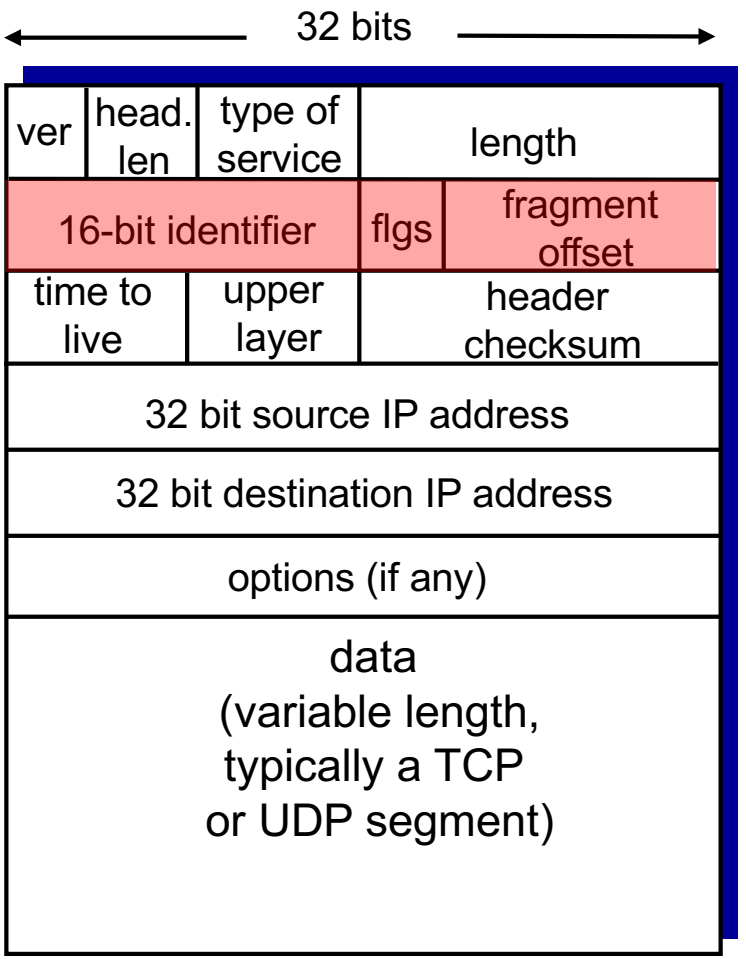
- **Version (4 bits)** : n° version protocole (IPv4=0100 ou IPv6=0110)
- **Longueur en-tête (4 bits) :** nombre mots de 32 bits de l'en-tête (pas d'options=5 et 15 au max)
- **Longueur totale (32 bits)** : taille en octets du datagramme (y compris en-tête) : 20 ≤ taille ≤ 65535



En-tête IP

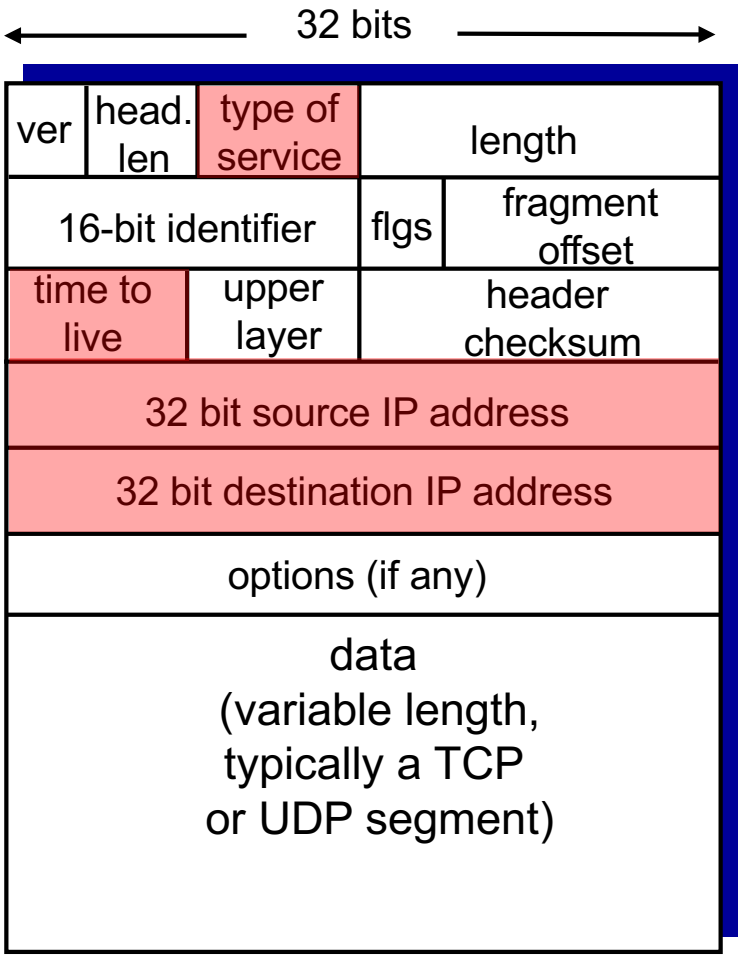
❖ **Informations liées à la fragmentation :**

- **Identifiant de datagramme (16 bits)**
- **Drapeaux d'état de fragmentation (3 bits) :**
  - b1 = inutilisé,
  - b2 = Do not Fragment (DF),
  - b3 =More Frag (MF)
- **Fragment offset (13 bits)** : position du fragment: tous les fragments d'un datagramme (sauf le dernier) ont une longueur multiple de 8 octets



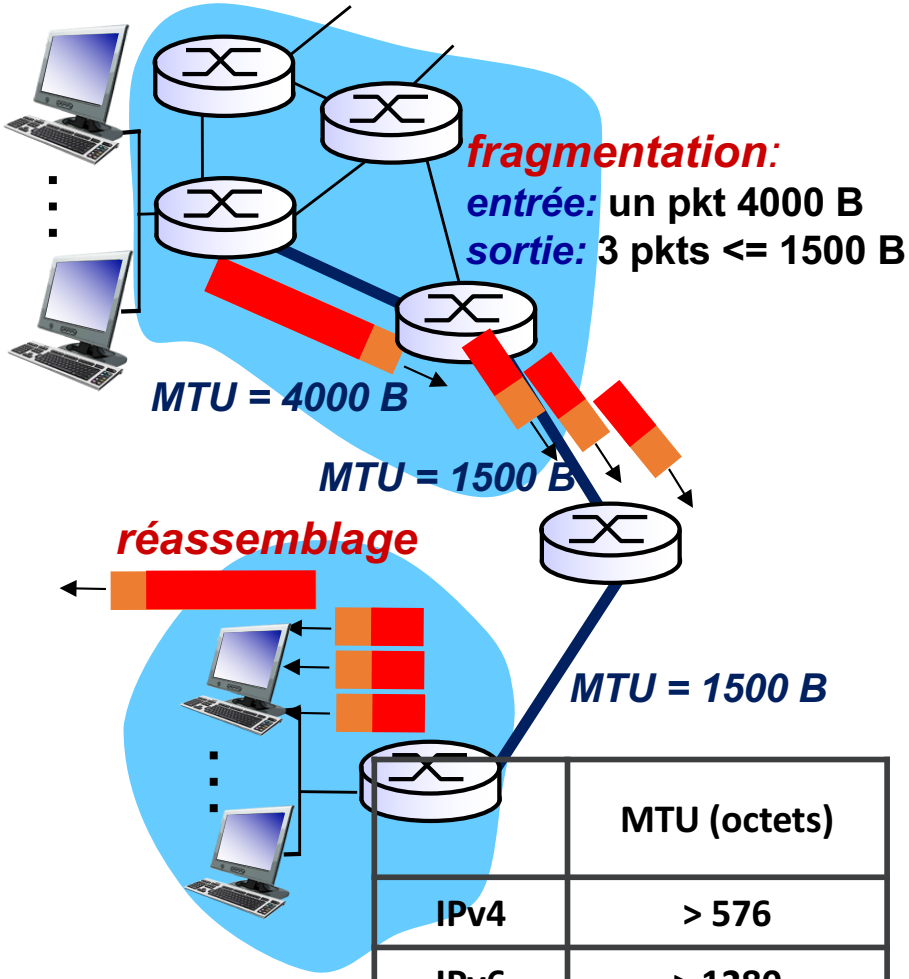
En-tête IP

- **Informations liées au routage** :
  - **Type of service, ToS (8 bits) :** utilisé pour donner priorité aux datagrammes des certains types
  - **Time to Live, TTL** : durée de vie du datagramme dans le réseau :
    - sur paquet IP: indique le nombre de routeurs traversés
    - l'application au dessus IP fixe une valeur décrémentée au passage de chaque routeur
    - quand TTL=0 paquet détruit et routeur renvoie *ICMP type =11* (*time exceeded*)
    - dans Unix, généralement TTL=64
  - **Adresses IP source (32 bits)**
  - **Adresses IP destination (32 bits)**



Fragmentation

- ***Maximum Transnission Unit (MTU)*** :
  - Chaque lien a un MTU: taille max du datagramme IP transportable par une unité de couche liaison en une fois.
  - Chaque fois qu'un routeur transfère un datagramme d'un lien vers un autre lien de + faible MTU, le datagramme est fragmenté.
  - Les routeurs ne réassemblent pas :
    - destinataire doit être capable de réassembler !
    - élimine les inconvénients précédents !
    - trafic non optimal (ajout en-tête à frag.) !
    - datagrammes peuvent prendre des chemins ≠!
  - Bits d'en-tête IP utilisées pour identifier des fragments en ordre.
  - En théorie, Ethernet permet jusqu'à 1500 octets, mais les hôtes IPv4 ne sont obligés qu'à supporter 576 octets [RFC791].



