Couche Internet

Chapitre 3 : le routage

Arnold Hien

Université d'Orléans

L3 Réseaux 1

2026-2027

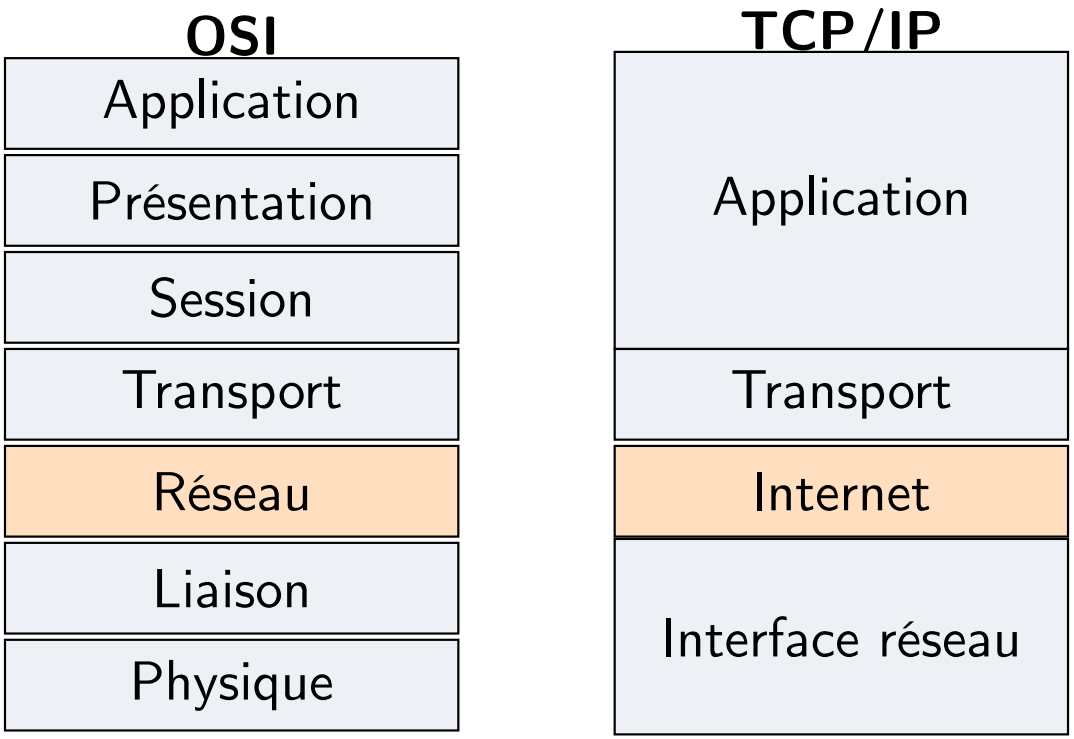
Plan du chapitre

1. Le problème du routage
2. Vecteurs de distance (DV)
3. États de liens (LS)
4. Routage externe

Le problème du routage

1. Le problème du routage
2. Vecteurs de distance (DV)
3. États de liens (LS)
4. Routage externe

Les modèles



Ce chapitre : la couche Réseau / Internet.

Rôle de la couche réseau

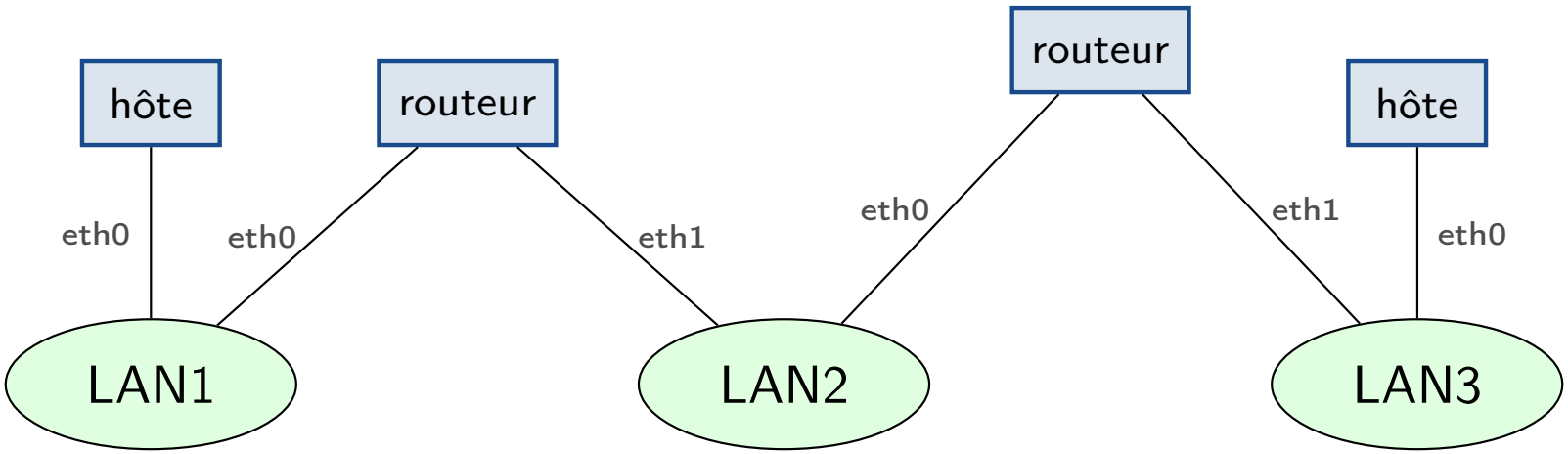
Objectif

Chaque machine doit pouvoir contacter n'importe quelle autre machine sur le réseau. Pour cela, qu'elle soit sur le même réseau local ou non, il faut pouvoir :

- l'identifier : adressage IP
- la joindre : routage

L'adressage IP a été vu au chapitre précédent ; ce chapitre traite du routage.

Interconnexion de LAN

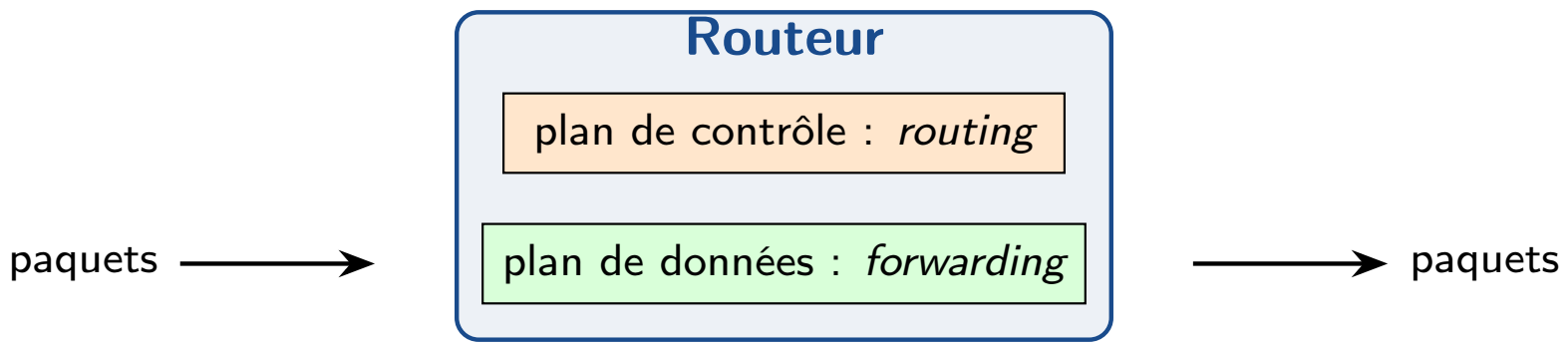


- Chaque interface a une adresse IP (e.g. DHCP)
- tous les hôtes en bord de réseau connaissent une passerelle
- comment remplir les tables de routage des routeurs ?

Routeur : double mission

Les routeurs doivent :

- transférer les paquets (*forwarding*) : **plan de données** ;
- établir la table de routage (*routing*) : **plan de contrôle**.

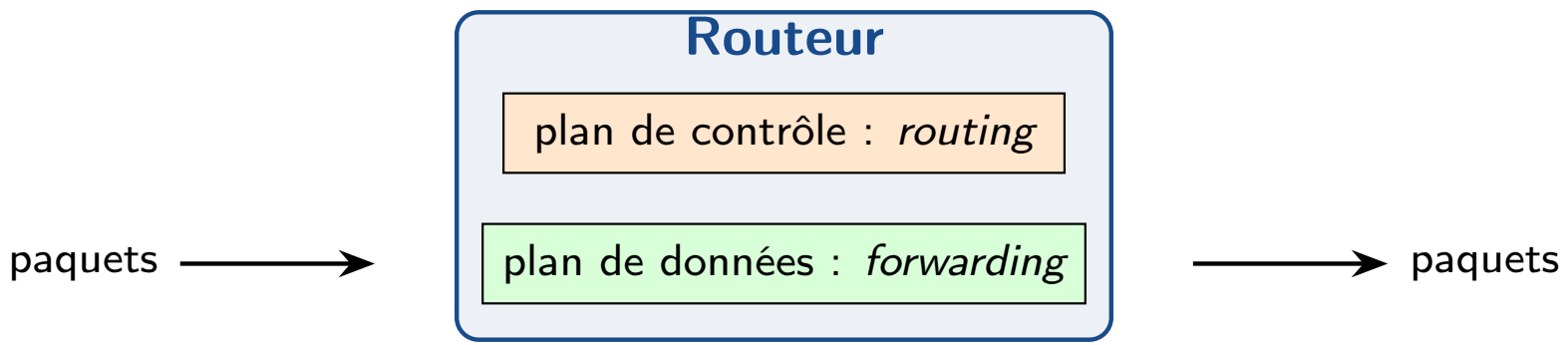


Ce chapitre : le plan de contrôle (comment construire les tables ?).

Routeur : double mission

Les routeurs doivent :

- transférer les paquets (*forwarding*) : **plan de données** ;
- établir la table de routage (*routing*) : **plan de contrôle**.



Ce chapitre : le plan de contrôle (comment construire les tables ?).

Comment choisir le chemin ?

- Comment trouver le plus court chemin ?
- Comment mesurer la distance ?
- Comment gérer les évolutions du réseau ?

Routage dynamique : algorithmique de graphe

L'algorithme doit être :

- fiable : pas de paquet perdu à cause du routage
- simple : décision facile à prendre, pré-calcul peu coûteux (temps et occupation du réseau)

Il doit pouvoir s'adapter au réseau :

- mise à jour si un routeur disparaît du graphe
- changement de route si le réseau devient surchargé

Algorithme au choix

AS (système autonome) : défini par une politique de routage commune.
Plusieurs algorithmes coexistent : à l'intérieur des AS, entre AS . . .

Les options :

- Algorithme centralisé
- Algorithme distribué :
  - à connaissance globale
  - à connaissance locale

Nous verrons un algorithme distribué de chaque type :

- vecteurs de distance (connaissance locale)
- états de liens (connaissance globale)

Algorithme au choix

AS (système autonome) : défini par une politique de routage commune.
Plusieurs algorithmes coexistent : à l'intérieur des AS, entre AS . . .

Les options :

- Algorithme centralisé
- Algorithme distribué :
  - à connaissance globale
  - à connaissance locale

Nous verrons un algorithme distribué de chaque type :

➠ vecteurs de distance (connaissance locale)

➠ états de liens (connaissance globale)

Algorithme au choix

AS (système autonome) : défini par une politique de routage commune.
Plusieurs algorithmes coexistent : à l'intérieur des AS, entre AS . . .

Les options :

- Algorithme centralisé
- Algorithme distribué :
  - à connaissance globale
  - à connaissance locale

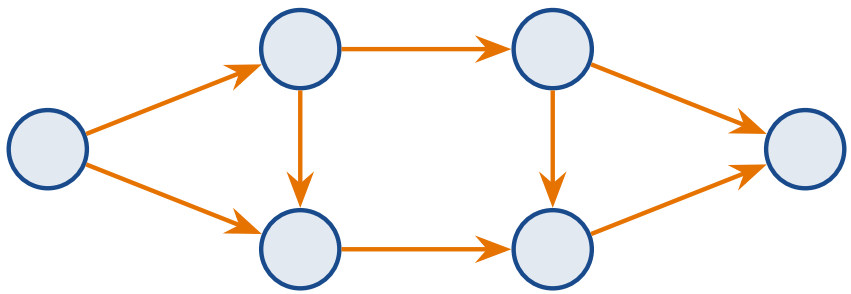
Nous verrons un algorithme distribué de chaque type :

➠ vecteurs de distance (connaissance locale)

➠ états de liens (connaissance globale)

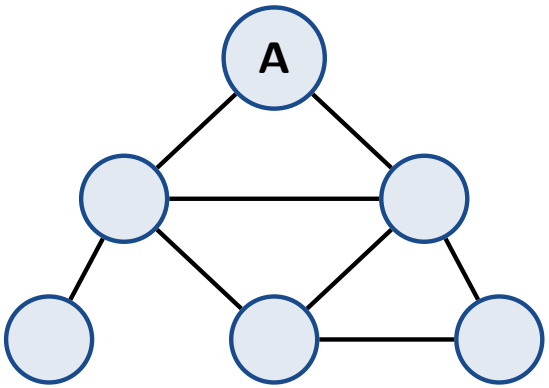
Algorithme par inondation

- Chaque paquet est envoyé en broadcast sur l'ensemble du réseau
- Tous les routeurs transmettent à tous leurs voisins
- TTL pour éviter les boucles infinies (diamètre d'internet ?)
- Fiabilité, simplicité d'usage

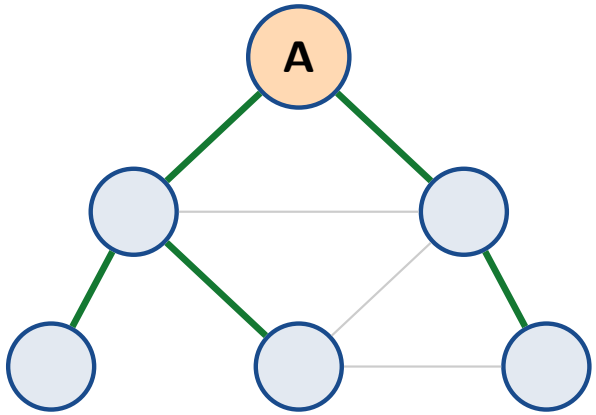


Coûteux en trafic, mais utile : servira à diffuser la topologie (états de liens).

Comment choisir le chemin ? Arbre des plus courts chemins



le graphe du réseau



l'arbre des plus courts chemins depuis A

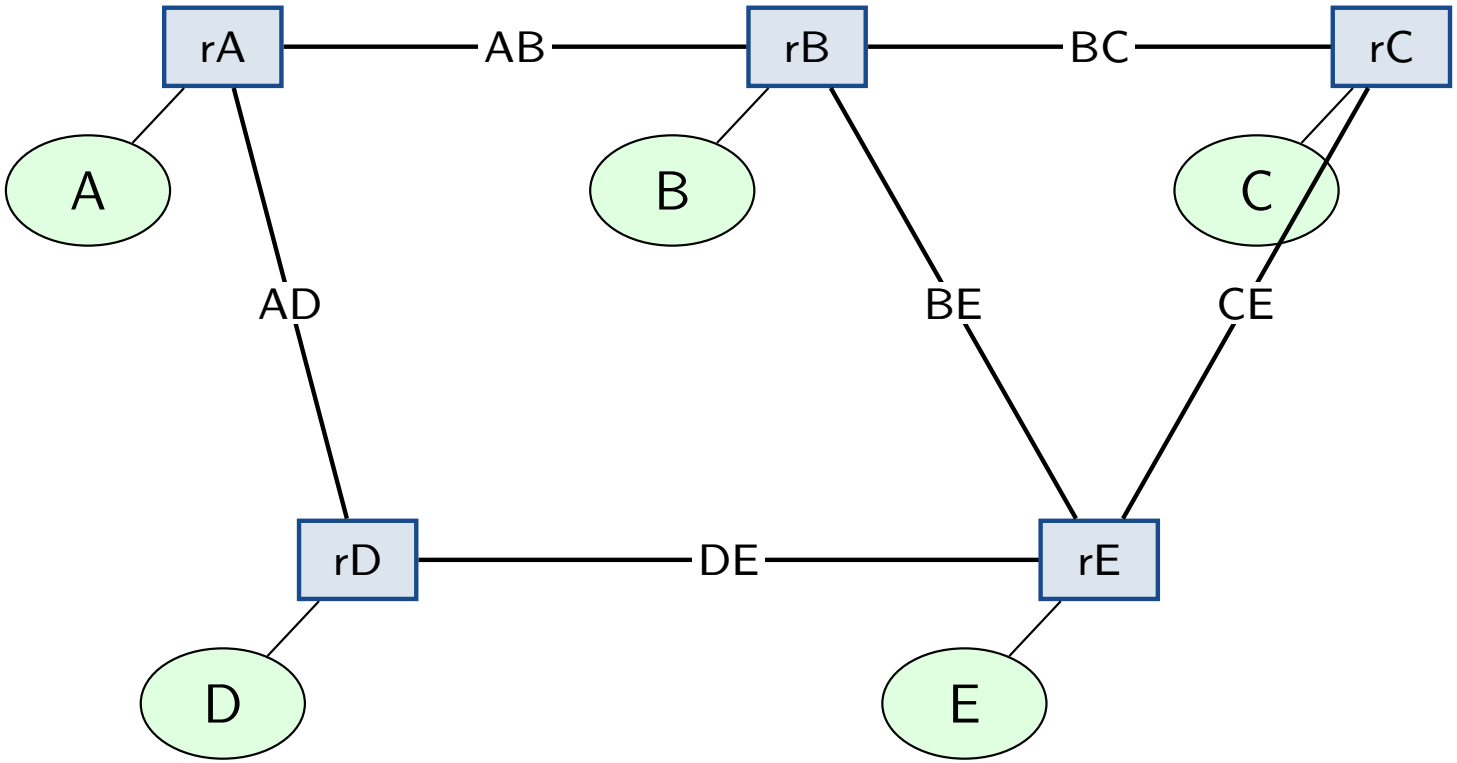
Chaque routeur doit établir son propre arbre des plus courts chemins vers toutes les destinations.