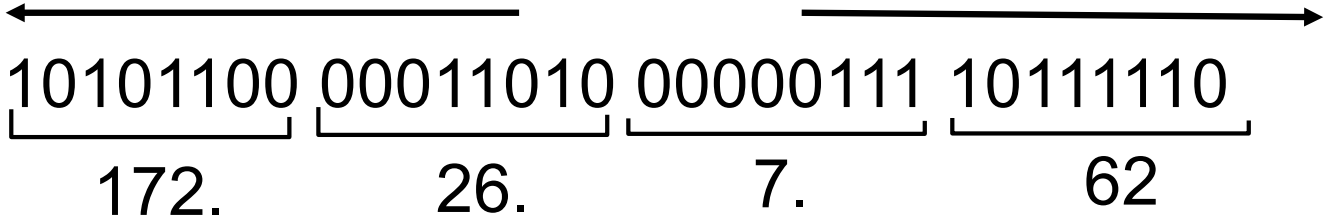
	MTU (octets)
IPv4	> 576
IPv6	> 1280
Ethernet	1500

Adressage IP

- **Une unique adresse IP associé à chaque interface**
  - Les adresses IP sont attribuées par l'ICANN (*Internet Corporation for Assigned Names and Numbers*) (http://www.icann.org/)

- **Interface** : Connexion entre l'hôte / routeur et la liaison physique:
  - Un router a généralement de multiples interfaces
  - Un hôte a typiquement une ou deux interfaces (p. ex., Ethernet filaire, wifi)

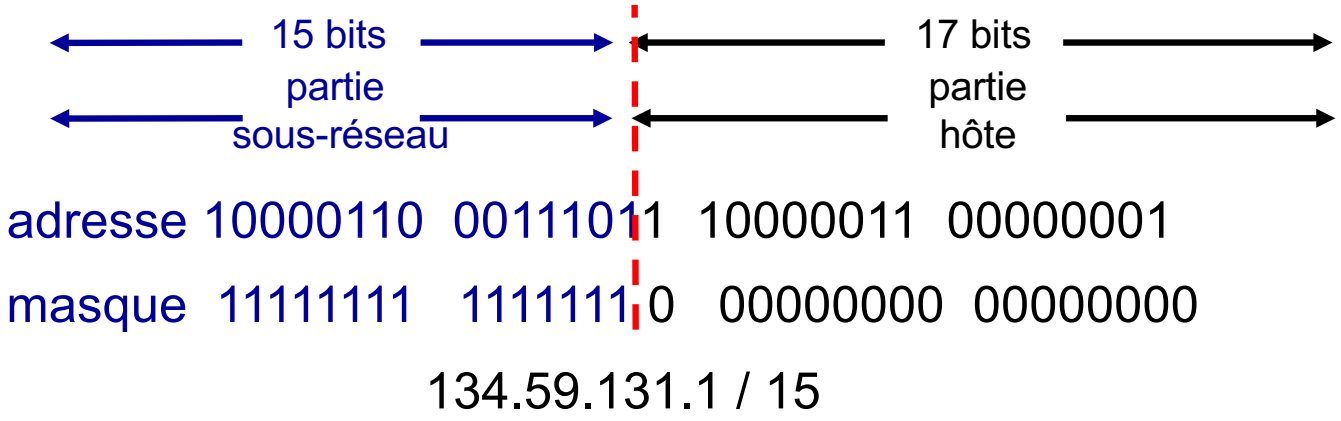
- Ecriture décimale pointée:



Adressage IP

❖ Classless Inter Domain Routing (CIDR)

- Chaque adresse IP est composé de deux parties:
  - Une partie de sous-réseau (*netid*) de longueur arbitraire: les *m* bits les plus à gauche
  - Une partie de hôte (*hostid*) : les *32-m* bits les plus à droite
- On peut récupérer le *netid* grâce au **masque** de sous-réseau: *m* bits à 1 et le reste à 0.
- Donc, le format d'adresse est : *a.b.c.d / m*, où *m* est le nombre de bits du *netid*.

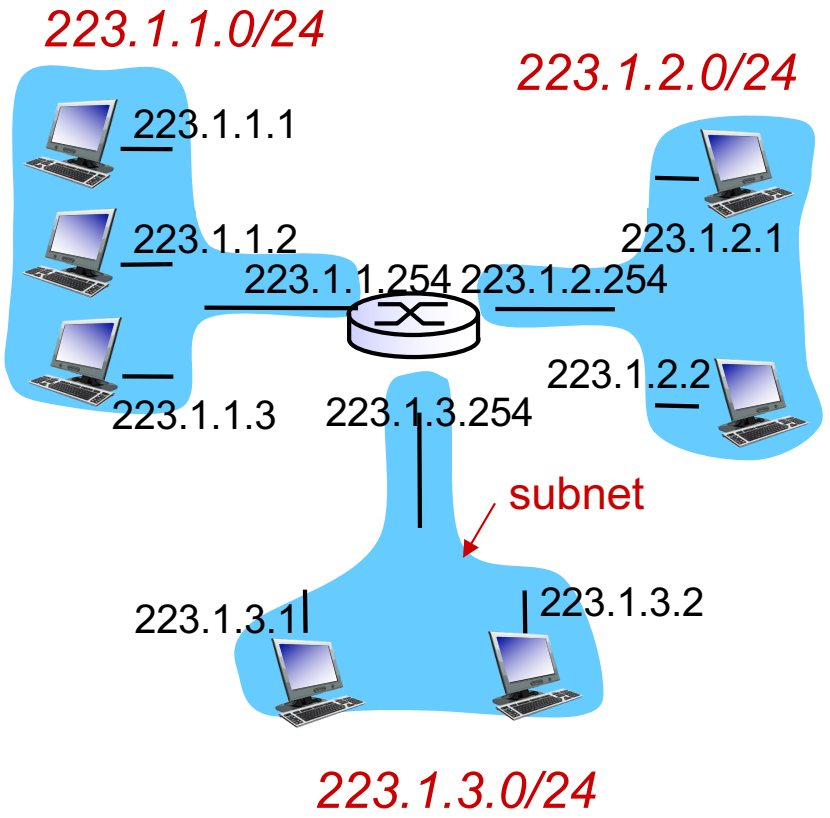


- Dans l'exemple, en décimal pointé:
  - 134.58.0.0/15 est l'adresse réseau
  - 0.1.144.1 est l'adresse machine.

Adressage IP

- **Sous-réseau** :
  - Les interfaces des hôtes avec le même *netid*.
  - Ils peuvent s'atteindre sans intervenir un routeur.
  - Donc, on découpe le réseau en îlots isolés.
  - Chaque îlot isolé est un sous-réseau.

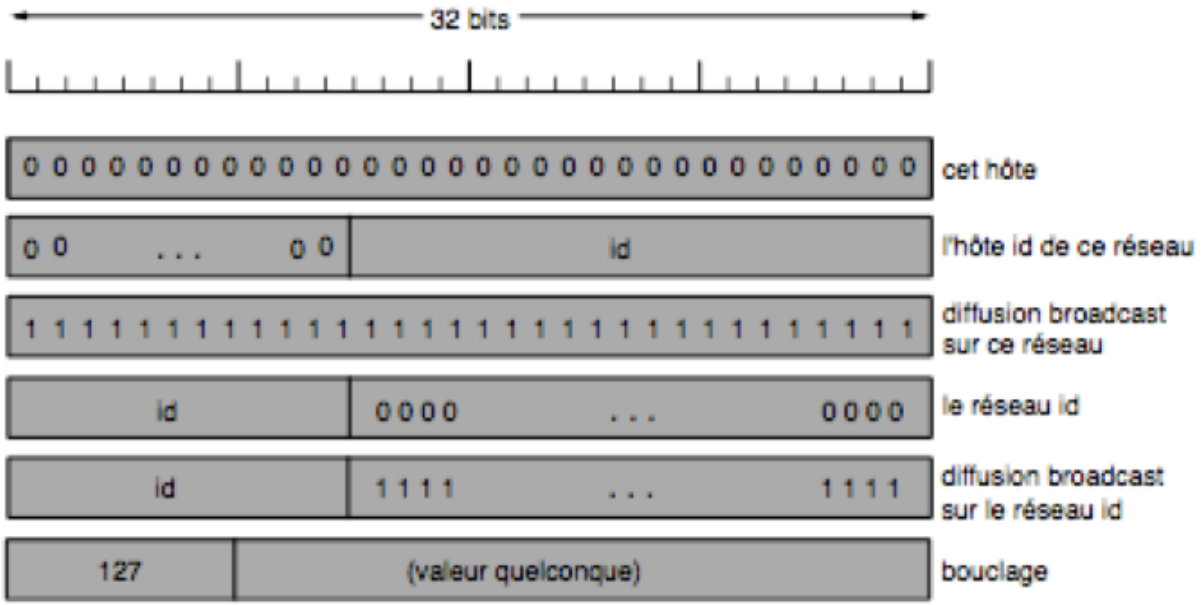
- Combien d'adresses disponibles pour chaque sous-réseau ?
  - $2^{32-m}-2$ adresses
  - L'adresse avec tout le *hostid* à 0, c'est l'adresse réseau
  - L'adresse avec tout le *hostid* à 1, c'est l'adresse de diffusion locale