Quelle métrique ?

Chaque routeur doit établir son propre arbre. Quelle métrique ?

- Corrélée au délai. Calcul par tests
- Corrélée à la congestion: contournement des zones noires. Bison Futé ?
- Corrélée à la capacité: plus elle est grande, plus la distance est faible
- Sans information : distance de 1 par saut

Le réseau test



Cinq réseaux A, B, C, D, E, chacun desservi par un routeur rA, rB, rC, rD, rE. Six liens : AB, BC, AD, BE, CE, DE. Ce réseau servira d'exemple dans tout le chapitre.

Note de lecture des slides

Dans la suite, les tables de routage sont simplifiées, en particulier les interfaces ne sont pas mentionnées (on peut ici les déduire sans ambiguïté).

On notera la différence entre tables de routage (connues depuis le chapitre précédent) et tables RIP que nous allons découvrir.

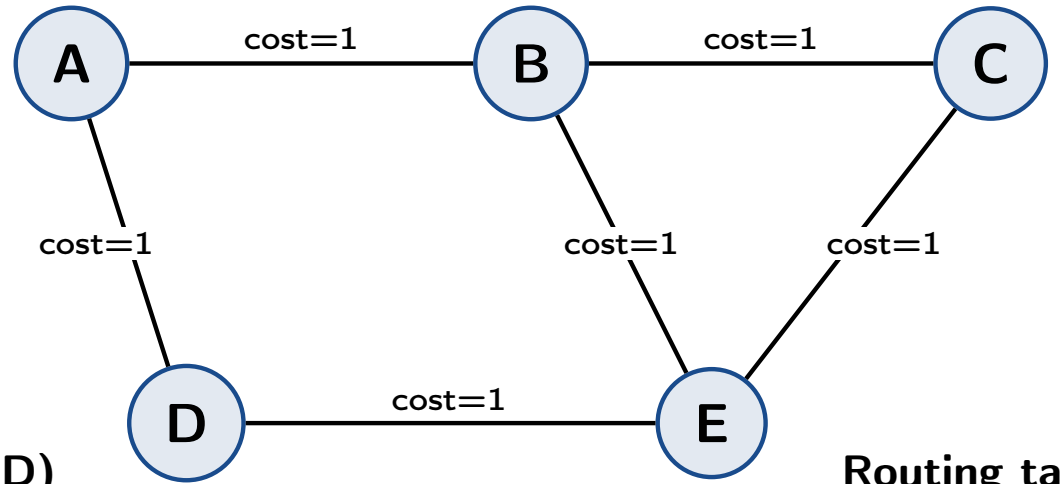
Sur les schémas suivants, un cercle « X » représente le réseau X et son routeur rX.

Les tables de routage désirées

Routing table (A)
A : local
B : rB
C : rB
D : rD
E : rD

Routing table (B)
A : rA
B : local
C : rC
D : rE
E : rE

Routing table (C)
A : rB
B : rB
C : local
D : rE
E : rE



Routing table (D)
A : rA
B : rA
C : rE
D : local
E : rE

Routing table (E)
A : rD
B : rB
C : rC
D : rD
E : local

Objectif : que chaque routeur construise automatiquement cette table.

Pause exercice — Wooclap

Routage



Code : **ARRSKRI**

Connectez-vous à `wooclap.com` avec le code de la séance.

Vecteurs de distance (DV)

1. Le problème du routage
2. Vecteurs de distance (DV)
3. États de liens (LS)
4. Routage externe

Bellman-Ford

Première option : distribué à connaissance locale

- Algorithme de Bellman-Ford pour calculer les plus courts chemins dans le graphe
- Chaque routeur connaît le coût de ses liens (calcul ou hypothèse)
- Chaque routeur transmet à ses voisins la liste des destinations connues avec leurs distances
- À la réception, le routeur met à jour sa table de routage

Notations

Dans ce qui suit :

- `R` est la table de routage à vecteurs de distance
- `d` est utilisé pour un réseau destination
- les `link` sont les domaines de collision sur lesquels on communique

Les vecteurs de distance sont envoyés par un routeur sur un domaine de collision (`link`), indifféremment à tous les routeurs participant au protocole RIP sur ce domaine de collision

Émission périodique

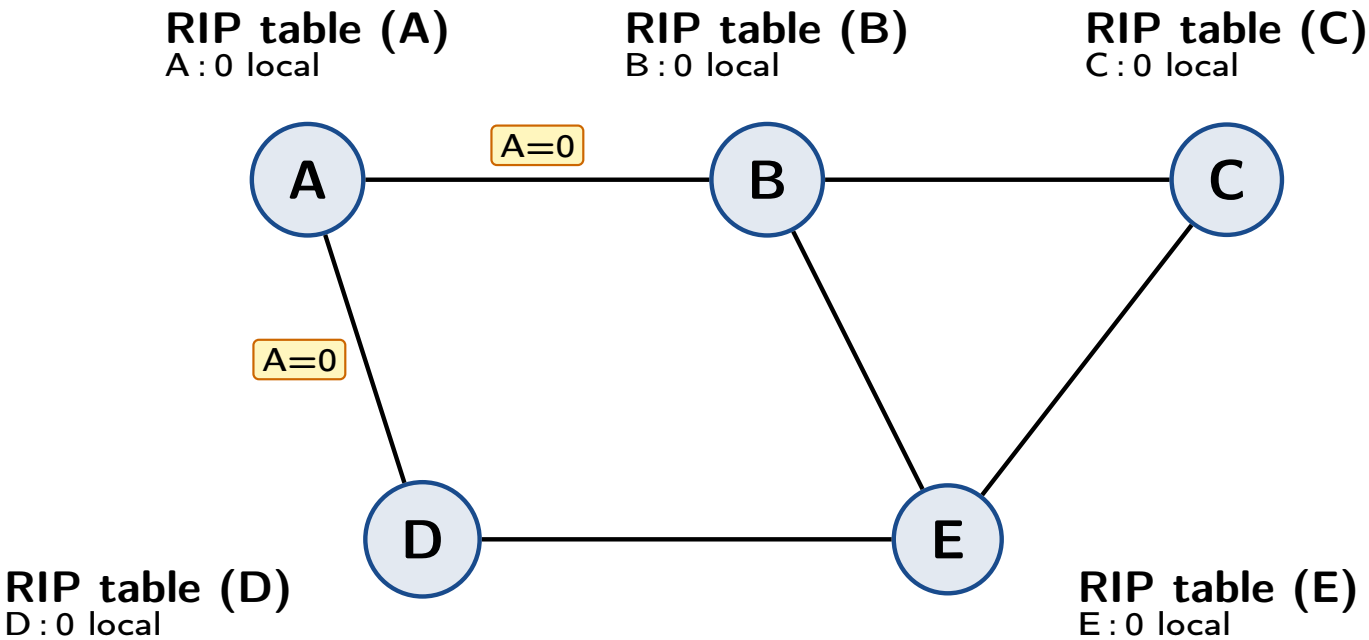
```
loop indefinitely {
    sleep(N seconds)
    V=[]
    for each destination d in R[] {
        V[d]=R[d].cost
    }
    for each link l {
        l.send(V)
    }
}
```

Toutes les N secondes, on envoie sur chaque lien le vecteur des distances connues

En réception

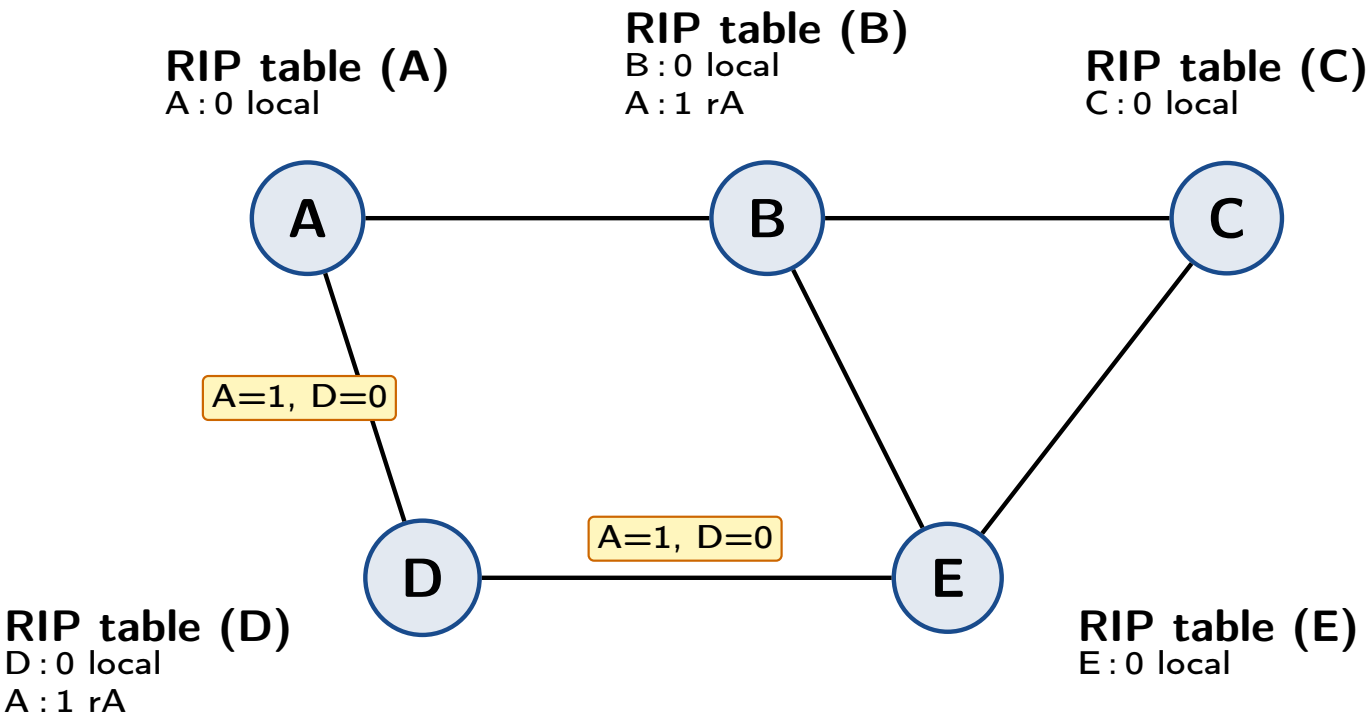
```
receive(Vector V[], router r)
{
    for each d in V[]
    {
        if (d in R) {
            if (V[d]+1.cost < R[d].cost) {
                R[d].cost=V[d]+1.cost
                R[d].gw=r
            }
        } else {
            R[d].cost=V[d]+1.cost
            R[d].gw=r
        }
    }
}
```

Exemple (les coûts sont tous 1)



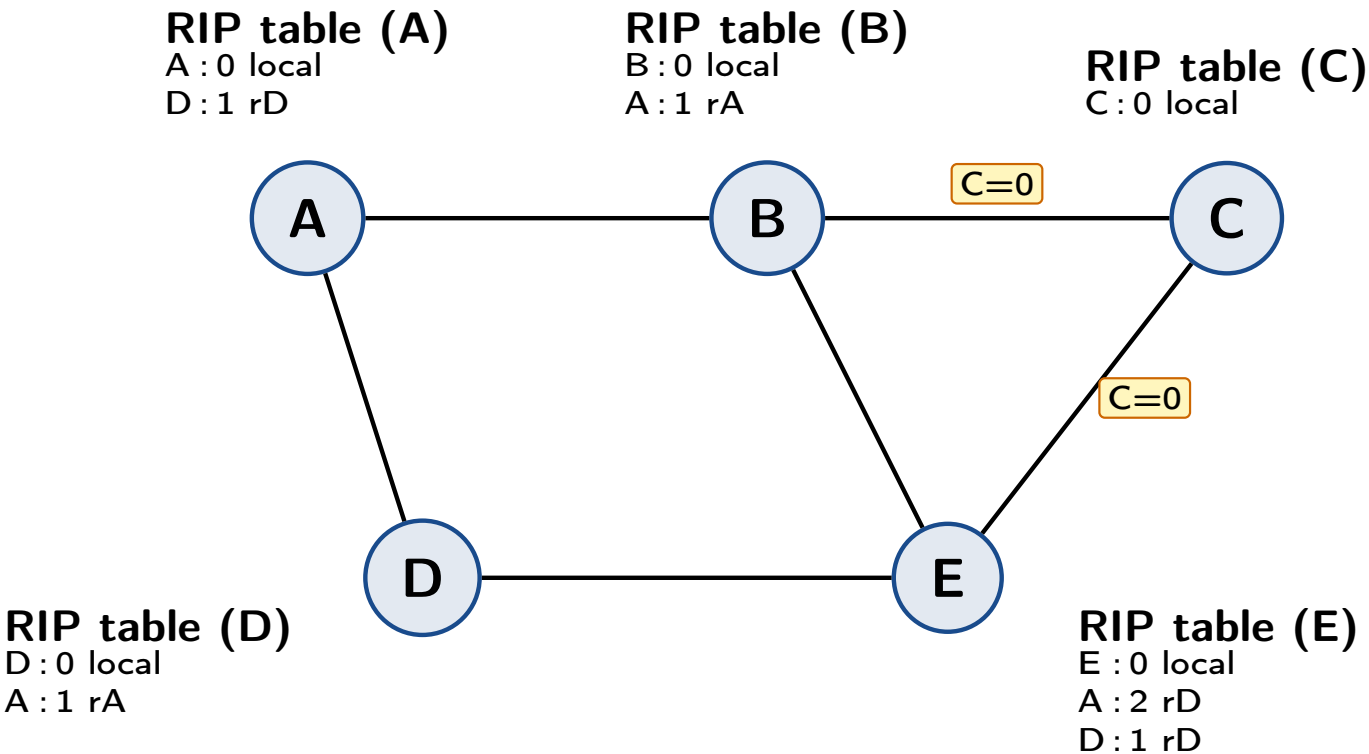
Chaque routeur ne connaît que son réseau local. A émet son vecteur.

Exemple (les coûts sont tous 1)



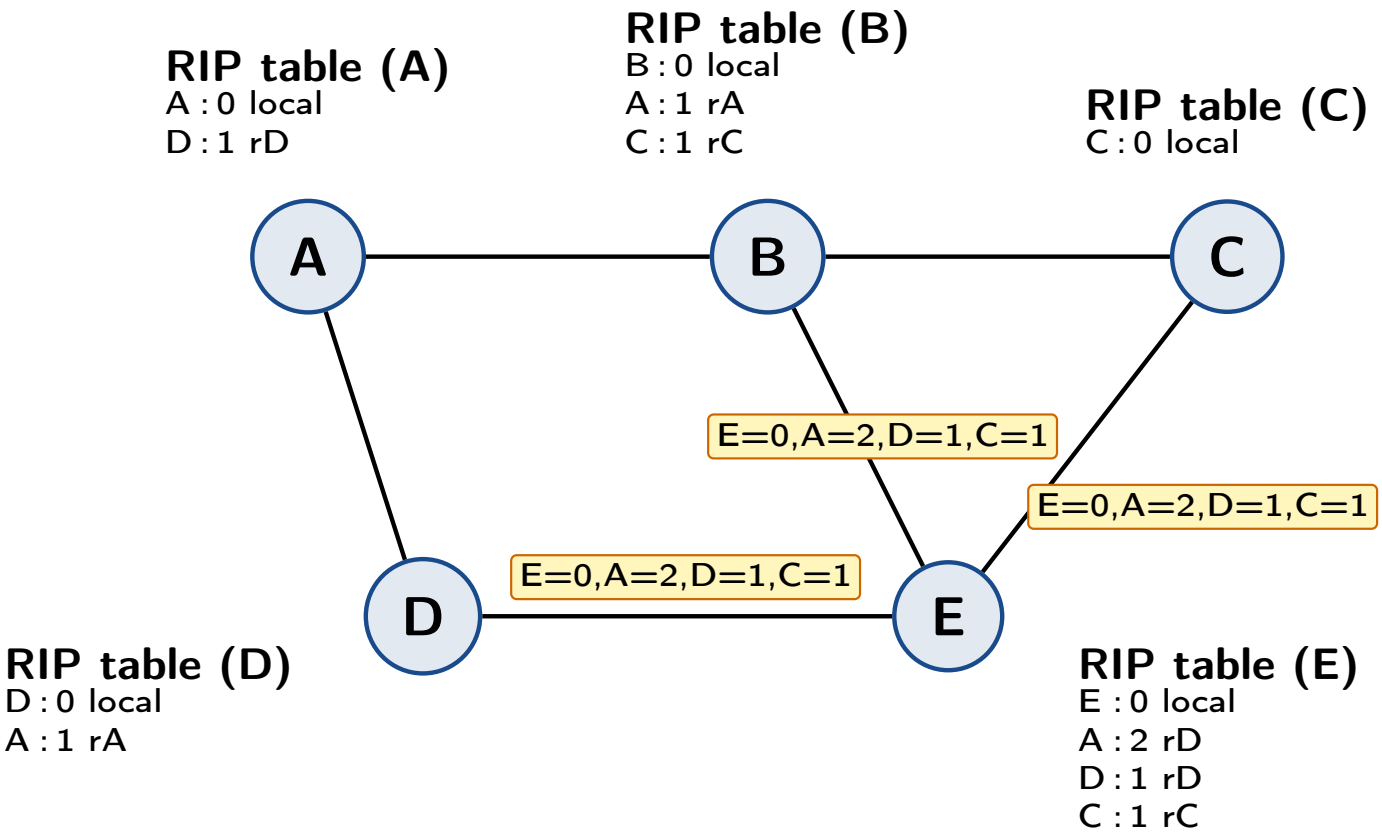
B et D ont appris A. D émet à son tour (vers A et E).

Exemple (les coûts sont tous 1)



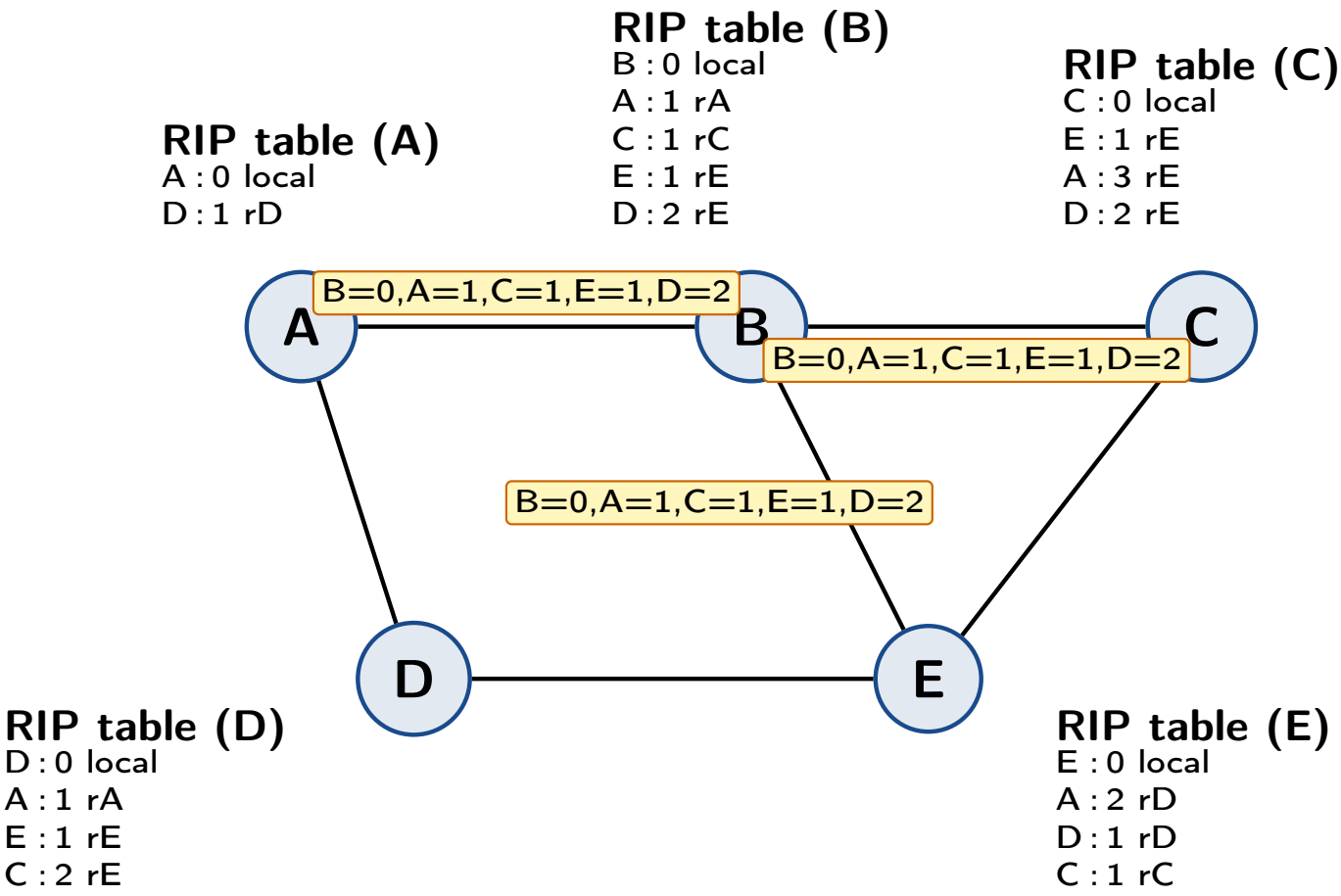
A connaît D ; E connaît A (coût 2) et D. C émet.

Exemple (les coûts sont tous 1)



B et E connaissent C. E émet son vecteur complet (vers B, C, D).

Exemple (les coûts sont tous 1)



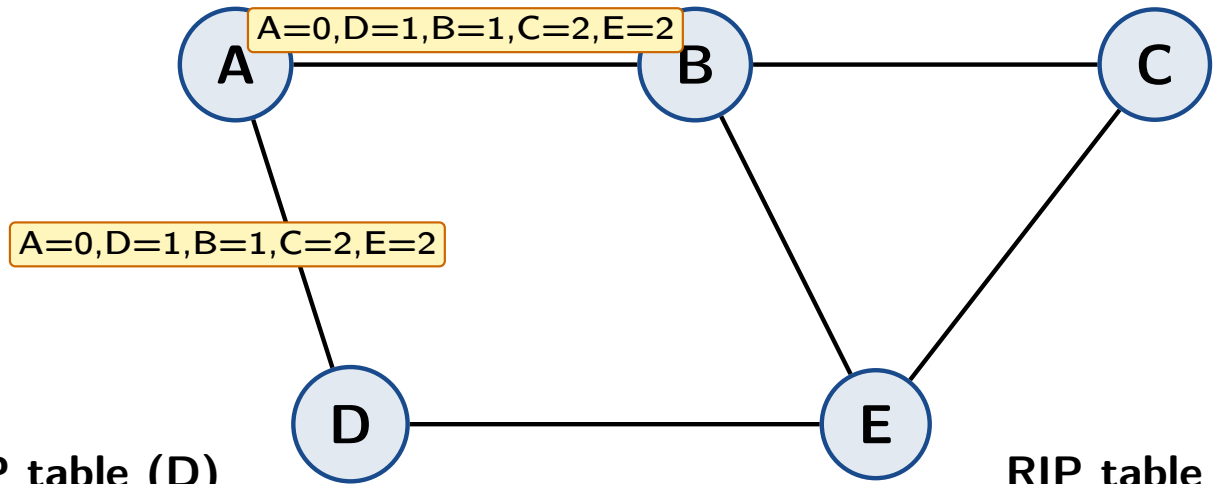
B, C, D se sont mis à jour grâce à E. B émet (vers A, C, E).

Exemple (les coûts sont tous 1)

RIP table (A)
A : 0 local
D : 1 rD
B : 1 rB
C : 2 rB
E : 2 rB

RIP table (B)
B : 0 local
A : 1 rA
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 2 rB
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

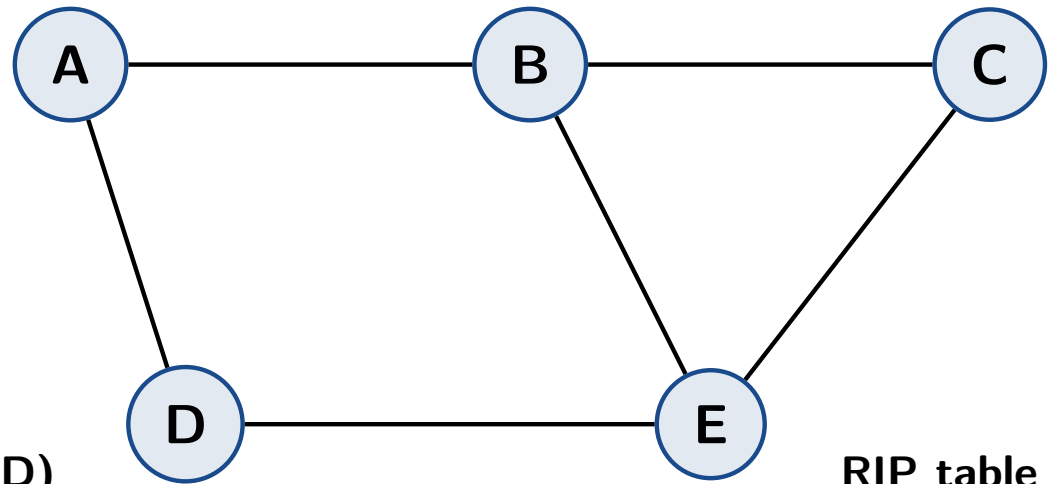
A, C, E se sont mis à jour grâce à B. A émet à nouveau (vers B, D).

Exemple (les coûts sont tous 1)

RIP table (A)
A : 0 local
D : 1 rD
B : 1 rB
C : 2 rB
E : 2 rB

RIP table (B)
B : 0 local
A : 1 rA
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 2 rB
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : 2 rA

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

Fin : D a appris B via rA. Toutes les tables sont complètes et optimales.

En cas de problème

Sans perte de lien, tous les routeurs finissent par atteindre une version optimale (donc stable) de la table de routage.

Si un lien se perd :

- il faut pouvoir le détecter :
  - faisable en couches basses
  - ou en considérant qu'un délai prolongé sans réception de vecteur de distance signifie la perte du lien
- et il faut recalculer les nouvelles tables de routage.

En cas de problème

Sans perte de lien, tous les routeurs finissent par atteindre une version optimale (donc stable) de la table de routage.

Si un lien se perd :

- il faut pouvoir le détecter :
  - faisable en couches basses
  - ou en considérant qu'un délai prolongé sans réception de vecteur de distance signifie la perte du lien
- et il faut recalculer les nouvelles tables de routage.

En réception (version 2)

On accepte aussi la mise à jour venant de la passerelle actuelle, même si le coût augmente :

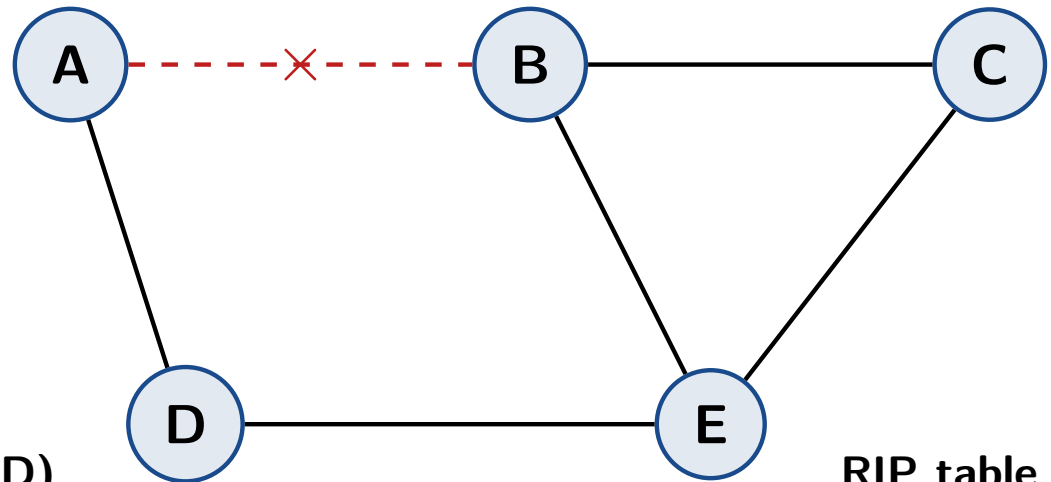
```
receive(Vector V[], router r) {
    for each d in V[] {
        if (d in R) {
            if ((V[d]+l.cost < R[d].cost) or (R[d].gw=r)) {
                R[d].cost=V[d]+l.cost
                R[d].gw=r
            }
            } else {
                R[d].cost=V[d]+l.cost
                R[d].gw=r
            }
        }
    }
```

Exemple avec lien perdu (A–B)

RIP table (A)
A : 0 local
D : 1 rD
B : ∞
C : ∞
E : ∞

RIP table (B)
B : 0 local
A : ∞
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 2 rB
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : 2 rA

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

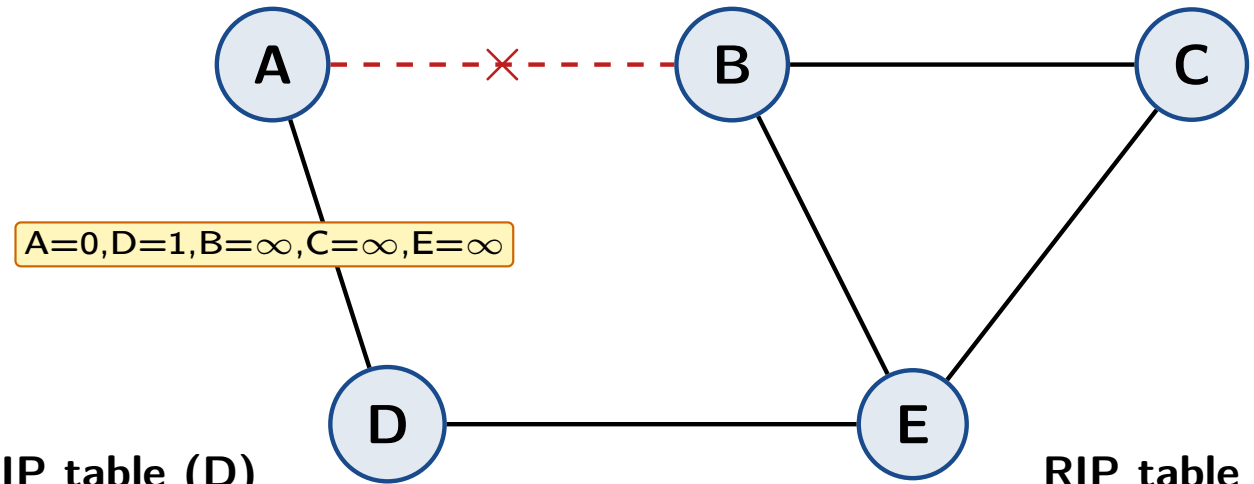
A et B détectent la perte : toutes leurs routes via ce lien passent à ∞.

Exemple avec lien perdu (A–B)

RIP table (A)
A : 0 local
D : 1 rD
B : ∞
C : ∞
E : ∞

RIP table (B)
B : 0 local
A : ∞
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 2 rB
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : 2 rA

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

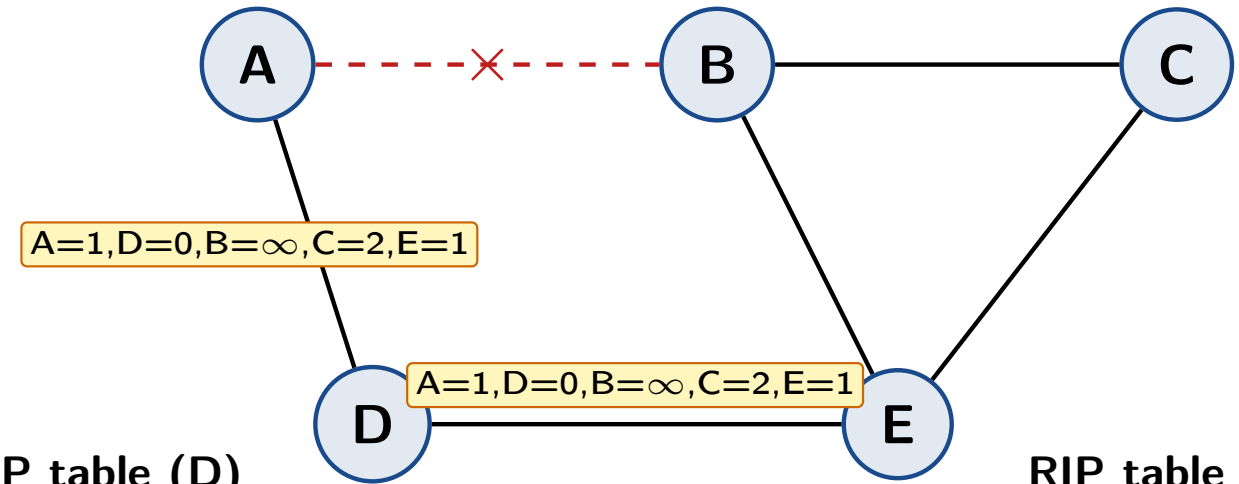
A émet son vecteur vers D.

Exemple avec lien perdu (A–B)

RIP table (A)
A : 0 local
D : 1 rD
B : ∞
C : ∞
E : ∞

RIP table (B)
B : 0 local
A : ∞
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 2 rB
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : ∞

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

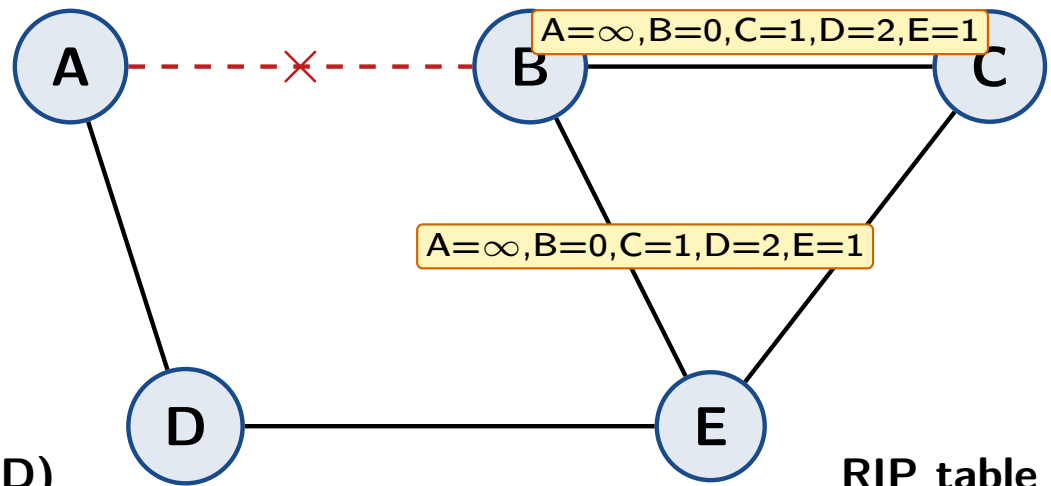
D met B à ∞ (règle « gw=r » : la route passait par rA). D émet.