masque: /24

Adressage IP

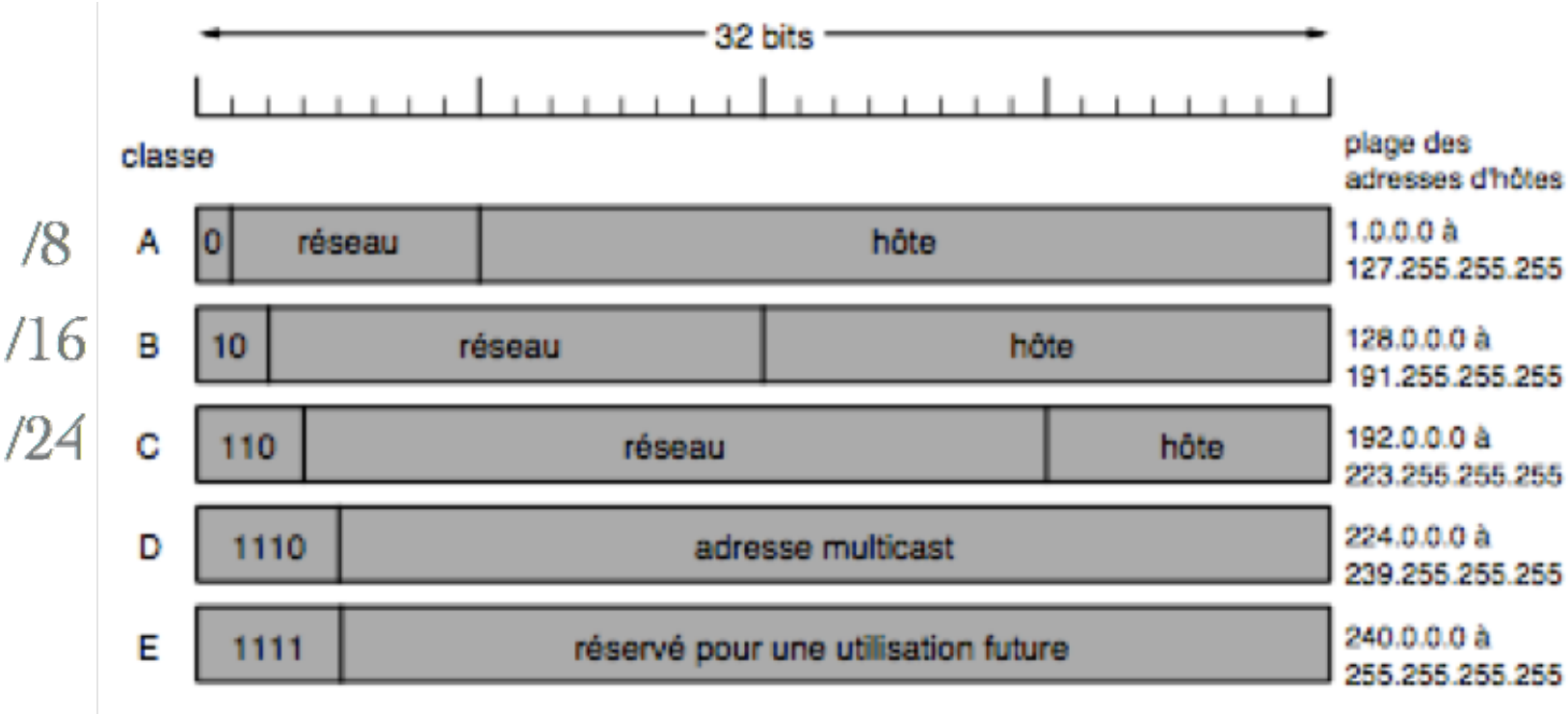
- **Adresses particulières**



Adressage IP

❖ Classes d'adresses

- Originalement, la longueur de *netid* était figée selon l'appartenance à une classe (5 classes)



Adressage IP

❖ Adresses IPv4 spéciales (RFC 1918)

- 127.x.x.x : *Loopback* (qui représente la machine locale)
- Non routables:
  - 10.0.0.0 - 10.255.255.255 (10.0.0.0/8, adresses privées de classe A)
  - 172.16.0.0 - 172.31.255.255 (172.16.0.0/12, adresses privées de classe B)
  - 192.168.0.0 -192.168.255.255 (192.168.0.0/16, adresses privées de classe C)
  - 224.0.0.0 - 239.255.255.255 (adresses de multicast, classe D)
- Pourquoi ?
  - Déploiement réseau TCP/IP privé sans risque de conflit d'adresses avec le reste du monde (passage a l'échelle)
  - Protection des attaques de l'Internet
  - Contrôle de l'accès vers l'Internet : les machines avec adresses privées ne peuvent communiquer directement avec le monde extérieur: nécessité d'une passerelle (routeur)
  - Contrôle de son propre plan d'adressage sans dépendance extérieure

Adressage IP

- **IP forwarding** :
  - Une seule table de routage (IP des, masque, iface sortie) :
    - paquet reçu: IP dest. récupérée converti binaire
    - parcourir les entrées avec masque jusqu'à correspondance
    - si plusieurs candidats, prendre celui avec masque le plus long (p.ex. entre /20 et /24)

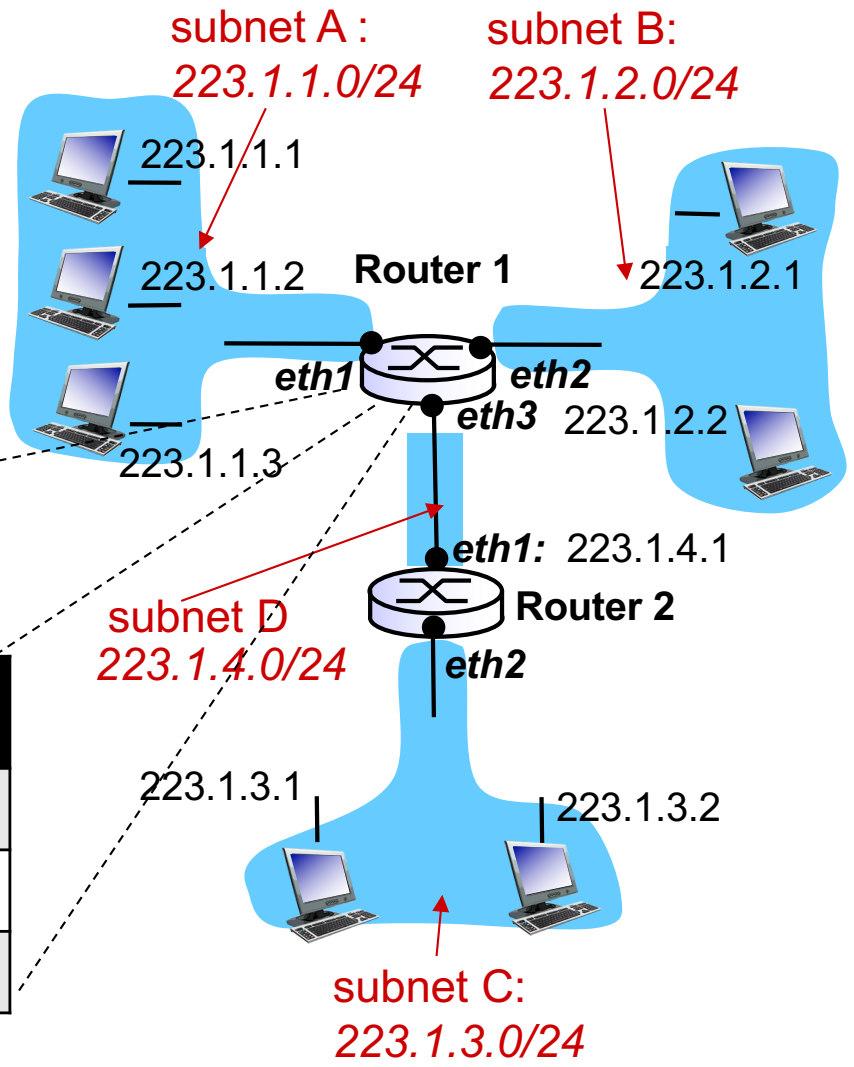


Tableau Router 1

Adresse Destination	Masque	Interface Sortie	Passerelle (*Next router*)
223.1.1.0	**/24**	**eth1**	**0.0.0.0**
223.1.2.0	**/24**	**eth2**	**0.0.0.0**
223.1.3.0	**/24**	**eth3**	**223.1.4.1**

Adressage IP

- Un autre exemple:
  - IP destination reçue:
    - 194.24.17.4
    - 11000010 00011000 00010001 00000100
  - *correspondant au réseau C*

Adresse Destination	Masque	Interface sortie
194.24.0.0	/20	eth1
194.24.8.0	/21	eth2
194.24.16.0	/22	eth3

Dst : 11000010 00011000 00010001 00000100

Opération And Logique

A : 11000010 00011000 00000000 00000000 → *No*

B : 11000010 00011000 00001000 00000000 → *No*

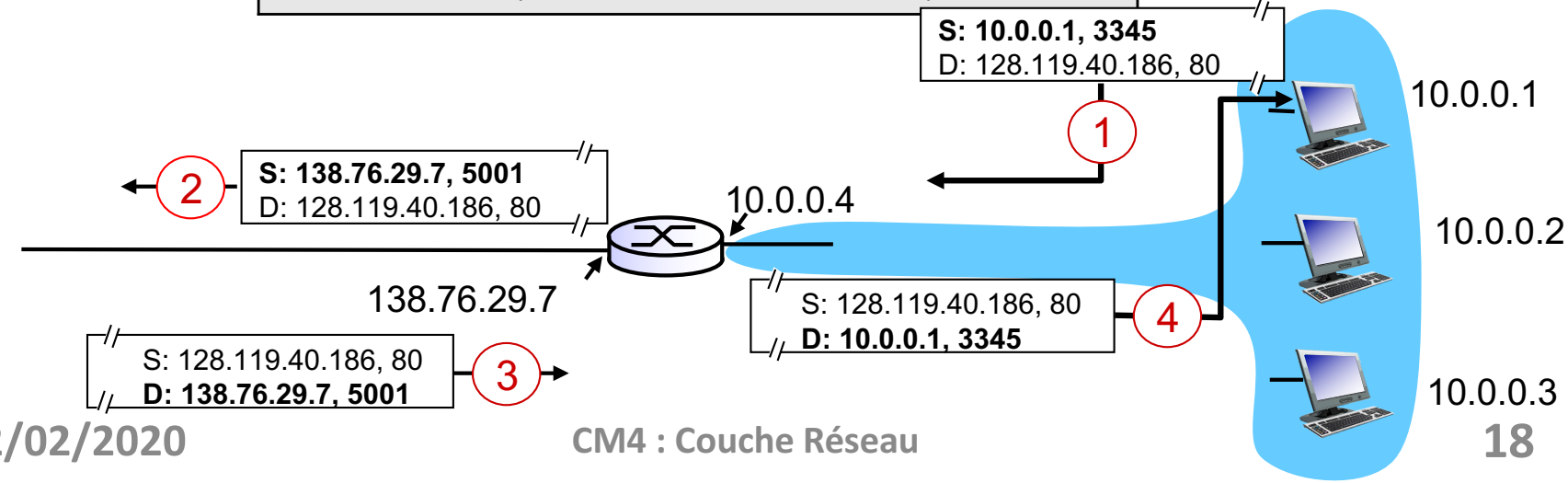
C : 11000010 00011000 00010000 00000000 → *Oui*

Adressage IP

❖ Networking Address Translation (NAT)

- une IP par organisation(p. ex. UNS)
- une seule IP par station dans l'organisation/LAN avec @IP privées (non routables) :
  - classe A: 10.0.0.0/8
  - classe B: 172.16.0.0/12
  - classe C: 192.168.0.0/16
- *NAT translation table* pour traduire des IP en utilisant des ports TCP/UDP

Internet (Adresse Publique, Port)	Réseau Local (Adresse Privée, Port)
138.76.29.7 , 5001	10.0.0.1 , 3345