Exemple avec lien perdu (A–B)

RIP table (A)
A : 0 local
D : 1 rD
B : ∞
C : 3 rD
E : 2 rD

RIP table (B)
B : 0 local
A : ∞
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 2 rB
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : ∞

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

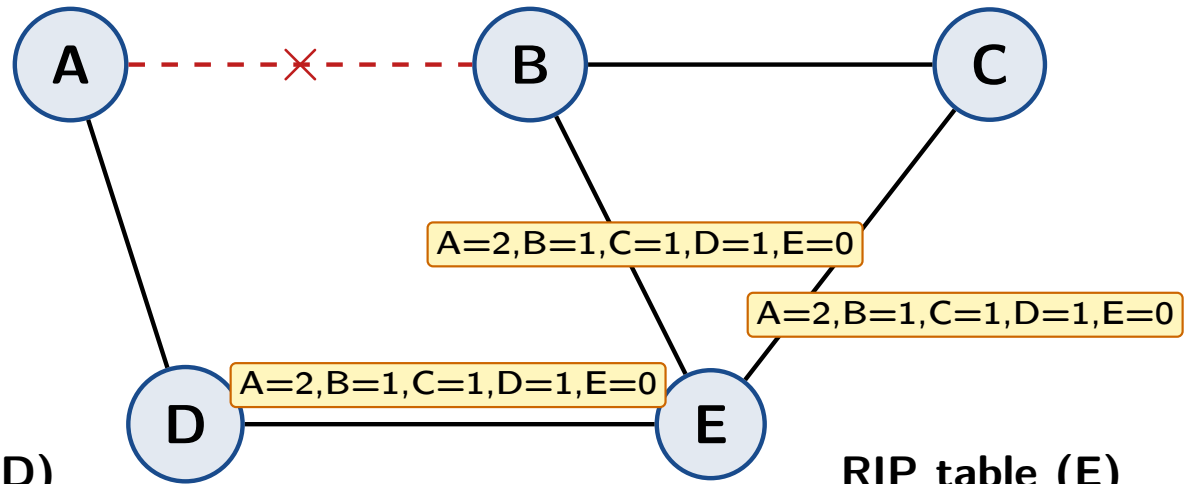
A retrouve C (coût 3) et E (coût 2) via rD. B émet vers C et E.

Exemple avec lien perdu (A–B)

RIP table (A)
A : 0 local
D : 1 rD
B : ∞
C : 3 rD
E : 2 rD

RIP table (B)
B : 0 local
A : ∞
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : ∞
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : ∞

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

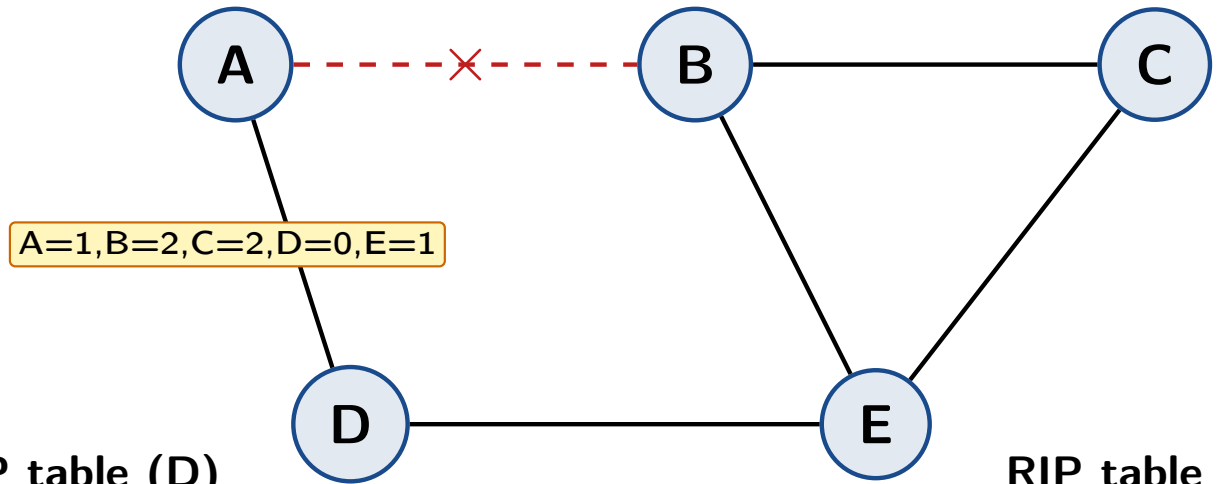
C perd A (∞, sa route passait par rB). E émet vers B, C, D.

Exemple avec lien perdu (A–B)

RIP table (A)
A : 0 local
D : 1 rD
B : ∞
C : 3 rD
E : 2 rD

RIP table (B)
B : 0 local
A : 3 rE
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 3 rE
D : 2 rE
B : 1 rB



RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : 2 rE

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

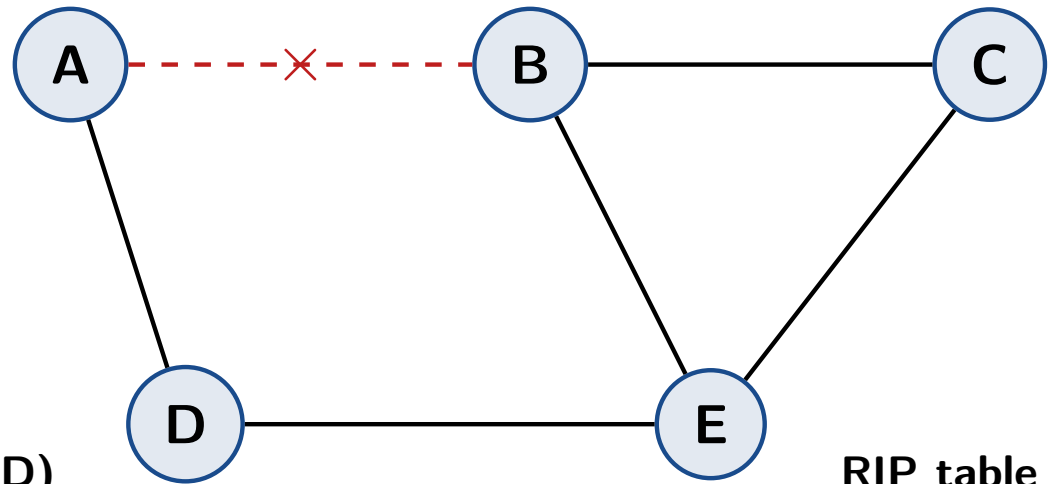
B, C retrouvent A via rE (coût 3) ; D retrouve B via rE. D émet vers A.

Exemple avec lien perdu (A–B)

RIP table (A)
A : 0 local
D : 1 rD
B : 3 rD
C : 3 rD
E : 2 rD

RIP table (B)
B : 0 local
A : 3 rE
C : 1 rC
E : 1 rE
D : 2 rE

RIP table (C)
C : 0 local
E : 1 rE
A : 3 rE
D : 2 rE
B : 1 rB



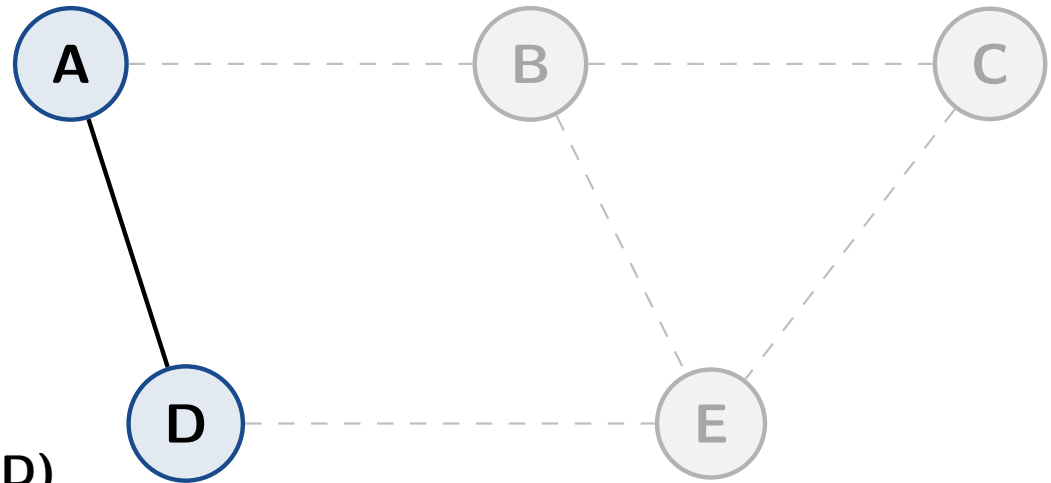
RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : 2 rE

RIP table (E)
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

Fin : A rejoint B en 3 sauts via rD. Nouvel état stable et optimal.

Perte de connexité

RIP table (A)
A : 0 local
D : 1 rD
B : 3 rD
C : 3 rD
E : 2 rD

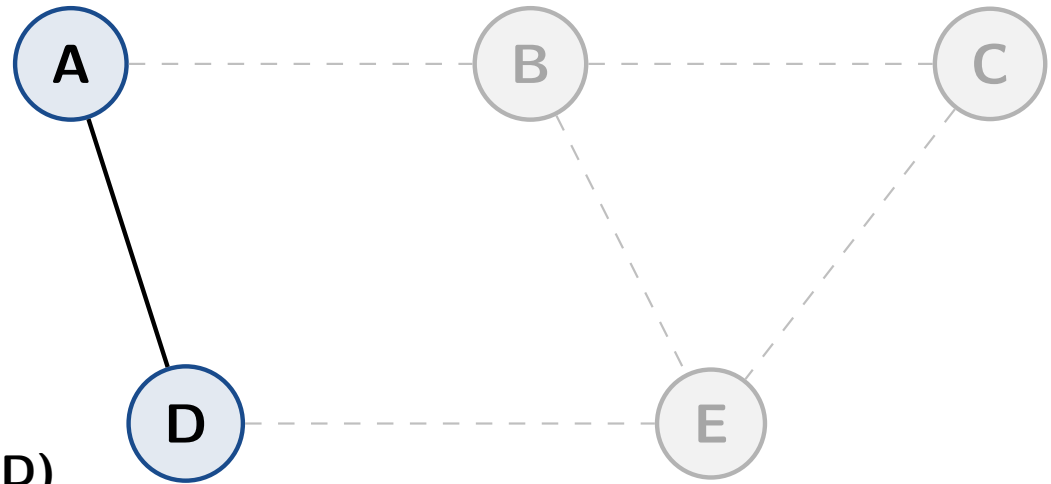


RIP table (D)
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : 2 rE

A et D se retrouvent isolés du reste du réseau (lien D–E perdu).

Perte de connexité

RIP table (A)
A : 0 local
D : 1 rD
B : 3 rD
C : 3 rD
E : 2 rD



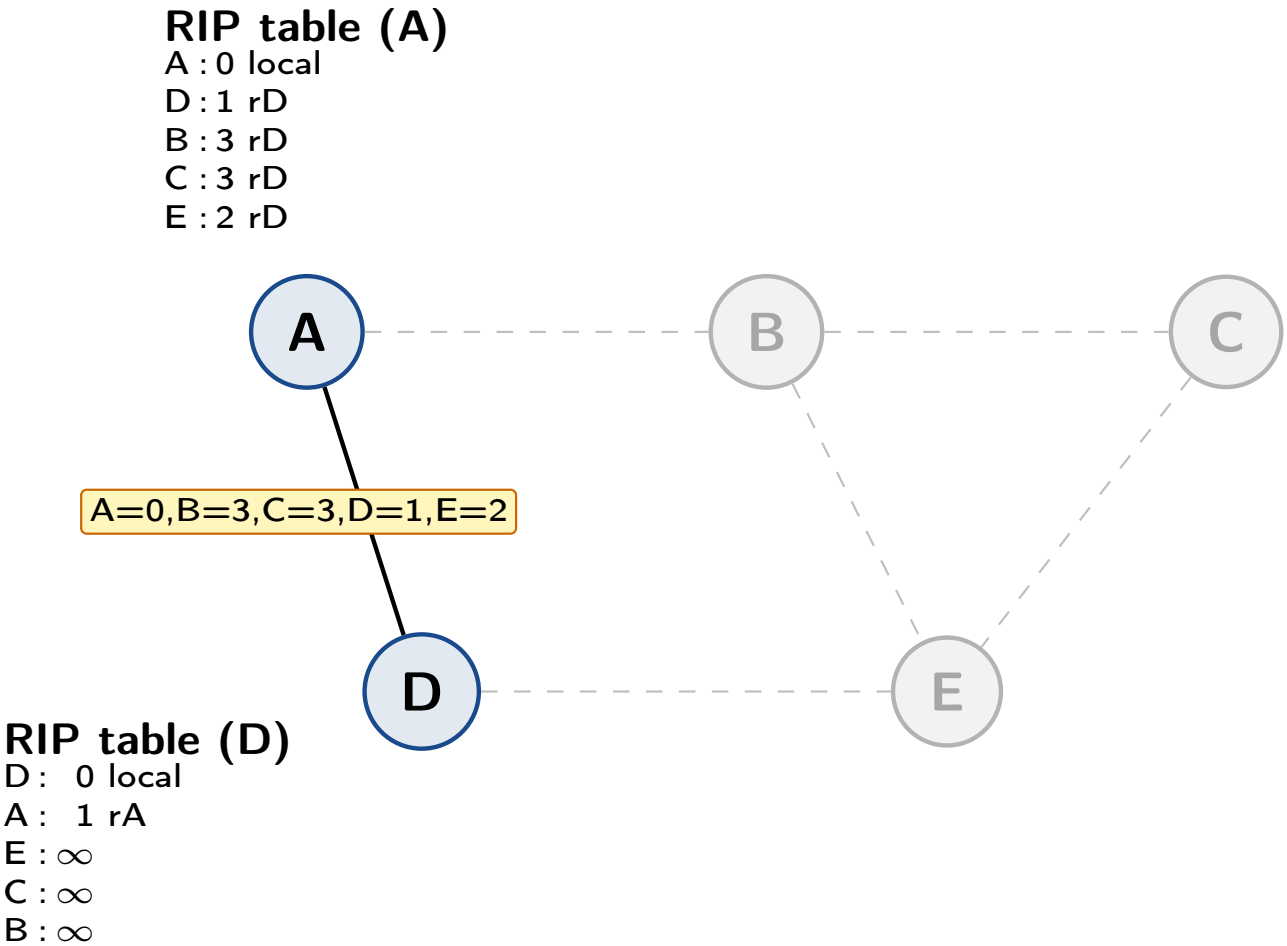
RIP table (D)
D : 0 local
A : 1 rA
E : ∞
C : ∞
B : ∞

D détecte la perte du lien D–E : E, C, B passent à ∞ chez D.

Perte de connexité

- Si D émet en premier, A met à jour sa table de routage et on est de nouveau dans un état stable
- Si A émet en premier . . .

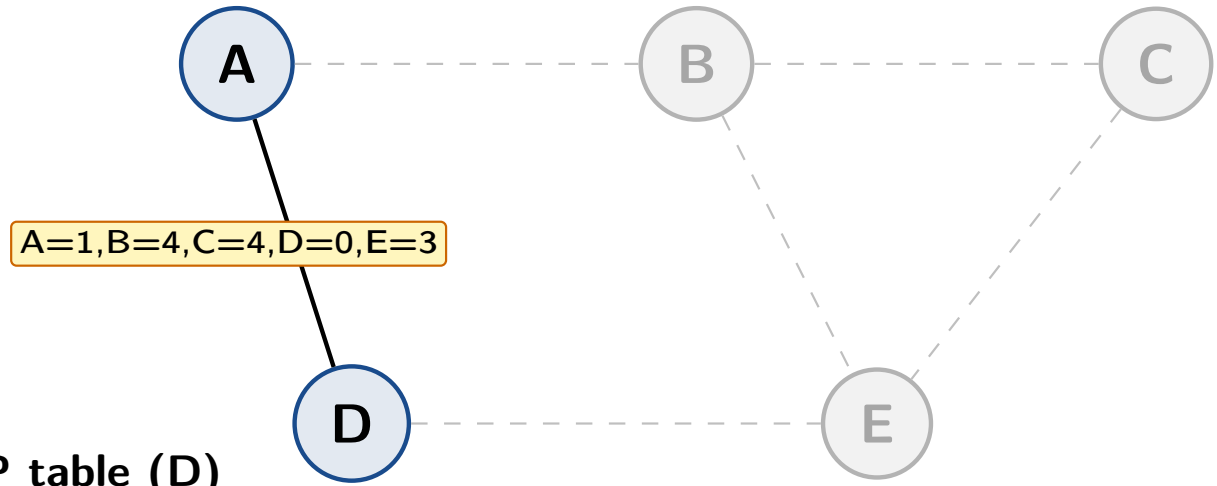
Perte de connexité



A émet en premier : il annonce à D des routes. . . qui passent par D !

Perte de connexité

RIP table (A)
A : 0 local
D : 1 rD
B : 3 rD
C : 3 rD
E : 2 rD

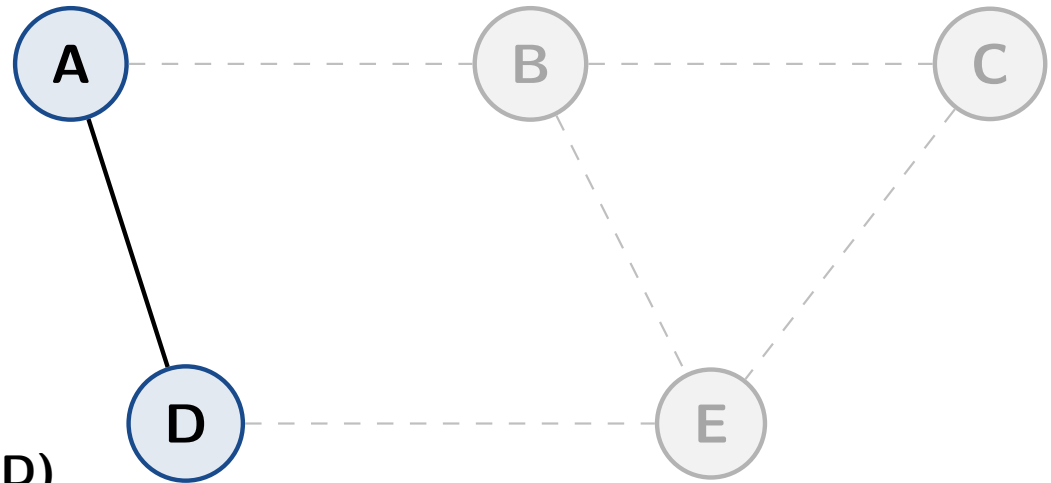


RIP table (D)
D : 0 local
A : 1 rA
E : 3 rA
C : 4 rA
B : 4 rA

D « apprend » E, C, B via rA (coûts 3, 4, 4) et émet à son tour.

Perte de connexité

RIP table (A)
A : 0 local
D : 1 rD
B : 5 rD
C : 5 rD
E : 4 rD



RIP table (D)
D : 0 local
A : 1 rA
E : 3 rA
C : 4 rA
B : 4 rA

A incrémente à son tour (5, 5, 4)... et ainsi de suite, sans fin.

Perte de connexité : compter à l'infini

Dans ce cas, les tables de A et D vont s'incrémenter alternativement sans fin

- Le problème vient du fait que A et D s'annoncent des routes qu'ils ont appris l'un de l'autre
- Tout dépend de l'ordre des mises à jour
- Pour remédier à ce problème, il faut empêcher A de provoquer la mise à jour de la table de D

Convergence lente : des solutions

Dans certains cas (e.g. perte de connexité), la convergence vers une nouvelle situation stable peut prendre beaucoup de temps :

- Hold down : on peut forcer un routeur qui détecte la perte d'un lien à *ignorer les mises à jour* pendant un certain temps (plus d'une période d'émission de vecteurs de distance)
- Triggered update : on peut forcer un routeur qui détecte la perte d'un lien à *envoyer immédiatement* une mise à jour à ses voisins
- Split horizon (horizon éclaté) : on peut forcer chaque routeur à *ne pas communiquer* à ses voisins les chemins qui passent par ceux-ci

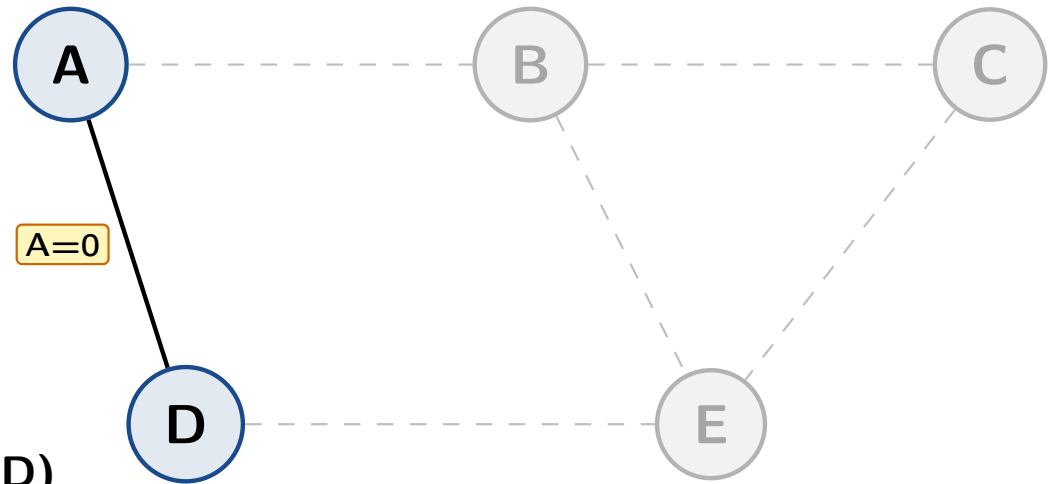
Émission périodique (version 2) : avec horizon éclaté

```
loop indefinitely {
    sleep(N seconds)
    for each link l {
        V=[]
        for each destination d in R[] {
            if (R[d].gw not on l) {
                V[d]=R[d].cost
            }
        }
        l.send(V)
    }
}
```

Le vecteur est maintenant différent sur chaque lien.

Avec horizon éclaté

RIP table (A)
A : 0 local
D : 1 rD
B : 3 rD
C : 3 rD
E : 2 rD

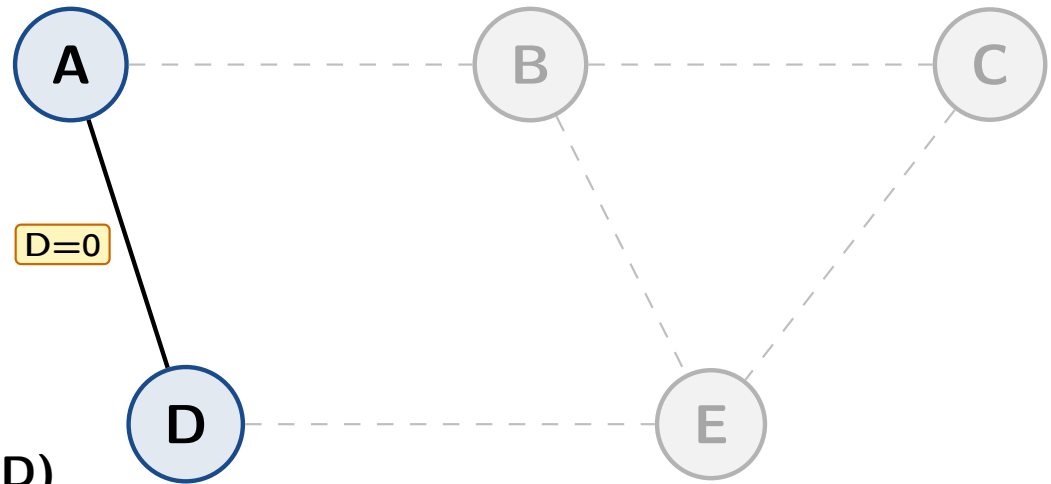


RIP table (D)
D : 0 local
A : 1 rA
E : ∞
C : ∞
B : ∞

A n'annonce à D que A=0 : les routes via rD ne sont pas annoncées sur ce lien.

Avec horizon éclaté

RIP table (A)
A : 0 local
D : 1 rD
B : 3 rD
C : 3 rD
E : 2 rD

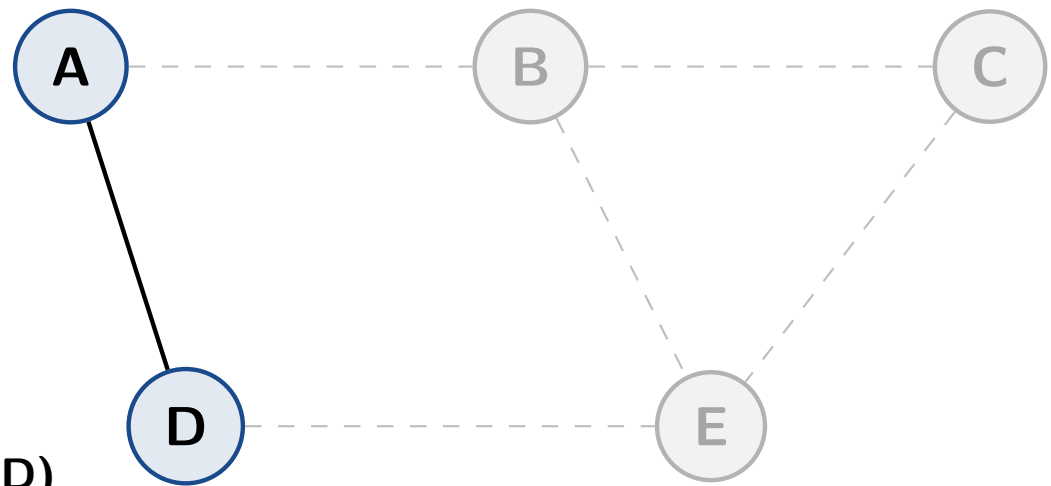


RIP table (D)
D : 0 local
A : 1 rA
E : ∞
C : ∞
B : ∞

De même, D n'annonce à A que D=0. Pas de fausse route apprise.

Avec horizon éclaté

RIP table (A)
A : 0 local
D : 1 rD
B : 3 rD ?
C : 3 rD ?
E : 2 rD ?



RIP table (D)
D : 0 local
A : 1 rA
E : ∞
C : ∞
B : ∞

Les entrées B, C, E de A ne sont plus confirmées : elles expireront (délai).

Horizon éclaté avec empoisonnement

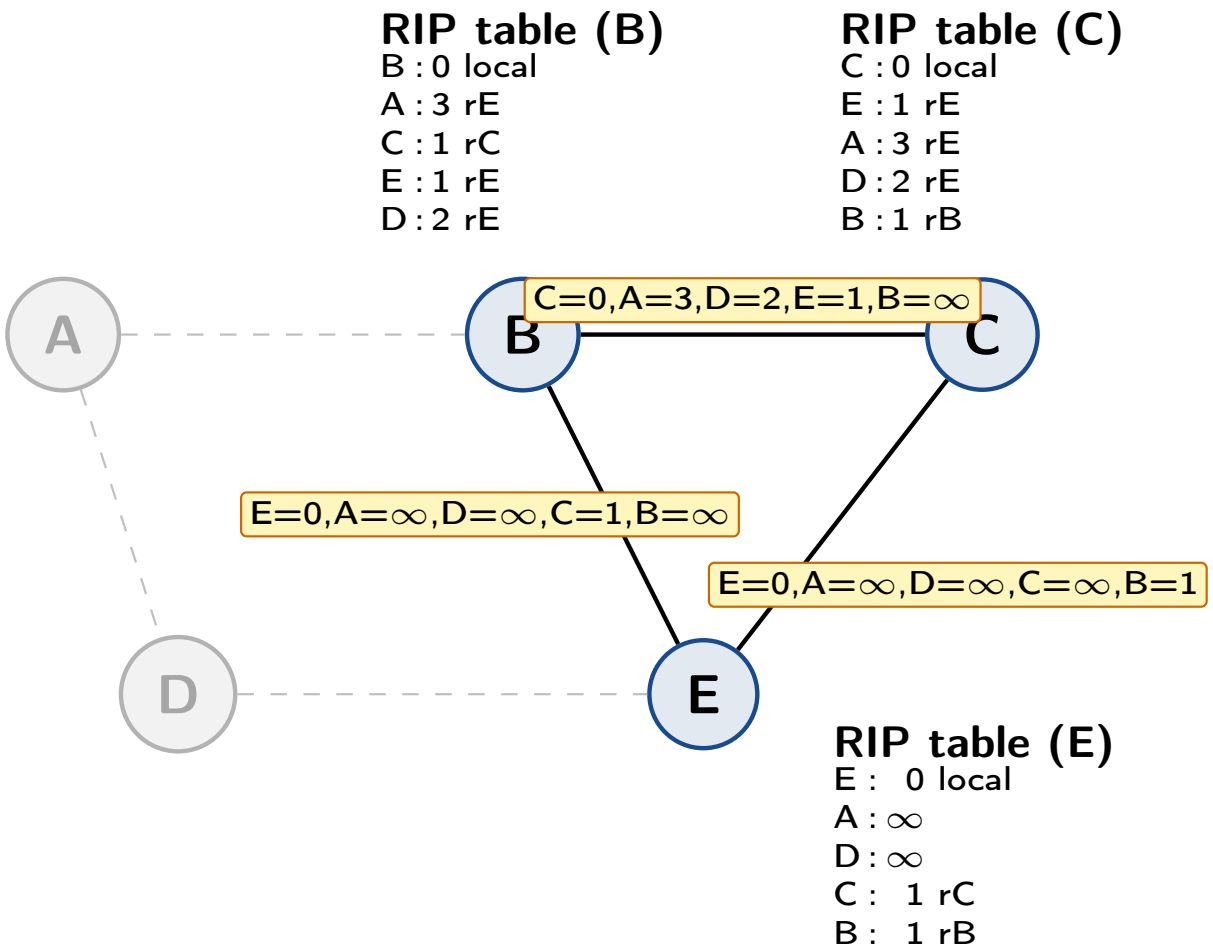
Pour accélérer le processus, au lieu de *ne pas communiquer* à ses voisins les chemins perdus, on leur annonce un coût ∞.

- Par exemple si 2 routeurs pointent l'un vers l'autre, l'empoisonnement provoque la mise à jour immédiate
- Sans empoisonnement, il faut attendre un délai au bout duquel le réseau est déclaré non accessible.

Émission périodique (version 3)

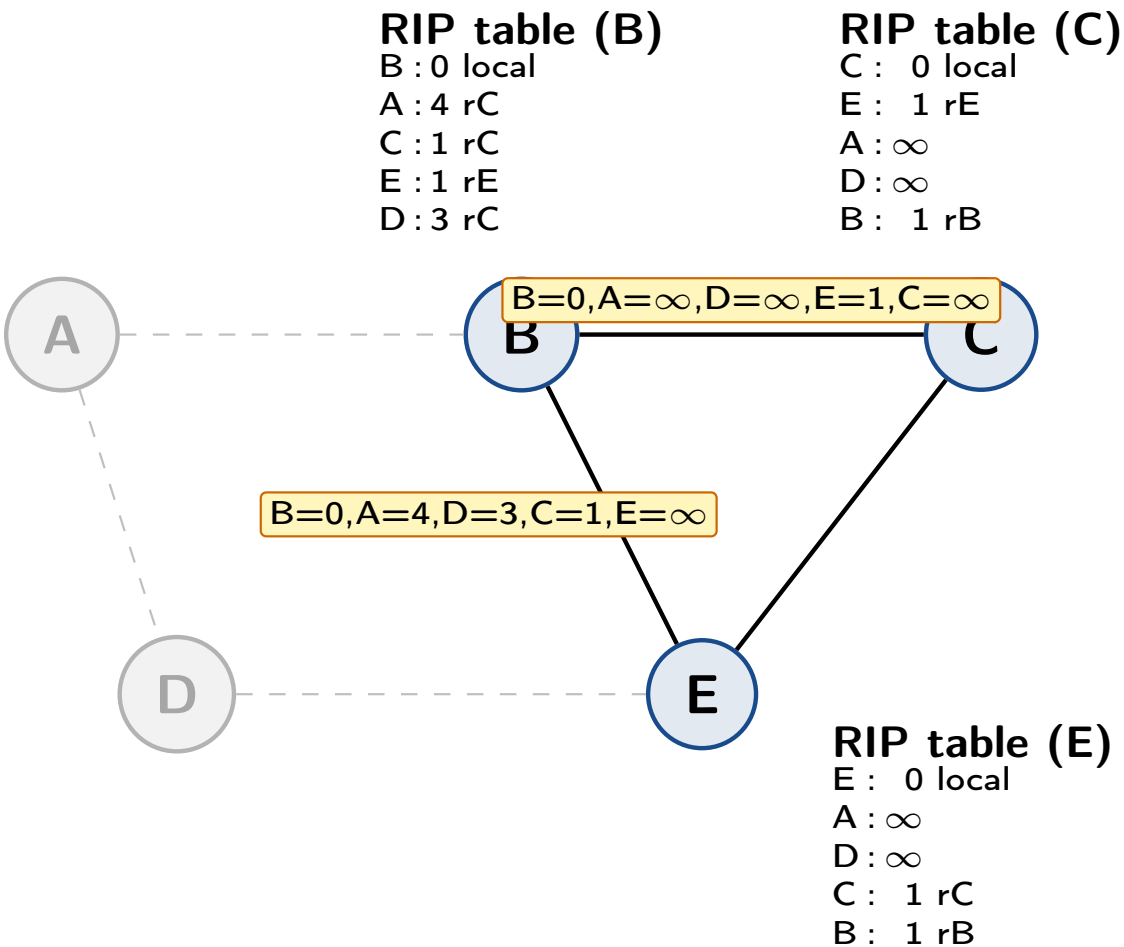
```
loop indefinitely {
    sleep(N seconds)
    for each link l {
        V=[]
        for each destination d in R[] {
            if (R[d].gw not on l) {
                V[d]=R[d].cost
            } else {
                V[d]=infinity
            }
        }
        l.send(V)
    }
}
```