ICMP

❖ **Internet Control Message Protocol**

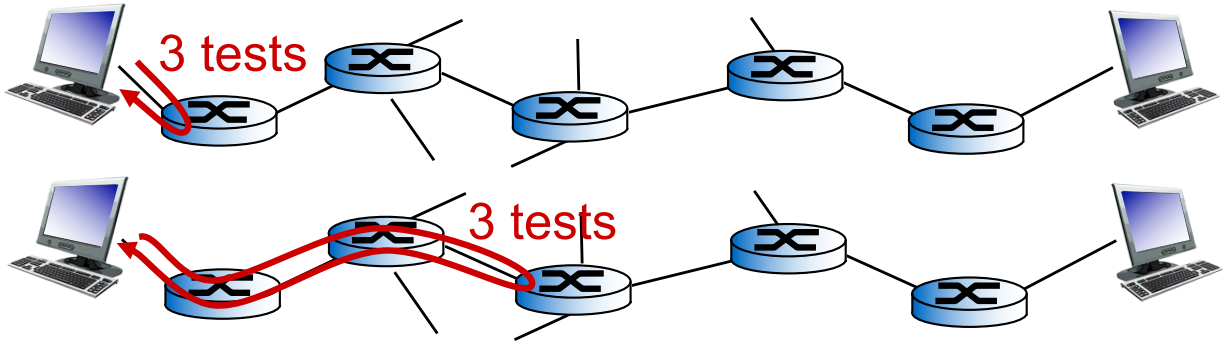
- gestion des informations d'erreurs des machines connectées
- peu de contrôle dans IP → ICMP signale les erreurs aux protocoles des couches voisines
- ICMP utilisé par routeurs pour signaler une erreur : un problème de dépôt (*delivery*)
- si erreur dans un datagramme ICMP, pas de message d'erreur (éviter effet boule de neige)
- ICMP encapsulé dans datagramme IP

en-tête (8 bits)	type (8 bits)	code (8 bits)	checksum (16 bits)	message (taille variable)

type	code	message	signification
8	0	demande ECHO	utilisé par `ping` pour test réseau envoi data destinataire et demande réponse
3	0	dest. inaccessible	réseau inaccessible
3	1	dest. inaccessible	machine inaccessible
3	2	dest. inaccessible	protocole inaccessible
3	3	dest. inaccessible	port inaccessible
11	0	TTL expiré	TTL expiré

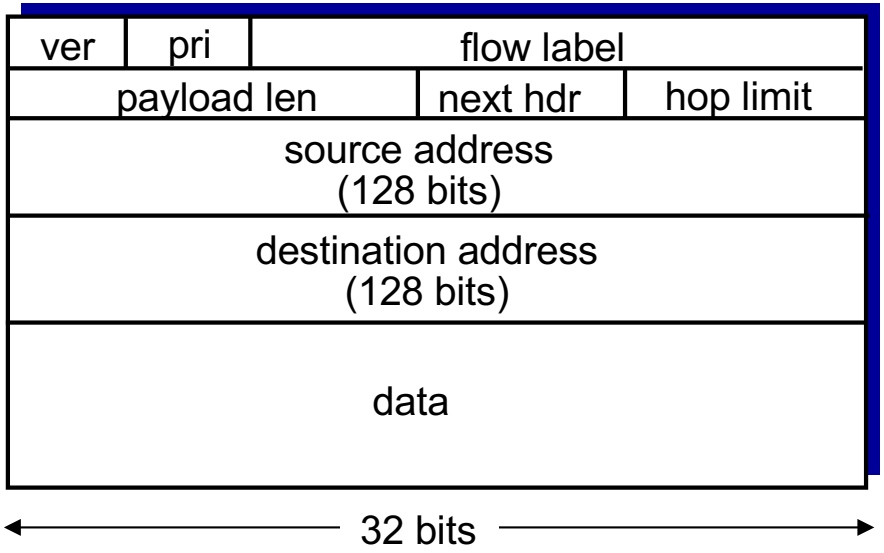
ICMP et `traceroute`

- Source envoie une série de segments UDP vers destinataire :
  - premier ensemble de segments à TTL = 1
  - deuxième ensemble a TTL = 2, etc.
  - envoyés vers un Numéro de port improbable
- Quand n-ème ensemble de datagrammes arrive au n-ème routeur :
  - Le routeur ignore les datagrammes
  - Et retourne des messages ICMP vers la source (type 11, code 0)
  - Les messages ICMP incluent le nom du routeur et l'adresse IP
- Lorsque les messages ICMP arrivent, la source enregistre les RTT
- Critères d'arrêt :
  - Le segment UDP arrive finalement à l'hôte de destination
  - Destination renvoie ICMP "port inaccessible" message (type 3, code 3)
  - La source s'arrête



IPv6

- **Motivation de base** :
  - Pour adresser + de machines suite croissance machines et applications (adresses sur 128 bits)
- **Format d'entêtes modifié pour** :
  - meilleure QoS: champ `priority`
  - traitement / forwarding simplifié : champ `header` qui identifie le protocole de couche supérieure pour les données
  - Options : autorisé, mais en dehors de l'en-tête, indiqué par le champ `header`
  - notion de «flux»: champ `flow label` qui identifie les datagrammes dans le même flux
  - champ `Checksum` : supprimé afin de réduire le temps de traitement à chaque saut
  - évolution et cohabitation avec IPv4

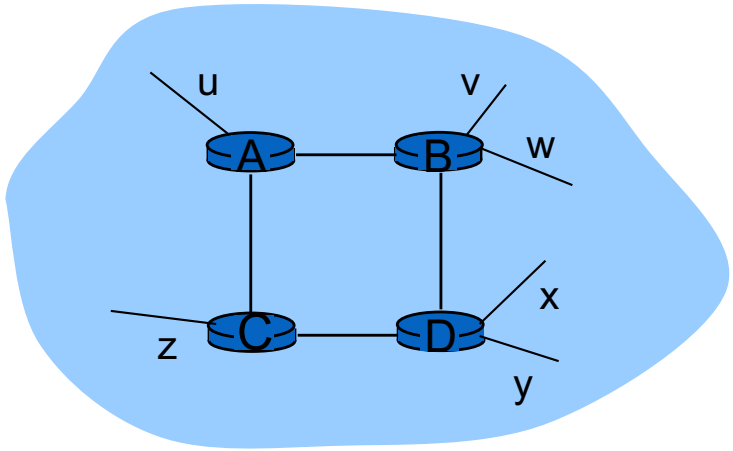


Routage

- Le protocole IP ne se occupe pas de trouver le routage, c.-à-d., trouver la suite de machines à traverser pour acheminer un datagramme IP vers son destination.
- Trouver le routage est nécessaire pour remplir les tables de *forwarding* qu'IP utilise.
- Deux types de protocoles de routage :
  - à vecteur de distances (*Distance Vector, DV*)
    - structure de données minimale: *destination / next hop / distance*
    - mise à jour simple: il suffit des échanges des vecteurs des distance entre les nœuds voisins
    - exemple: protocole RIP (*Routing Information Protocol*) qui utilise l'algorithme Bellman-Ford
  - d'état de lien (*Link State, LS*)
    - structure de données plus lourde: connaissance complète du réseau à chaque nœud (graphe + états des liens)
    - mise à jour plus complexe: besoin de maj du graph a chaque nœud par la diffusion paquets d'état de liens
    - exemple: protocole OSPF (*Open Shortest Path First*) qui utilise l'algorithme Dijkstra

Protocole RIP (*Routing Information Protocol*)

- Basé sur algorithme à vecteur de distances (*Bellman-Ford*)
  - nombre de hop comme poids des arcs (nb de sous-réseaux traversés dont la destination)
  - limité à 15 *hops*
- Algorithme à vecteur de distances
  - échanges de vecteur de distances avec voisins toutes les 30s (*advertisement*)
  - chaque notification (*advertisement*) liste au plus 25 destinations (sous-réseaux en sens IP)



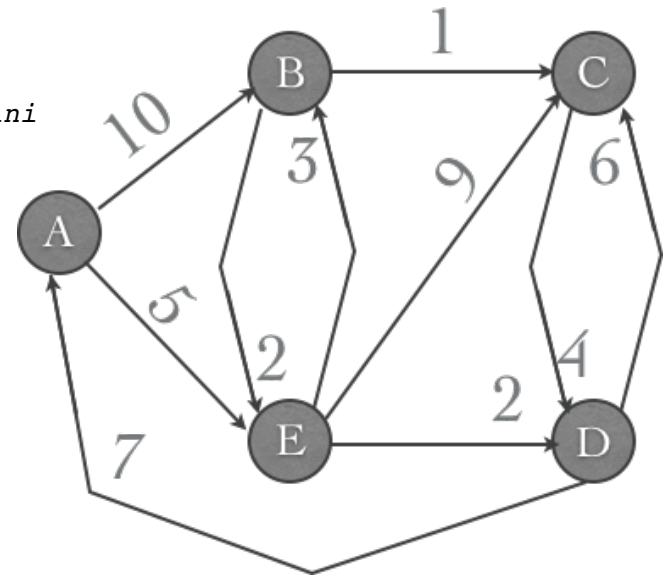
from router A to destination *subnets:*

subnet	hops
u	1
v	2
w	2
x	3
y	3
z	2

Bellman-Ford

- Cherche des plus courts chemins (PCC) d'origine A
- Donne arbre couvrant
- Exemple : Opération *backwards*

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

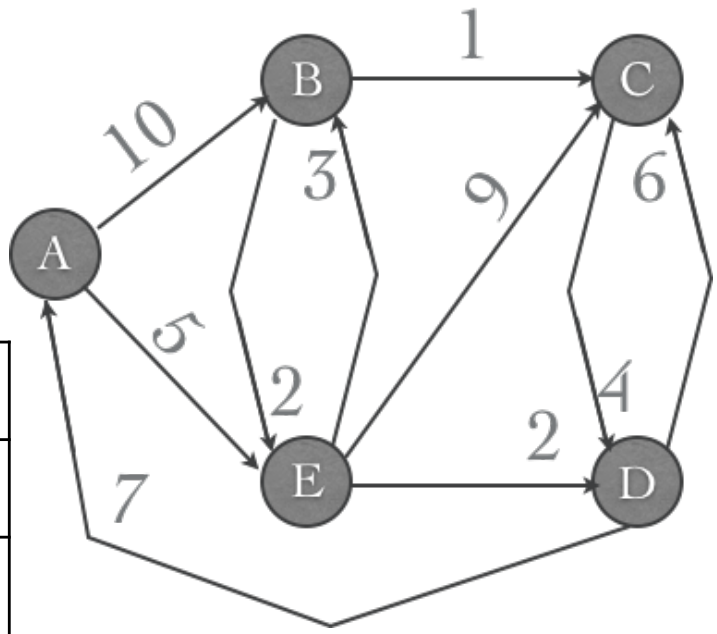


AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A			
2	0				



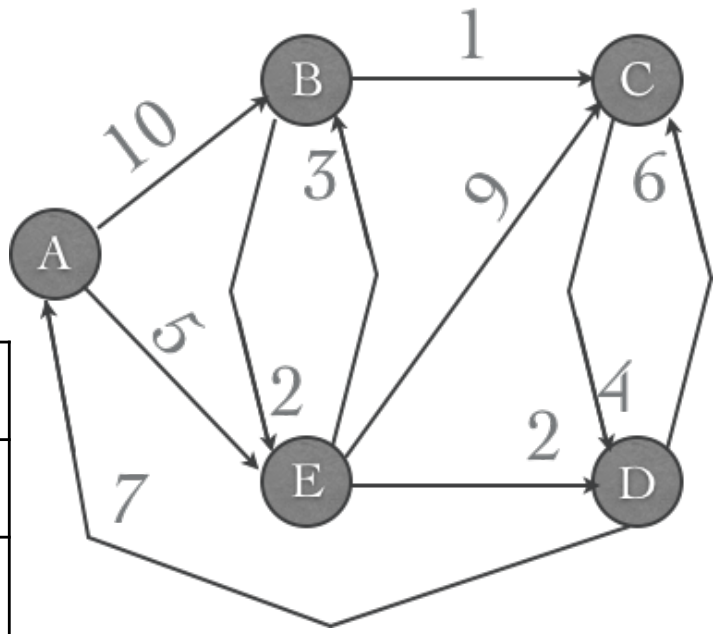
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à B : poids (B) c'est le min
poids(E) + poids(EB) = ∞ + 3 = ∞
poids(A) + poids(AB) = 0 + 10 =10