Boucle de longueur > 2



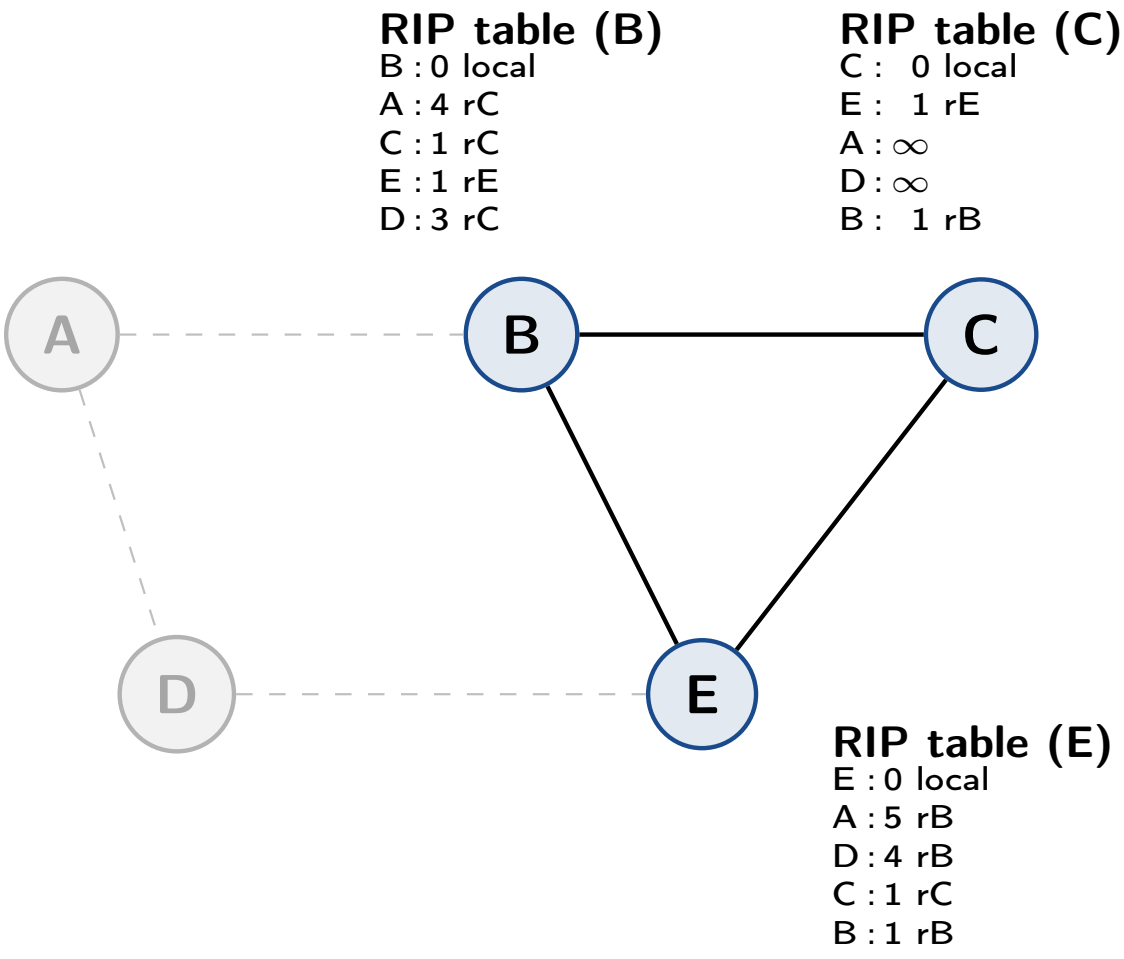
Problème : rE et rC émettent *en même temps* (avec empoisonnement).

Boucle de longueur > 2



B reçoit et met à jour en fonction de rE *puis* de rC : A=4, D=3 via rC. B émet.

Boucle de longueur > 2



E apprend A=5, D=4 via rB : **nouveau compte à l'infini**, malgré l'empoisonnement.

Solutions et problèmes

- Toutes les solutions possibles (Hold down, Triggered updates, Split horizon,...) sont partielles
- En ajoutant des messages (e.g. Triggered update), il y a risque de saturer le réseau (les messages sont en broadcast)
- En ajoutant des données (Split horizon with poisoning : les routeurs annoncent ∞ au lieu de ne rien annoncer sur le lien emprunté par un chemin), le volume de données qui circule augmente

Limites des vecteurs de distance

- Convergence lente
- Messages volumineux
- Passage à l'échelle compliqué

Solutions et problèmes

- Toutes les solutions possibles (Hold down, Triggered updates, Split horizon,...) sont partielles
- En ajoutant des messages (e.g. Triggered update), il y a risque de saturer le réseau (les messages sont en broadcast)
- En ajoutant des données (Split horizon with poisoning : les routeurs annoncent ∞ au lieu de ne rien annoncer sur le lien emprunté par un chemin), le volume de données qui circule augmente

Limites des vecteurs de distance

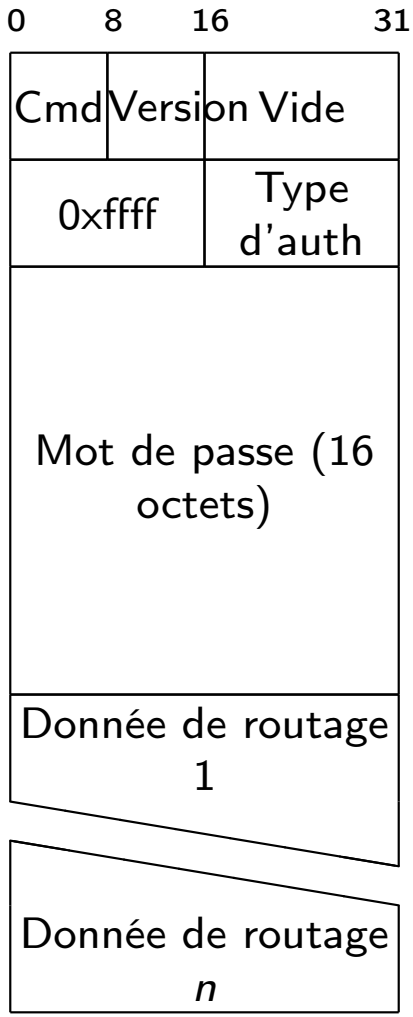
- Convergence lente
- Messages volumineux
- Passage à l'échelle compliqué

RIP : Routing Information Protocol

Protocole à base de vecteurs de distance (RFC 2453) :

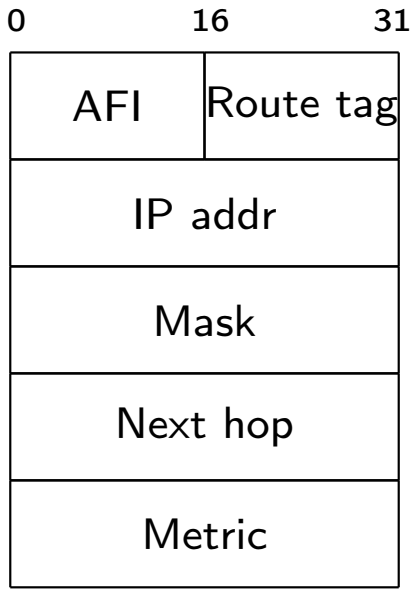
- $\infty = 16$
- tous les coûts sont de 1 (métrique = nombre de sauts)
- émission de message toutes les 30 s (Hold down de 60 s)
- datagrammes UDP, port 520, envoyés en multicast à tous les routeurs RIP.

Datagramme RIP



- **Commande** : 1 pour une requête, 2 pour une réponse.
- **Version** : 1 ancienne, 2 actuelle.
- Authentification optionnelle : léger gain en sécurité si mot de passe.
- 20 octets par entrée de donnée de routage.

Datagramme RIP : donnée de routage



- **AFI** (Address Family Identifier) : 2 pour IPv4.
- **Route tag** : marqueur peu utilisé, pour des routes apprises hors RIP.
- **IP addr / Mask** : réseau destination.
- **Next hop** : en général 0.0.0.0 ; peu utilisé, normalement le prochain saut est l'émetteur du message, mais dans certains cas (e.g. interconnexions RIP/OSPF), on peut gagner un saut.
- **Metric** : $\infty = 16$.

Pause exercice — Wooclap

RIP



Code : **DJKFQEB**

Connectez-vous à `wooclap.com` avec le code de la séance.

États de liens (LS)

1. Le problème du routage
2. Vecteurs de distance (DV)
3. États de liens (LS)
4. Routage externe

Dijkstra

Seconde option : distribué à connaissance globale

- Algorithme de Dijkstra pour calculer les plus courts chemins dans le graphe
- Chaque routeur connaît le coût de ses liens (calcul ou hypothèse)
- Chaque routeur transmet à tous les autres (par inondation) la topologie locale
- À la réception, le routeur reconstruit la carte du réseau et calcule un arbre des plus courts chemins

Découvrir la topologie locale

- Chaque routeur émet un message HELLO en local pour découvrir ses voisins
- Message envoyé en multicast sur chaque interface participant au routage
- Le message contient l'adresse de l'émetteur pour la réponse
- Message envoyé périodiquement pour détecter les changements de topologie

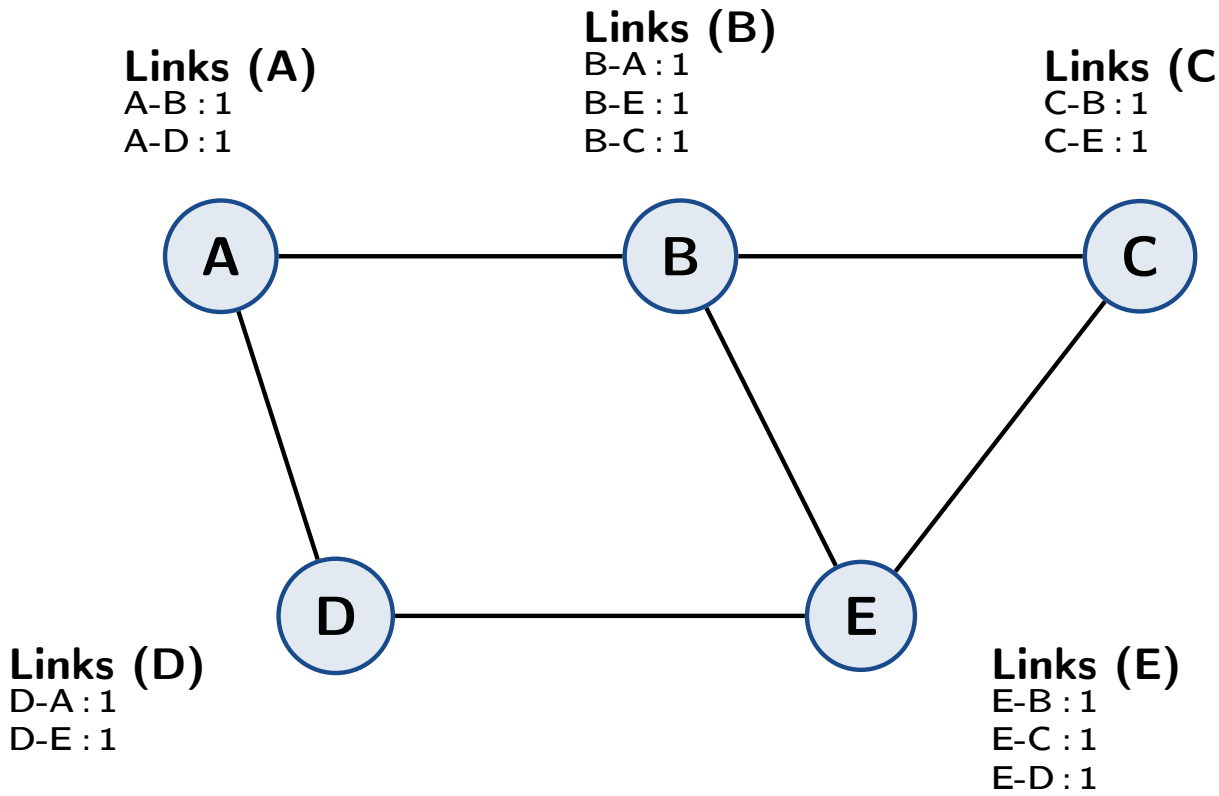
Le coût associé à un lien est choisi selon le contexte en fonction de différents critères : coût unité, proportionnel au débit ou à la latence, inversement proportionnel à la capacité...

Communiquer la topologie locale

- Chaque routeur émet un message d'états de liens à tous les routeurs voisins
- Message envoyé en multicast sur chaque interface participant au routage
- Le message contient la liste des liens locaux :
  - vers des routeurs participant au routage LS
  - vers des réseaux sur lesquels on ne fait pas de routage LS
- Message envoyé périodiquement et aux changements de topologie

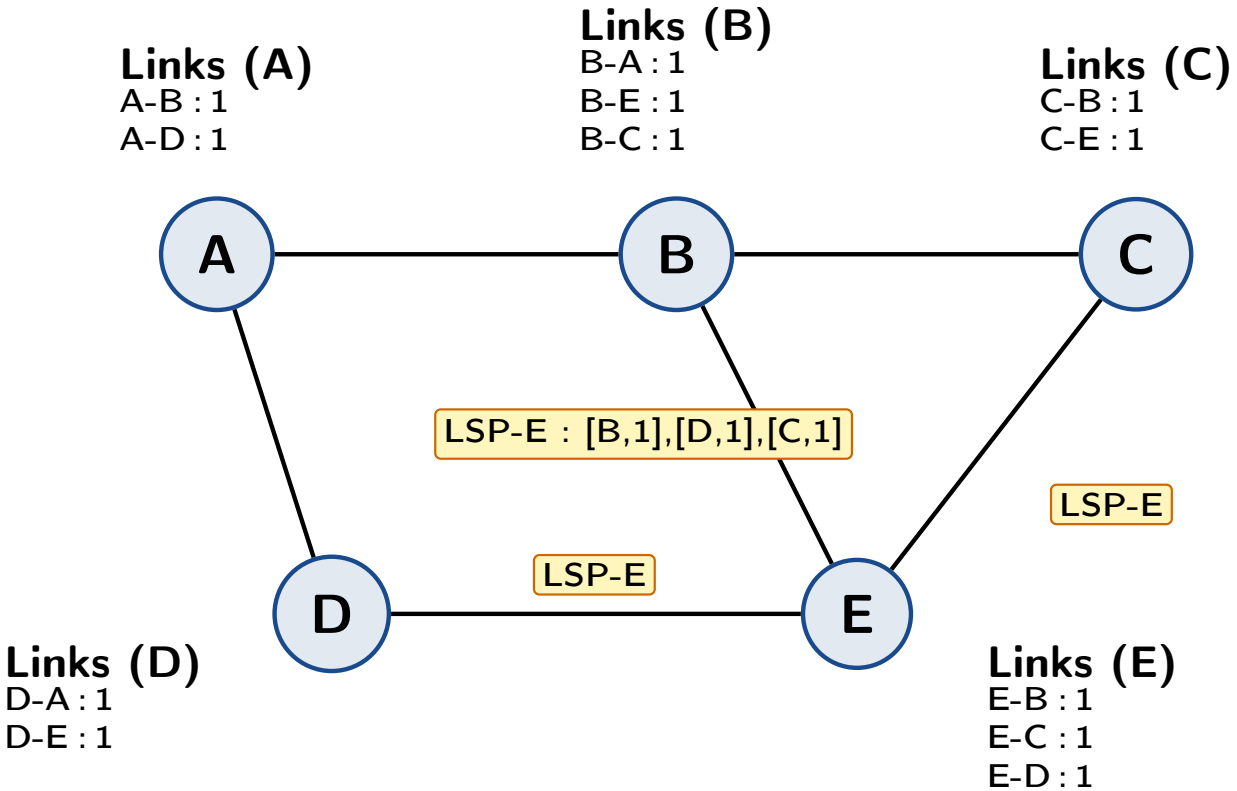
Message envoyé par inondation des routeurs

Envoi (coûts égaux à 1)



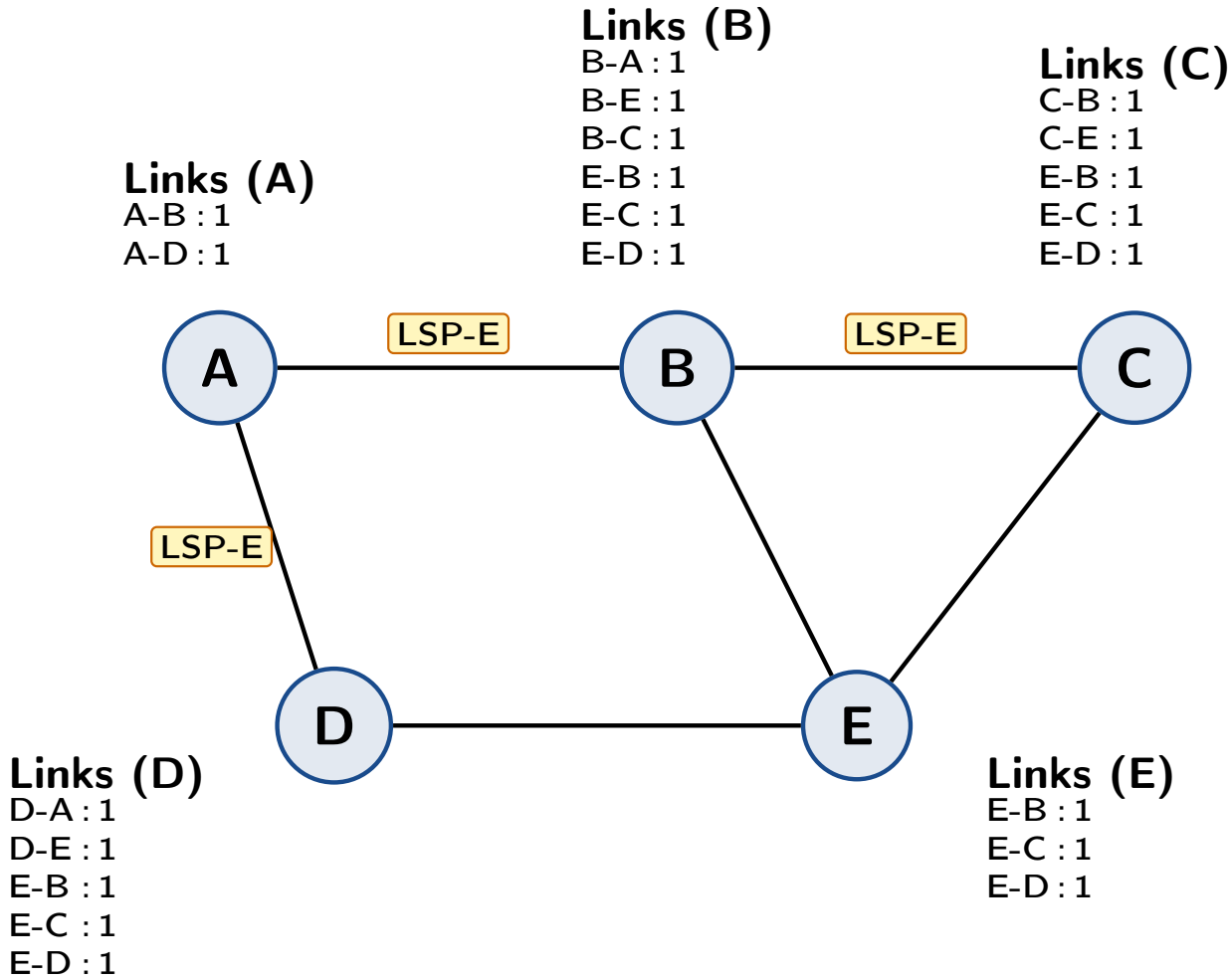
Chaque routeur connaît ses liens locaux (découverts par HELLO).