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A	11,B		
2	0				



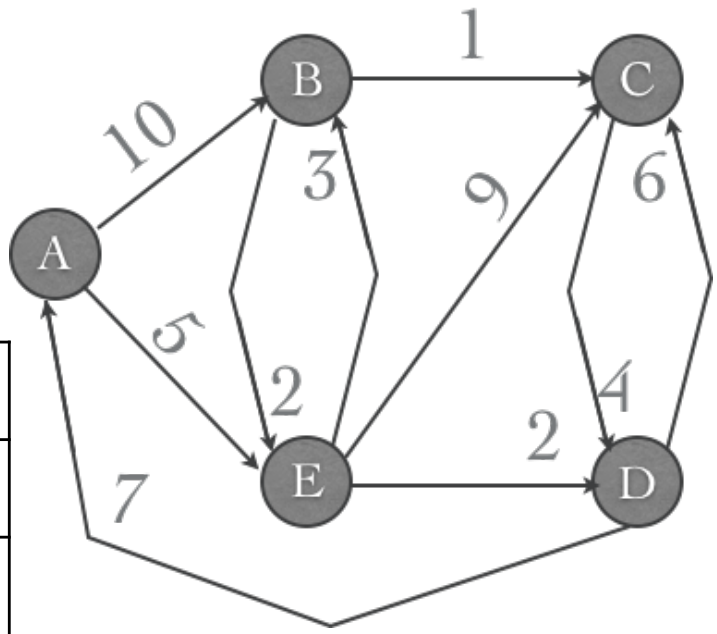
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à C : poids (C) c'est le min
poids(B) + poids(BC) = 10 + 1 = 11
poids(D) + poids(DC) = ∞ + 6 = ∞
poids(E) + poids(EC) = ∞ + 9 = ∞

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A	11,B	15,C	
2	0				



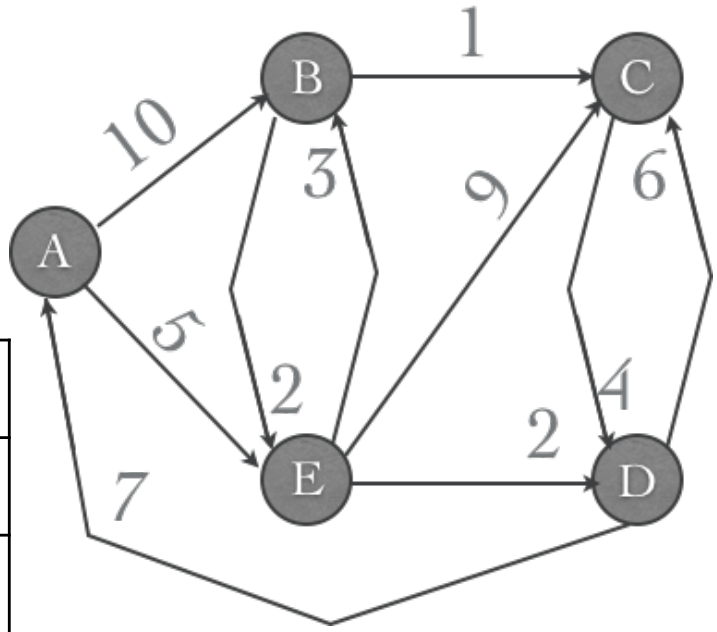
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à D : poids (D) c'est le min
poids(C) + poids(CD) = 11 + 4 = 15
poids(E) + poids(ED) = ∞ + 2 = ∞

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A	11,B	15,C	5,A
2	0				



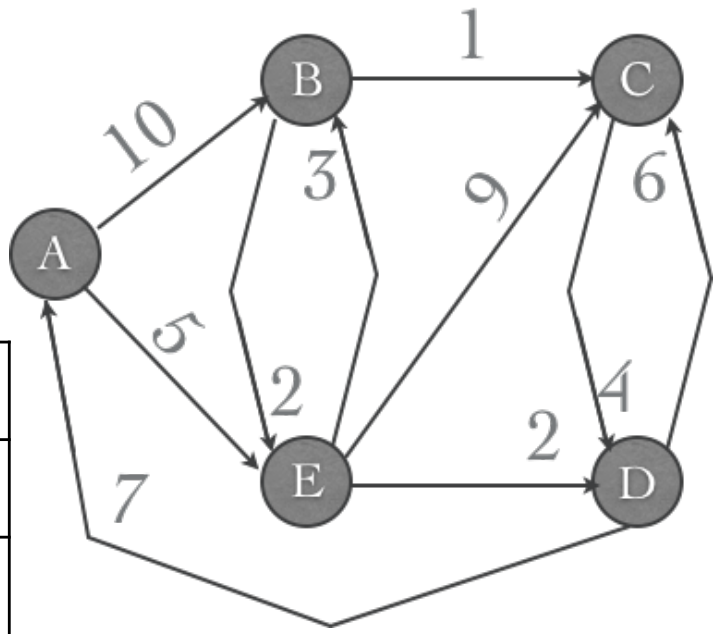
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à E : poids (E) c'est le min
poids(A) + poids(AE) = 0 + 5 = 5
poids(B) + poids(BE) = 10 + 2 = 12

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A	11,B	15,C	5,A
2	0	8,E			



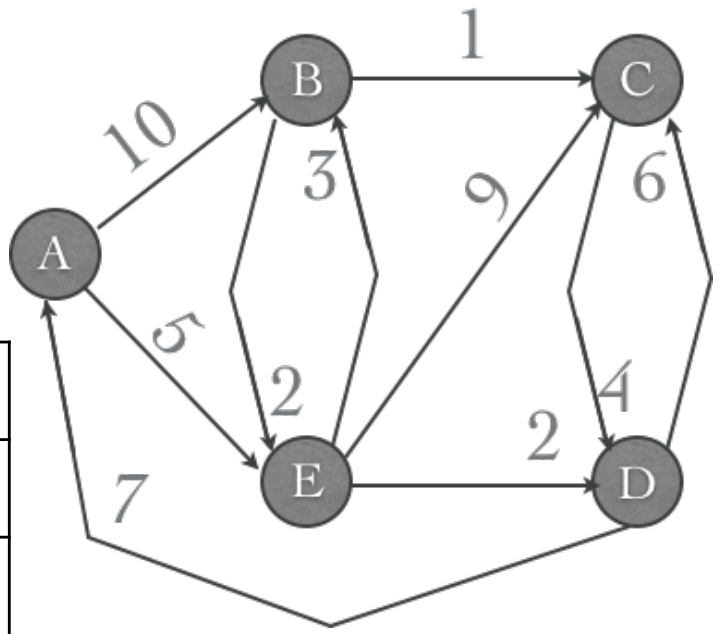
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à B : poids (B) c'est le min
poids(E) + poids(EB) = 5 + 3 = 8
poids(A) + poids(AB) = 0 + 10 = 10

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A	11,B	15,C	5,A
2	0	8,E	9,B		



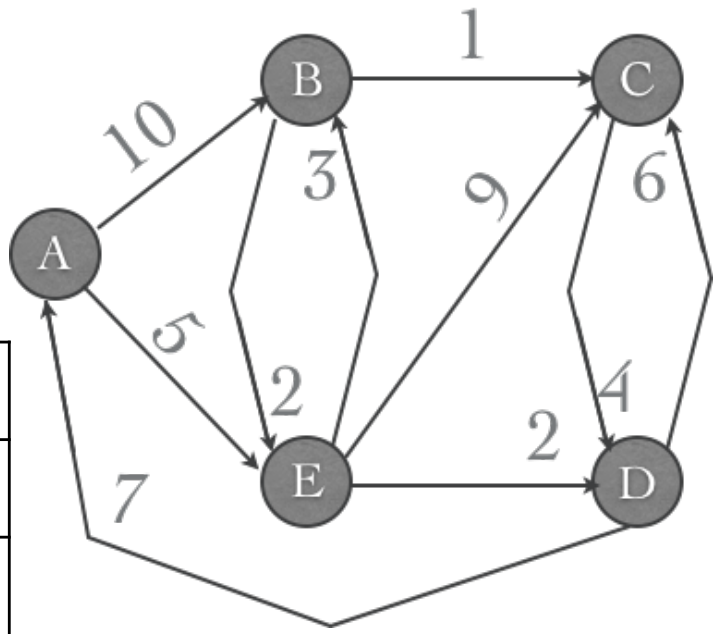
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à C : poids (C) c'est le min
poids(B) + poids(BC) = 8 + 1 = 9
poids(D) + poids(DC) = 15 + 6 = 21
poids(E) + poids(EC) = 5 + 9 = 14

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A	11,B	15,C	5,A
2	0	8,E	9,B	7,E	