Envoi (coûts égaux à 1)



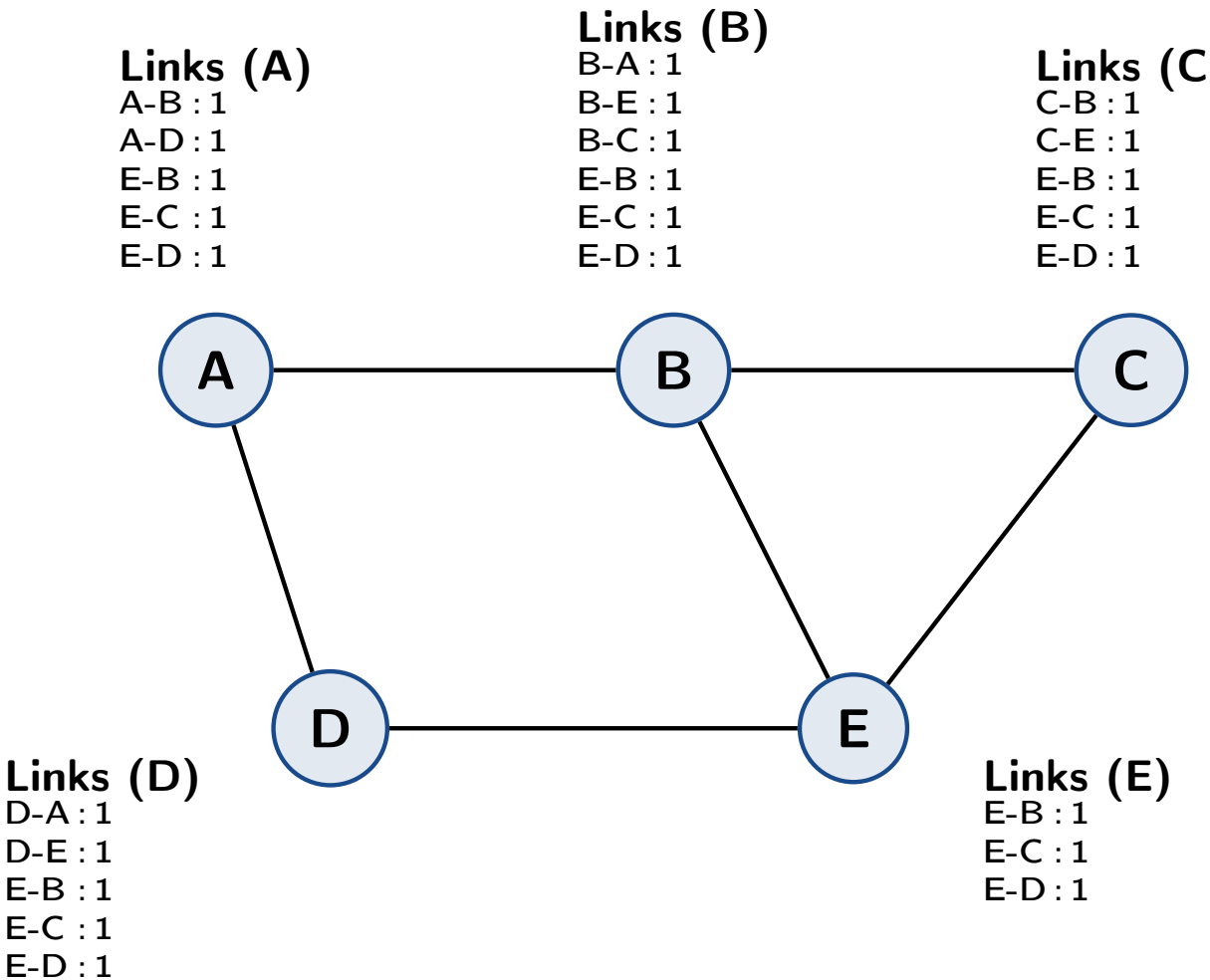
E émet son LSP (liste de ses liens) vers ses voisins B, C, D.

Envoi (coûts égaux à 1)



B, C, D enregistrent les liens de E et retransmettent (sauf vers E).

Envoi (coûts égaux à 1)

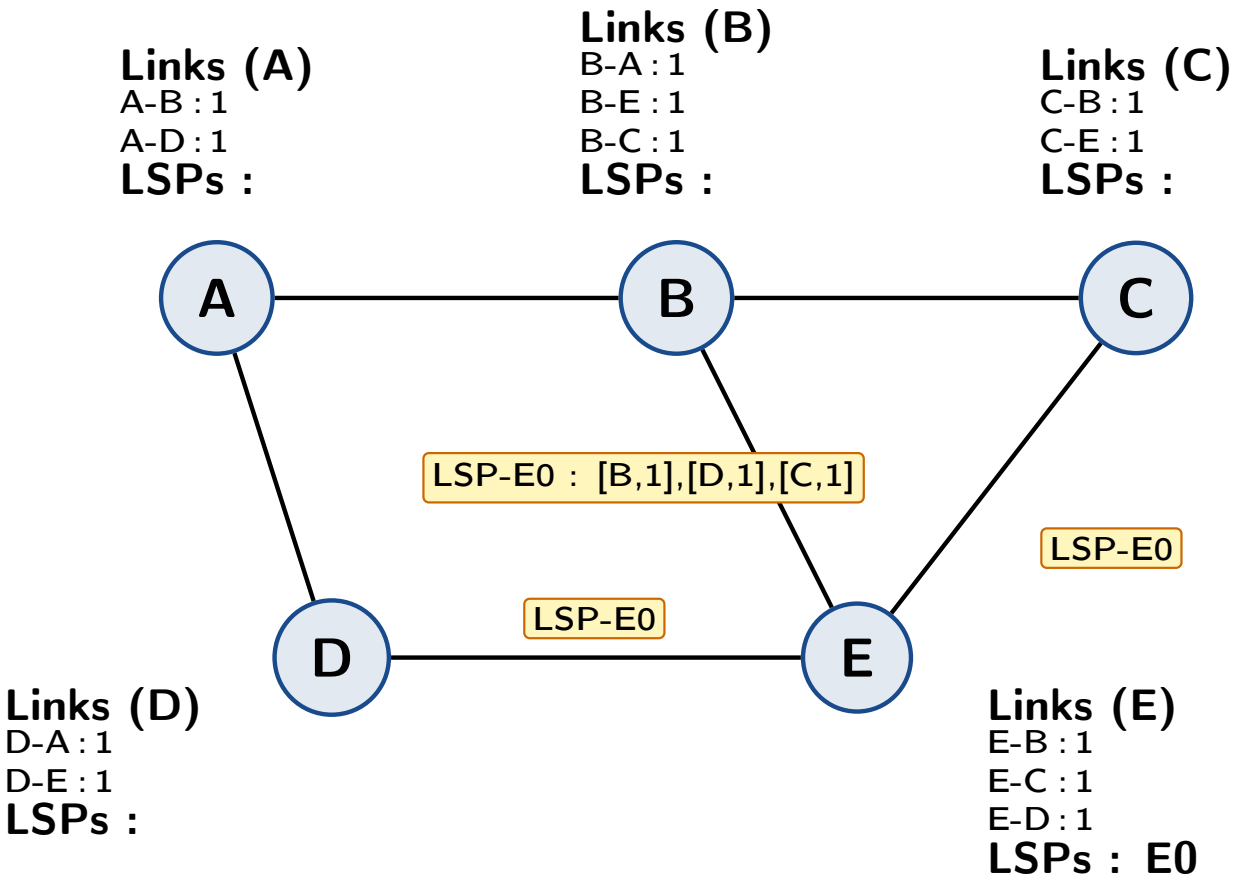


A reçoit le LSP de E (via B et via D) : tout le réseau connaît les liens de E.

Envois de LSP

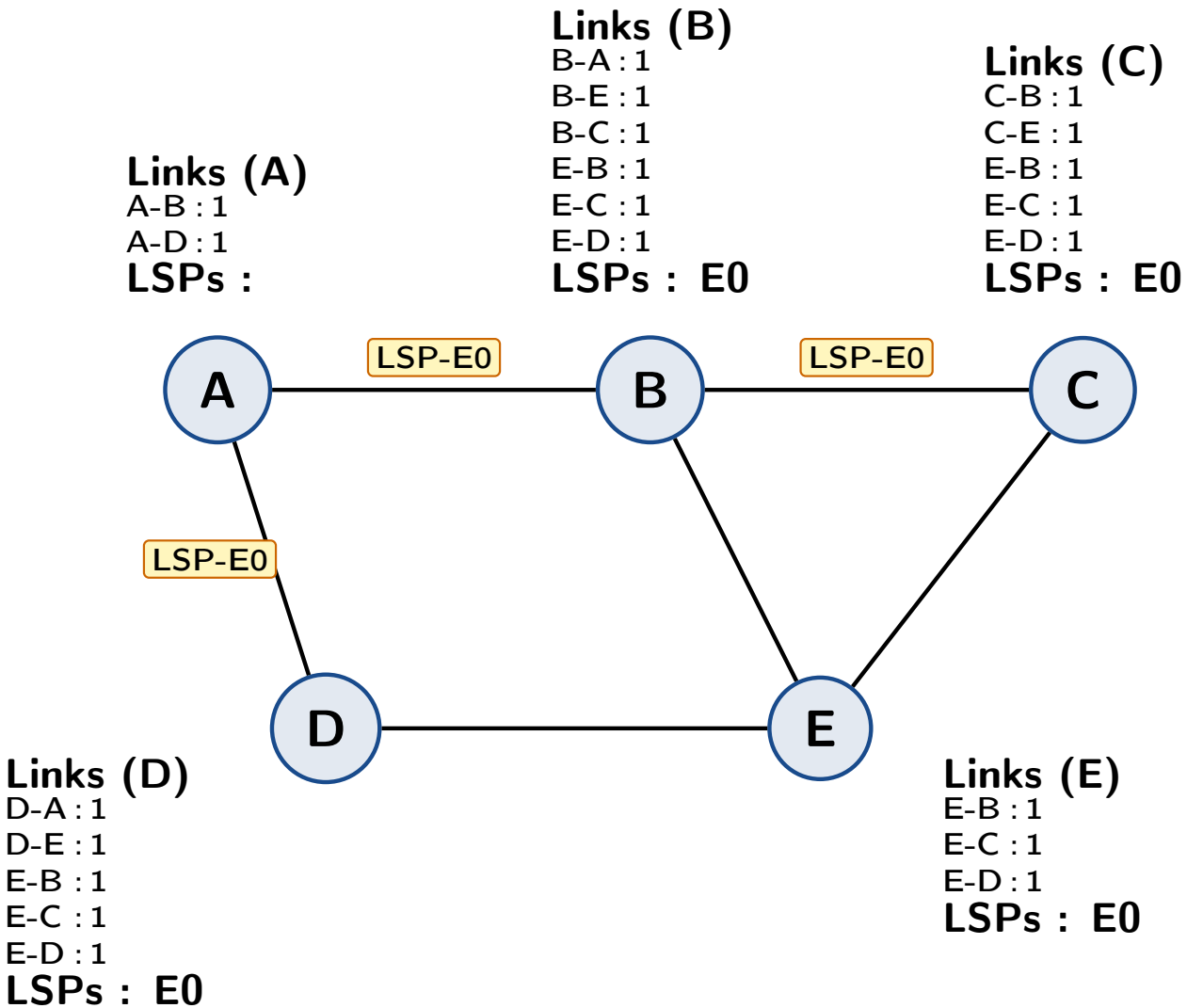
- Chaque routeur transmet les LSP à tous ses voisins excepté celui par qui il a reçu le paquet
- Pour éviter les boucles, il ne faut pas transmettre plusieurs fois le même LSP
- Identification par adresse du routeur émetteur et numéro de série incrémenté par l'émetteur à chaque envoi
- Les routeurs mémorisent le LSP (émetteur, numéro et liste de liens) le plus récent de chaque émetteur

Envoi avec numéros de série



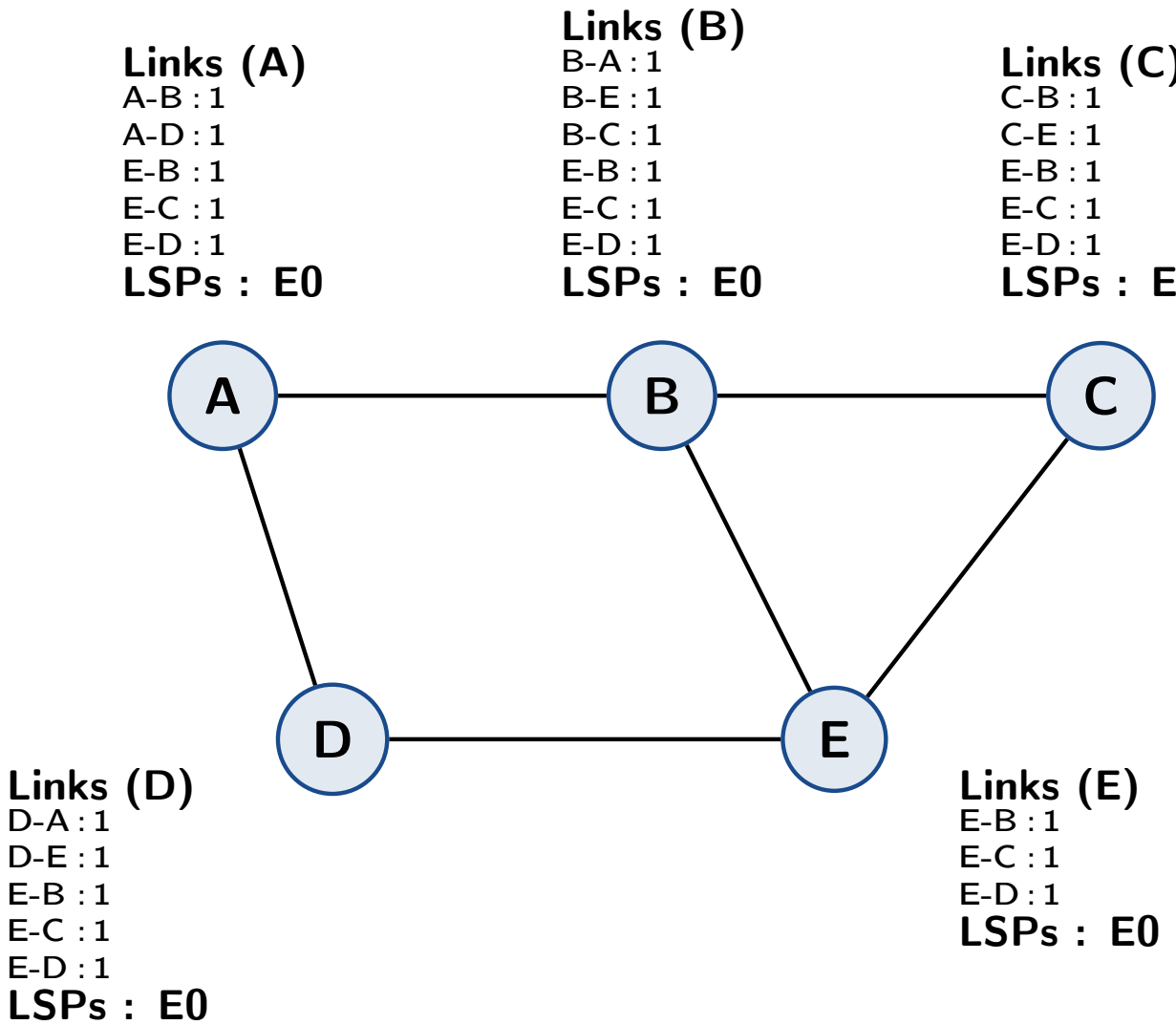
E émet le LSP numéroté E0 ; chaque routeur mémorise les LSP déjà vus.

Envoi avec numéros de série



B, C, D notent « E0 » et retransmettent : A le reçoit (deux fois), B et C aussi.