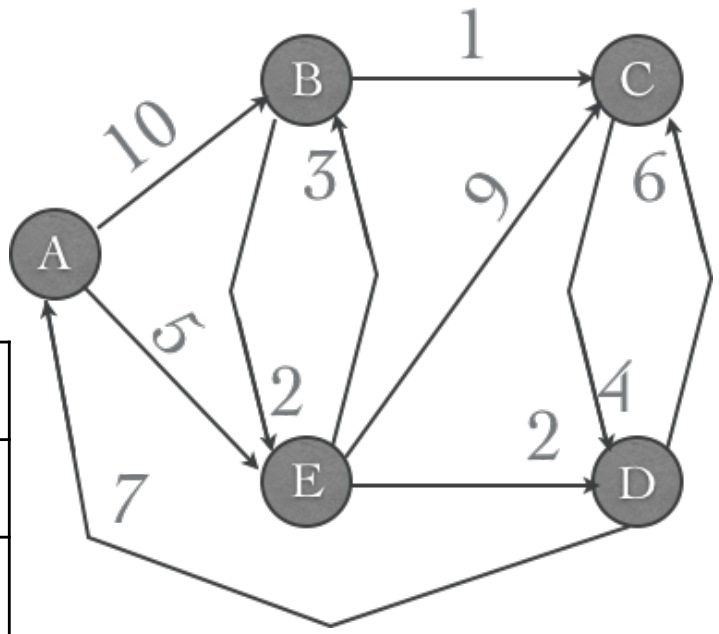
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à D : poids (D) c'est le min
poids(C) + poids(CD) = 9 + 4 = 13
poids(E) + poids(ED) = 5 + 2 = 7

Bellman-Ford

```
Bellman_Ford( G, s)
    initialisation ( G, s)  // poids des sommets mis à +infini
                            // poids du sommet initial à 0 ;
    pour i=1 à Nombre de sommets -1 faire
      |    pour chaque arc (u, v) de G faire
      |      |  paux = poids(u) + poids(arc(u, v));
      |      |  si paux <poids(v) alors
      |      |     |  pred(v) = u
      |      |     |  poids(v) = paux
    pour chaque arc (u, v) de G faire
      |   si poids(u) + poids(arc(u, v)) < poids(v) alors
      |     retourner faux
    retourner vrai
```

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	0	10,A	11,B	15,C	5,A
2	0	8,E	9,B	7,E	**5,A**



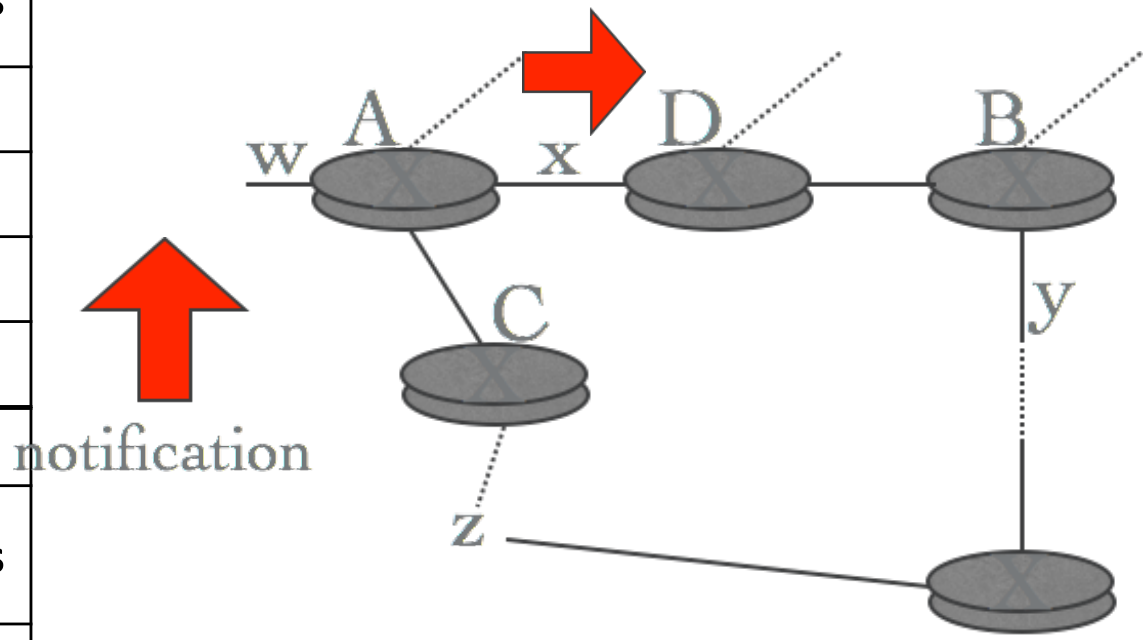
AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

PCC jusqu'à E : poids (E) c'est le min
poids(A) + poids(AE) = 0+ 5 = 5
poids(B) + poids(BE) = 8 + 2 = 10

RIP : Fonctionnement

Router D		
Sous-réseaux	**Routeur suivant**	**Nb. hops**
w	**A**	**1**
y	**B**	**1**
z	**B**	**7**
x	**-**	**0**

Router A		
Sous-réseaux	**Routeur suivant**	**Nb. hops**
w	**-**	**0**
z	**C**	**4**
x	**-**	**0**

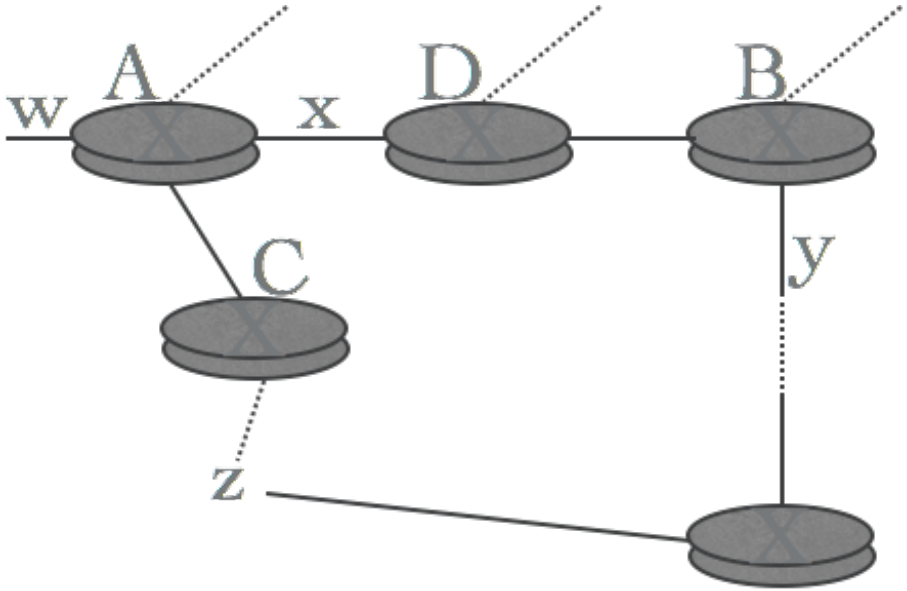


RIP : Fonctionnement

Router D		
Sous-réseaux	**Routeur suivant**	**Nb. hops**
w	**A**	**1**
y	**B**	**1**
z	**A**	**5**
x	**-**	**0**

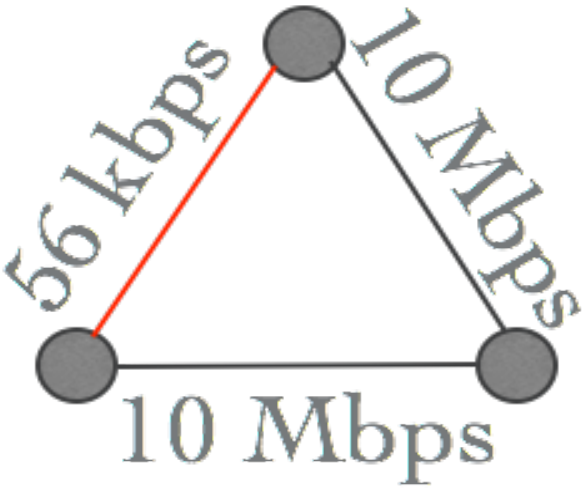
Router A		
Sous-réseaux	**Routeur suivant**	**Nb. hops**
w	**-**	**0**
z	**C**	**4**
x	**-**	**0**

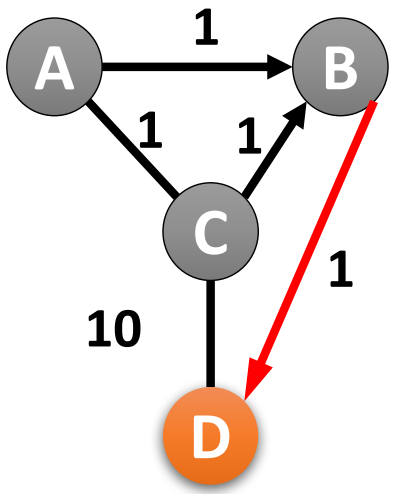
mise à jour



RIP : Limitations

- prend seulement en compte le nombre de hops
  - ne considère pas l'état de liaison pour améliorer la bande passante
  - → corrigé dans OSPF
- Problème de comptage vers l'infini si un lien (un sous-réseau IP) tombe en panne.
  - Délai de convergence pour apprendre la panne
  - Entre temps des boucles se forment.
  - Problème: proclamer l'accessibilité d'un réseau destination au voisin duquel on a appris la route vers cette destination.
  - Solutions:
    - *horizon partagé simple*: pas diffuser ces routes
    - *horizon partagé avec empoisonnement inverse*: diffuser ces routes avec leur métrique à l'infini





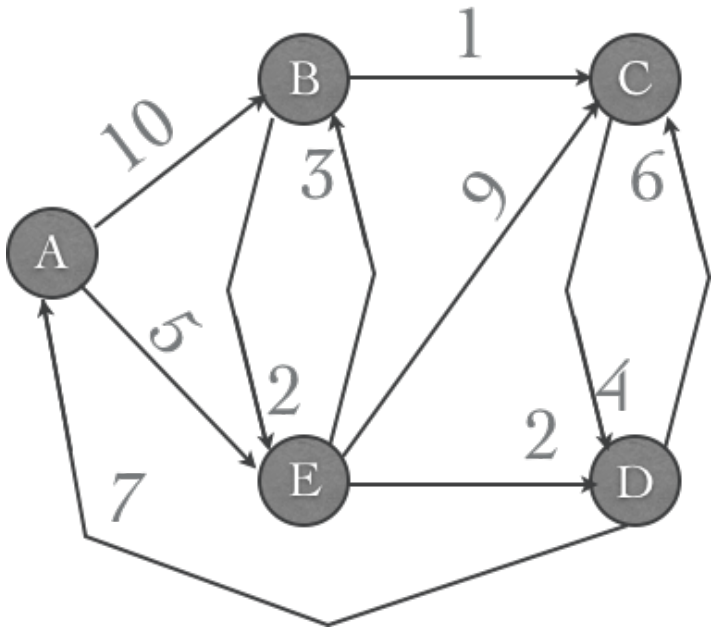
OSPF (Open Shortest Path First)

- Basé sur *algorithme d'état de lien* (*Dijkstra*)
  - coût de lien = *référence / bande passante du lien*, ou *référence = 100 Mbps*
    - Ethernet > = 100 Mbps -→ coût de lien = 1
    - Ethernet 10 Mbps -→ coût de lien = 10
  - diffuse paquets d'état de liens
  - connaît le graphe à chaque nœud
- Fonctionne par des annonces OSPF
  - portant une entrée pour chaque routeur voisin
  - diffusés sur tout le réseau (graphe) par inondation
  - encapsulés dans messages OSPF sur IP
- http://www.cisco.com/en/US/docs/internetworking/technology/handbook/OSPF.html
- Caractéristiques OSPF
  - sécurité: tous messages authentifiés
  - multi-chemins: multiplicité des chemins de même coût et équilibrage de charge
  - *Type of Service (TOS)* : association de métriques fonction TOS
  - hiérarchique: décompose grands réseaux en zones

Dijkstra

- Cherche des plus courts chemins (PCC) d'origine A
- Donne arbre couvrant

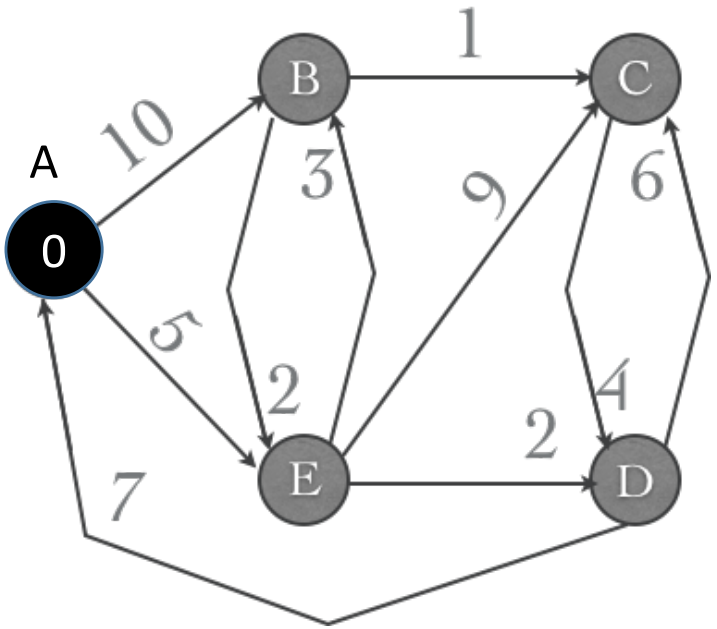
Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1					
2					
3					
4					



AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

Dijkstra

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1					
2					
3					
4					

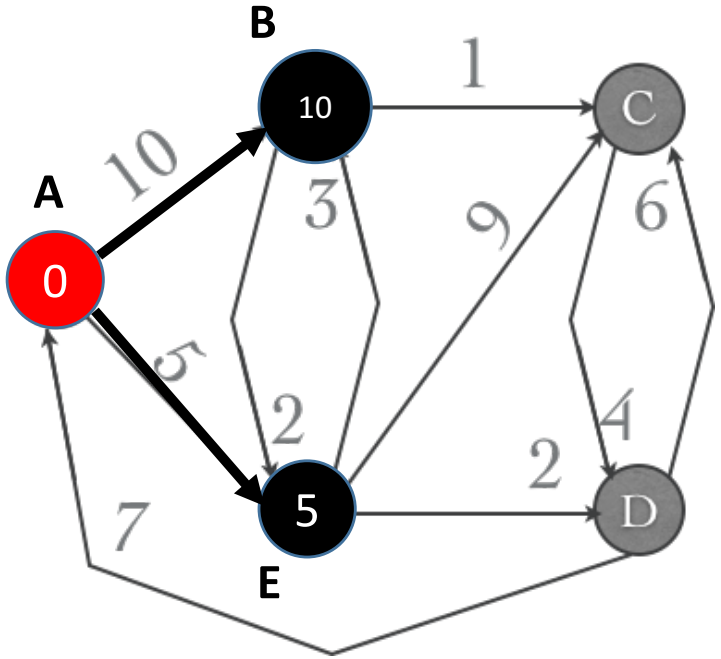


AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

Dijkstra

Qui est la destination de plus petit poids ? → A

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	*fini*	10,A	∞	∞	5,A
2	*fini*				
3	*fini*				
4	*fini*				

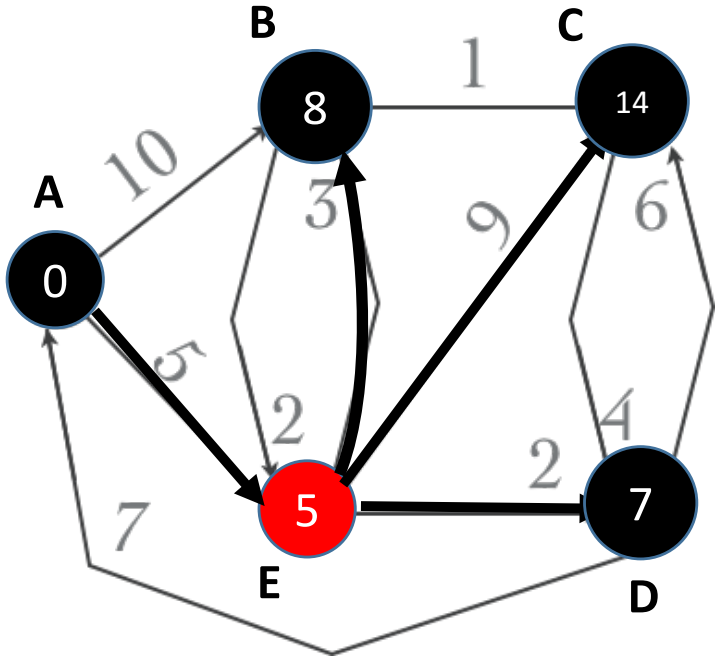


AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

Dijkstra

Qui est la destination de plus petit poids ? → E

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	*fini*	10,A	∞	∞	5,A
2	*fini*	8,E	14,E	7,E	*fini*
3	*fini*				*fini*
4	*fini*				*fini*

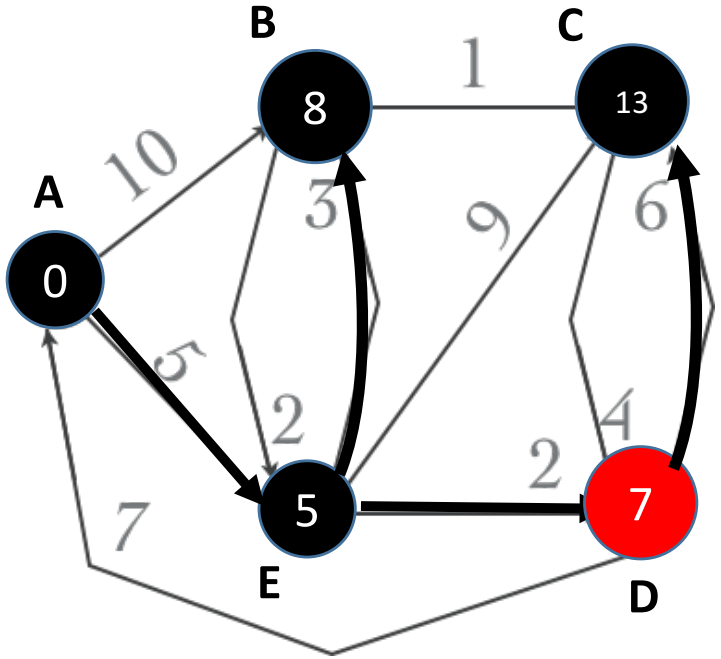


AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

Dijkstra

Qui est la destination de plus petit poids ? → D

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	*fini*	10,A	∞	∞	5,A
2	*fini*	8,E	14,E	7,E	*fini*
3	*fini*	8,E	13,D	*fini*	*fini*
4	*fini*			*fini*	*fini*

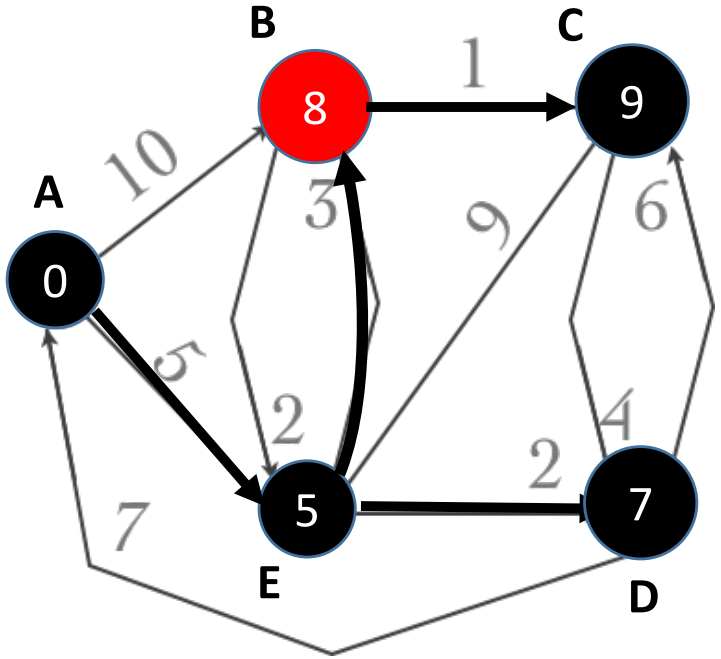


AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

Dijkstra

Qui est la destination de plus petit poids ? → B

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	*fini*	10,A	∞	∞	5,A
2	*fini*	8,E	14,E	7,E	*fini*
3	*fini*	8,E	13,D	*fini*	*fini*
4	*fini*	*fini*	9,B	*fini*	*fini*

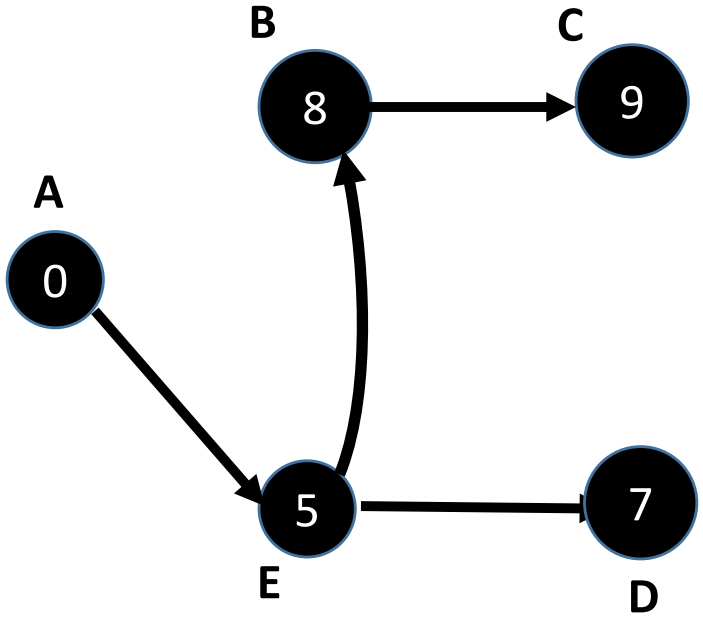


AB 10
AE 5
BC 1
BE 2
CD 4
DC 6
DA 7
EB 3
EC 9
ED 2

Dijkstra

Qui est la destination de plus petit poids ? → B

Iter.	A	B	C	D	E
0	0	∞	∞	∞	∞
1	*fini*	10,A	∞	∞	5,A
2	*fini*	8,E	14,E	7,E	*fini*
3	*fini*	8,E	13,D	*fini*	*fini*
4	*fini*	*fini*	9,B	*fini*	*fini*



Dijkstra

- Algorithme en O(|V|²)

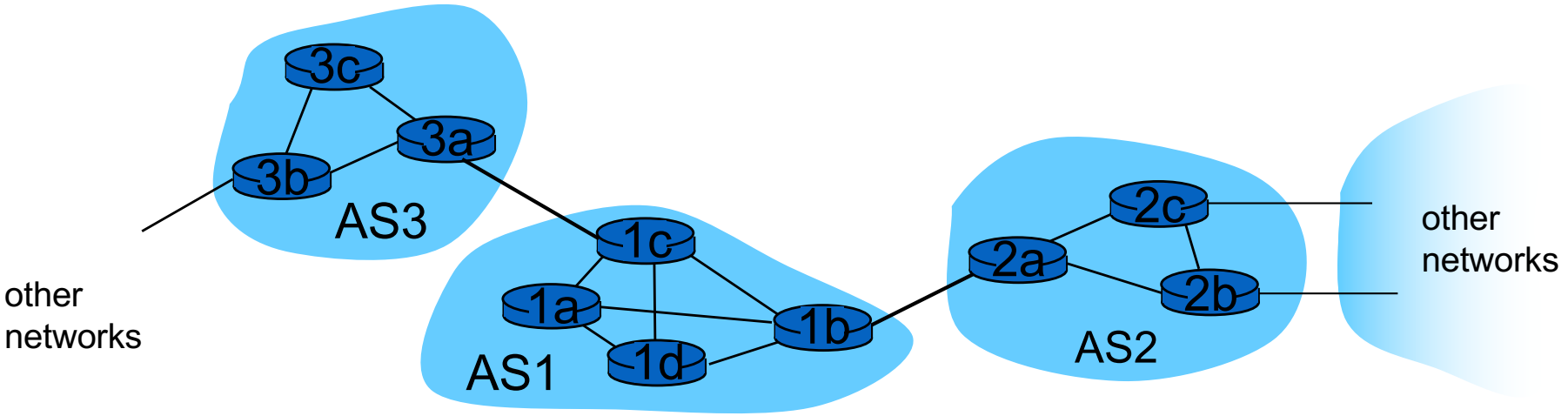
1. i=0, $S_0=\{u_0=s\}$, $L(u_0)=0$, $L(v)=\infty$
   $\forall$ $v \neq u_0$ si $|V|=1$ alors arrêt sinon aller en 2
2. $\forall$ $v \in V \setminus S_i$
   a. $L(v):=\min\{L(v) + d(u_i,v)\}$
   b. si $L(v)$ remplacé, mémoriser $(L(v),u_i)$ sur v
3. $u_{i+1}=\min_v\{L(v): v \in V \setminus S_i\}$
4. $S_{i+1}=S_i \cup \{u_{i+1}\}$
5. i++
   a. si $i=|V|-1$ alors arrêt sinon aller en 2

Comparaison BF vs Dijkstra (algos LS vs DV)

- Complexité du messages màj
  - LS: avec *V* nœuds, *E* liens E, O(*VE*) messages envoyés
  - DV: échange entre voisins uniquement : temps de convergence varie
- Vitesse de convergence
  - LS: O(n^2) requiert O (nE) messages.
    - Il peut y avoir des oscillations (si poids du lien est égal à la quantité de trafic transporté)
  - DV: le temps de convergence varie:
    - Il peut y avoir des boucles de routage
    - Problème de comptage vers l'infini (si augmentation importante du cout de lien)
- Robustesse: que se passe-t-il si le routeur fonctionne mal ?
  - LS:
    - Nœud peut annoncer ***un coût de lien incorrect***
    - Chaque nœud calcule uniquement sa propre table
  - DV:
    - Le nœud DV peut annoncer ***un coût de chemin incorrect***
    - La table de chaque nœud utilisée par d'autres : erreurs se propagent par le réseau

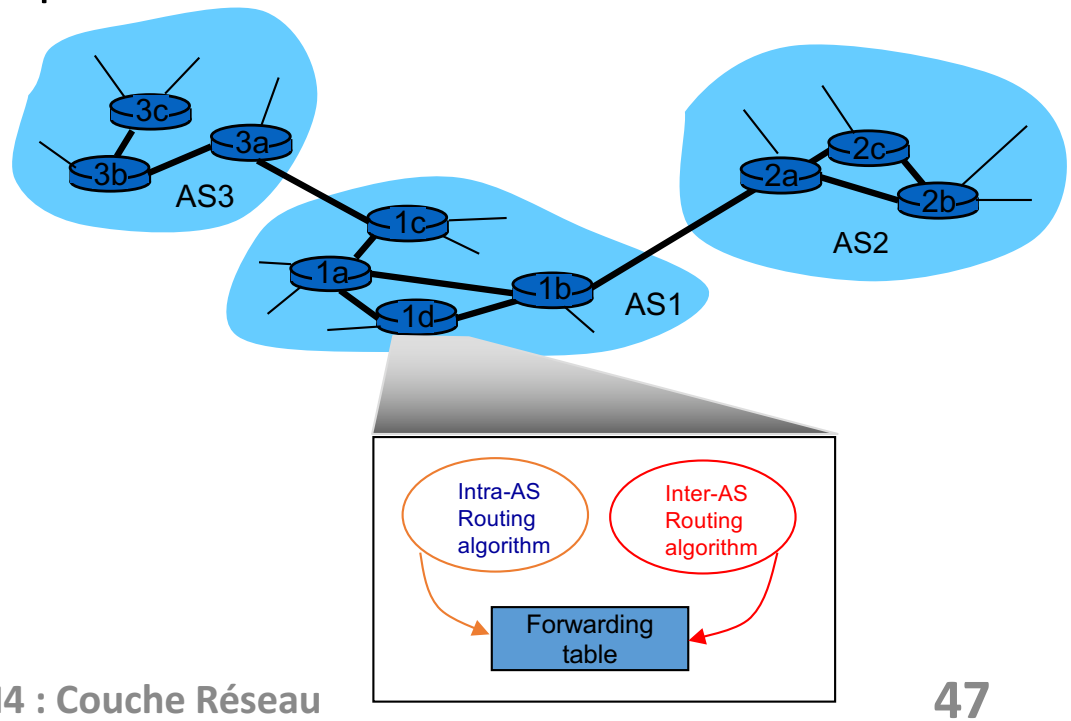
Systèmes autonomes

- Système autonome : *Autonomous System (AS)*
  - Ensemble de réseaux IP sous le contrôle d'une seule et même entité (p. ex. FAI)
- Notion administrative caractérisée par l'unité de décision en son sein qui définit une politique de ***routage***
- A l'opposé, Internet : réseau public pouvant prendre des décisions contradictoires
- Routeurs de passerelle (*gateway*) au "bord" de son propre AS avec un lien vers le routeur dans un autre AS



Systèmes autonomes

- Types de protocoles de routage par rapport à AS:
  - **Intra AS :** définit entrées pour destinations internes
    - Routeurs internes dans le même AS exécutent le même protocole de routage ***intra-AS***
    - Routeurs dans différents AS peuvent exécuter différents protocoles de routage ***intra-AS***
  - **Inter AS :** définit entrées pour destinations externes
- Tables de routages configurées par les deux



Routage externe (inter AS)

BGP: Border Gateway Protocol [rfc 4271]

- standard de facto
- **protocole à vecteur de chemins**
  - Comme un protocole à vecteur de distances, mais la place des distances, on propage la route de AS traversés (chemin) :

    `PREFIX` *(Destination)* / `NEXT-HOP` / `AS-PATH` *(chemin)*
  - Dans le routage externe *: on n'a pas une vision complète de la topologie de l'internet !!!!!*
- permet à chaque système autonome:
  - d'obtenir la liste d'accessibilité des AS voisins
  - de propager la liste aux routeurs internes
  - de calculer les bonnes routes
- permet à tout sous-réseaux de signaler son existence

Routage externe (inter AS)

- **BGP: Border Gateway Protocol [rfc 4271]**
- prend en compte des politiques de routage géopolitiques, économiques, sécurité …
  - Par exemple, ne jamais parcourir AS x
- selon catégorie AS :
  - sans issue, multi-connectés, de transit
  - fournisseur, client
- communication entre routeurs BGP
  - fiabilité par sessions TCP et détails des réseaux traversés dissimulés
- algorithme de type vecteur de distance
  - communique chemins complets et coûts et suppression boucles
- Routeur peut apprendre plus d'une route à l'AS, sélectionne l'itinéraire basé sur:
  1. Politique de routage local
  2. Le plus court `AS-PATH`
  3. Routeur `NEXT-HOP` le plus proche
  4. Critères supplémentaires

Routage interne (intra AS)

❖ Aussi connu sous le nom *IGP: interior gateway protocol :*

- *Link State*
  - **OSPF: open shortest path first (recommandé)**
  - IS-IS: intermediate system to IS (ISO)
- *Distance vector*
  - **RIP: routing information protocol (BSD-UNIX)**
  - IGRP: interior gateway routing protocol (cisco)