Envoi avec numéros de série



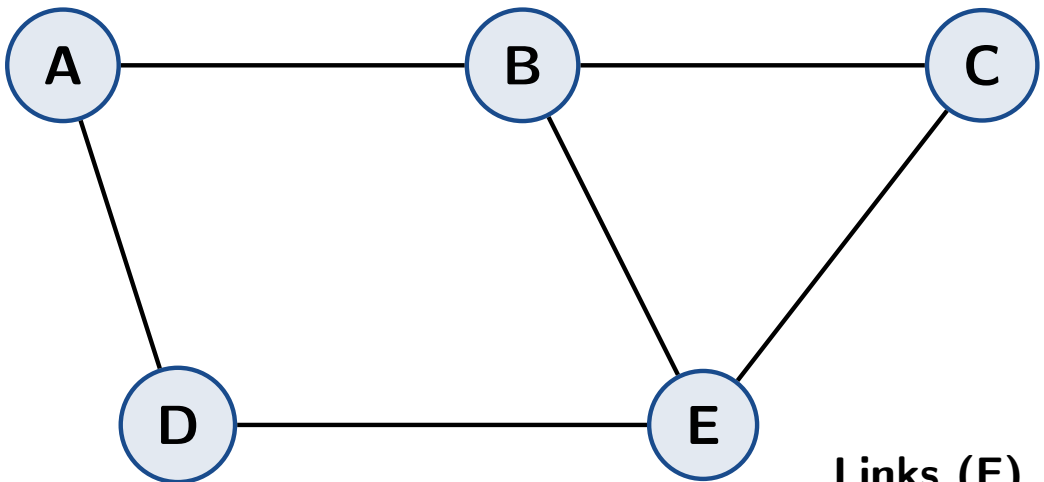
A, B, C détectent les doublons grâce au numéro E0 : pas de retransmission.

Situation stable

Links (A)
A-B : 1
A-D : 1
B-C : 1
B-E : 1
C-E : 1
D-E : 1
LSPs : E0,A0,B0,C0,D0

Links (B)
A-B : 1
A-D : 1
B-C : 1
B-E : 1
C-E : 1
D-E : 1
LSPs : E0,A0,B0,C0,D0

Links (C)
A-B : 1
A-D : 1
B-C : 1
B-E : 1
C-E : 1
D-E : 1
LSPs : E0,A0,B0,C0,D0



Links (D)
A-B : 1
A-D : 1
B-C : 1
B-E : 1
C-E : 1
D-E : 1
LSPs : E0,A0,B0,C0,D0

Links (E)
A-B : 1
A-D : 1
B-C : 1
B-E : 1
C-E : 1
D-E : 1
LSPs : E0,A0,B0,C0,D0

Après l'inondation des 5 LSP (A0 à E0), tous les routeurs connaissent tous les liens.

Calculer les chemins

Une fois la base de données des LSP constituée :

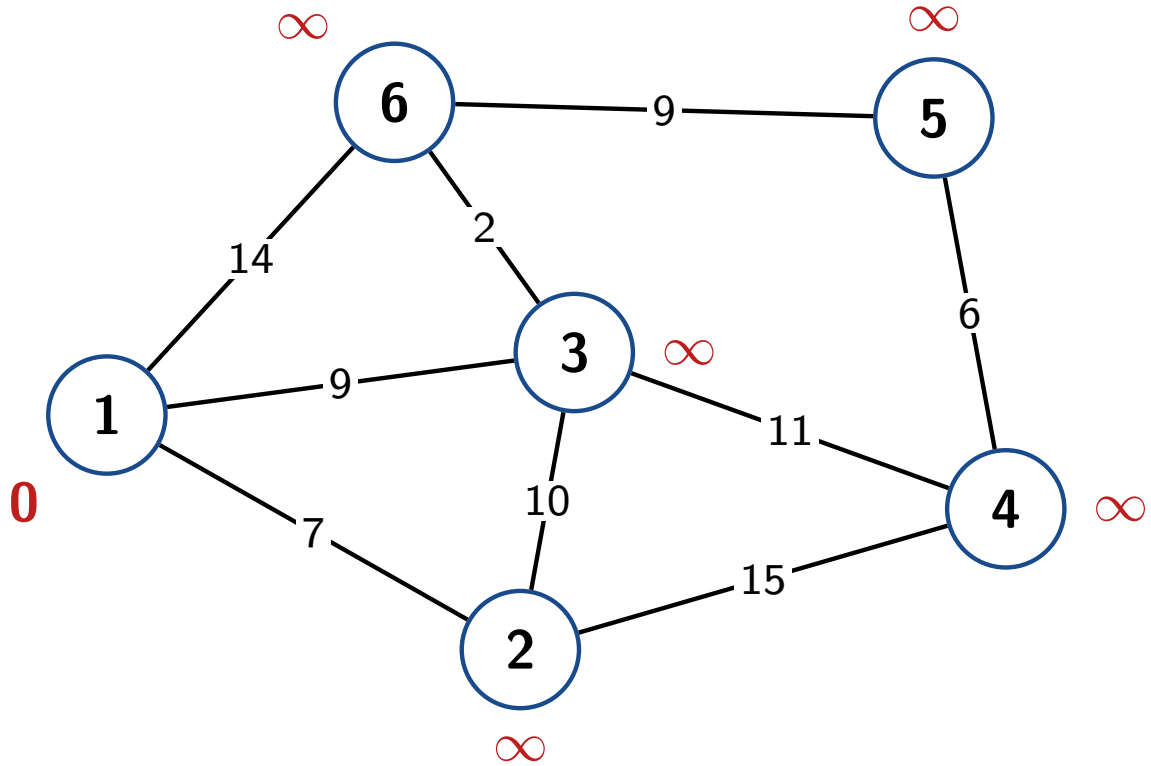
- chaque routeur peut reconstruire la topologie entière
- puis calculer un arbre des plus courts chemins
- algorithme de Dijkstra

Algorithme de Dijkstra

Un algorithme en $O(|E| + |V| \log |V|)$

```
for each vertex v in Graph {
    dist[v] = infinity
    previous[v] = undefined
}
dist[source] = 0
Q = the set of all vertices
while Q is not empty {
    u = vertex in Q with smallest distance in dist[]
    remove u from Q
    for each neighbor v of u {
        alt = dist[u] + dist_between(u, v)
        if alt < dist[v] {
            dist[v] = alt
            previous[v] = u
        }
    }
}
```

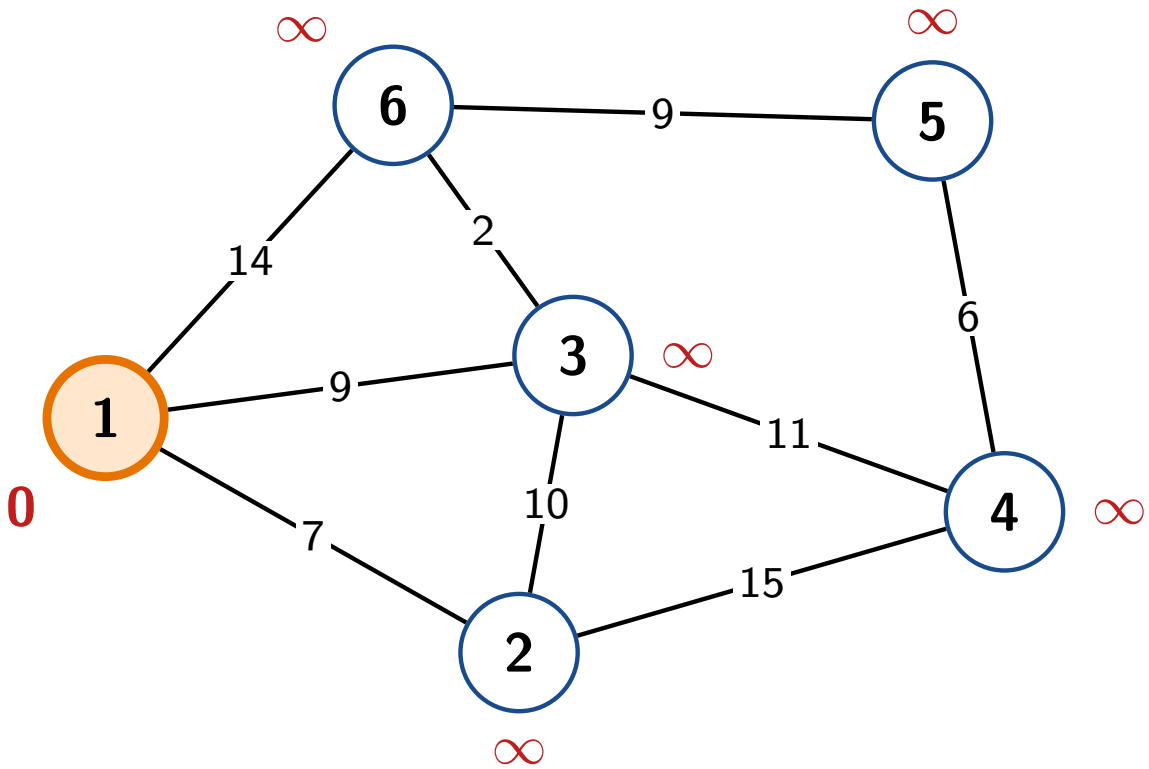
Dijkstra : exemple (source = 1)



$Q = \{1, 2, 3, 4, 5, 6\}$

Initialisation : dist[1]=0, toutes les autres distances à ∞.

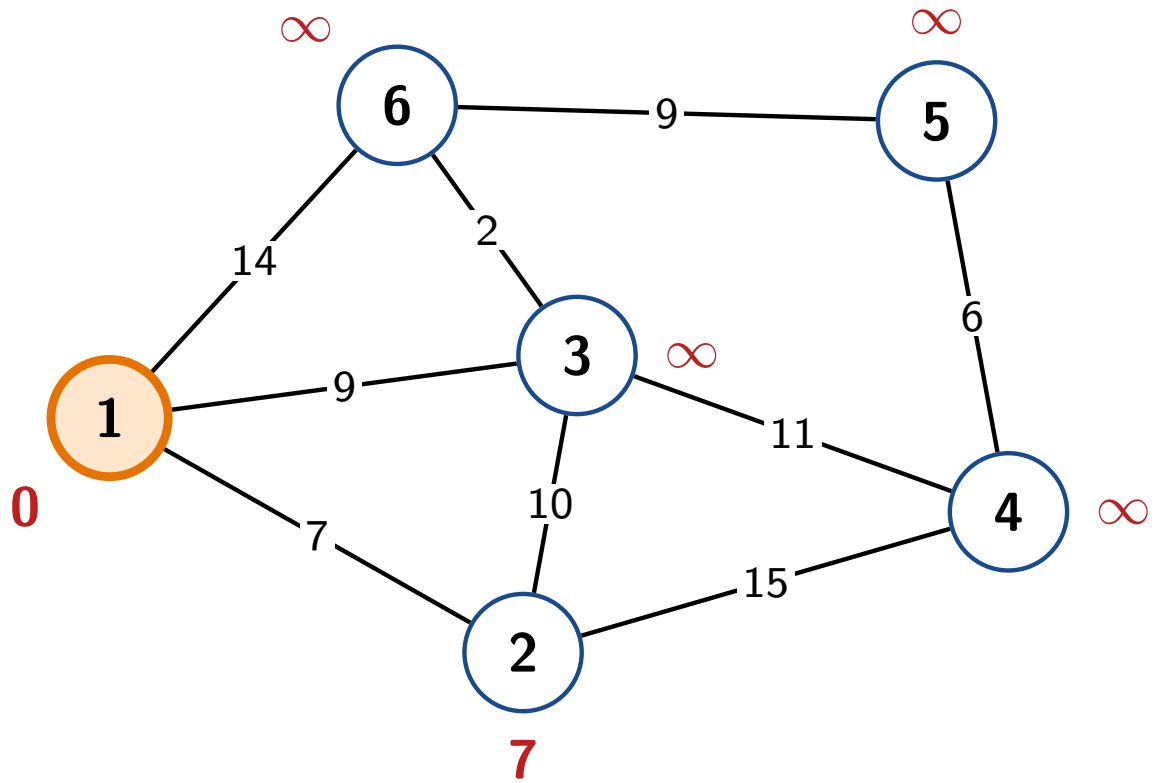
Dijkstra : exemple (source = 1)



$Q = \{2, 3, 4, 5, 6\}$

On extrait le sommet de plus petite distance : 1 (dist 0).

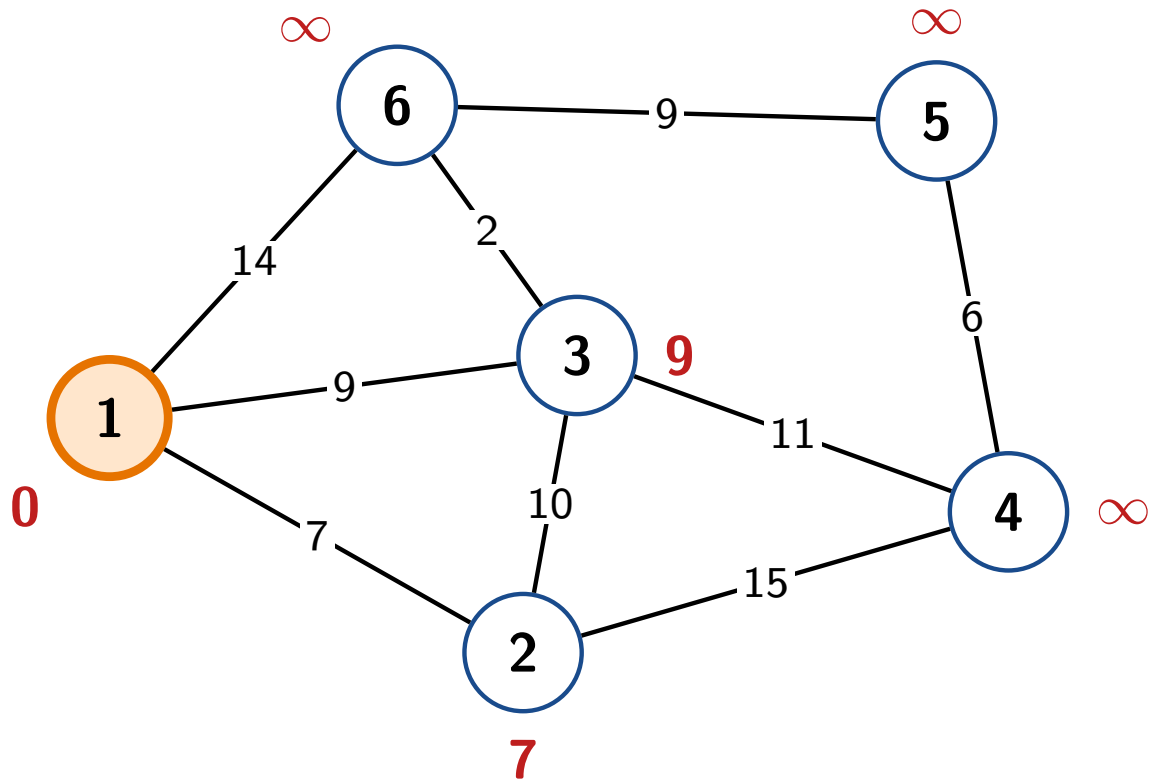
Dijkstra : exemple (source = 1)



$Q = \{2, 3, 4, 5, 6\}$

Relaxation du voisin 2 : $0 + 7 = 7$.

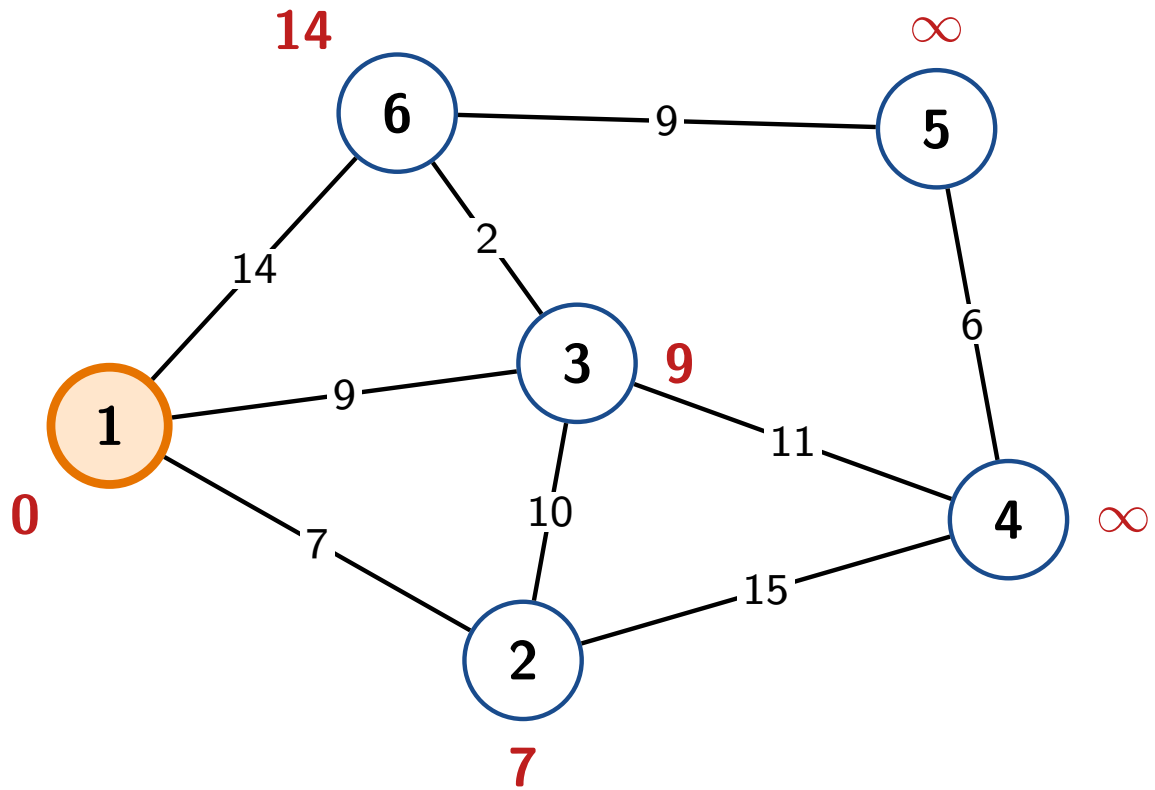
Dijkstra : exemple (source = 1)



$Q = \{2, 3, 4, 5, 6\}$

Relaxation du voisin 3 : $0 + 9 = 9$.

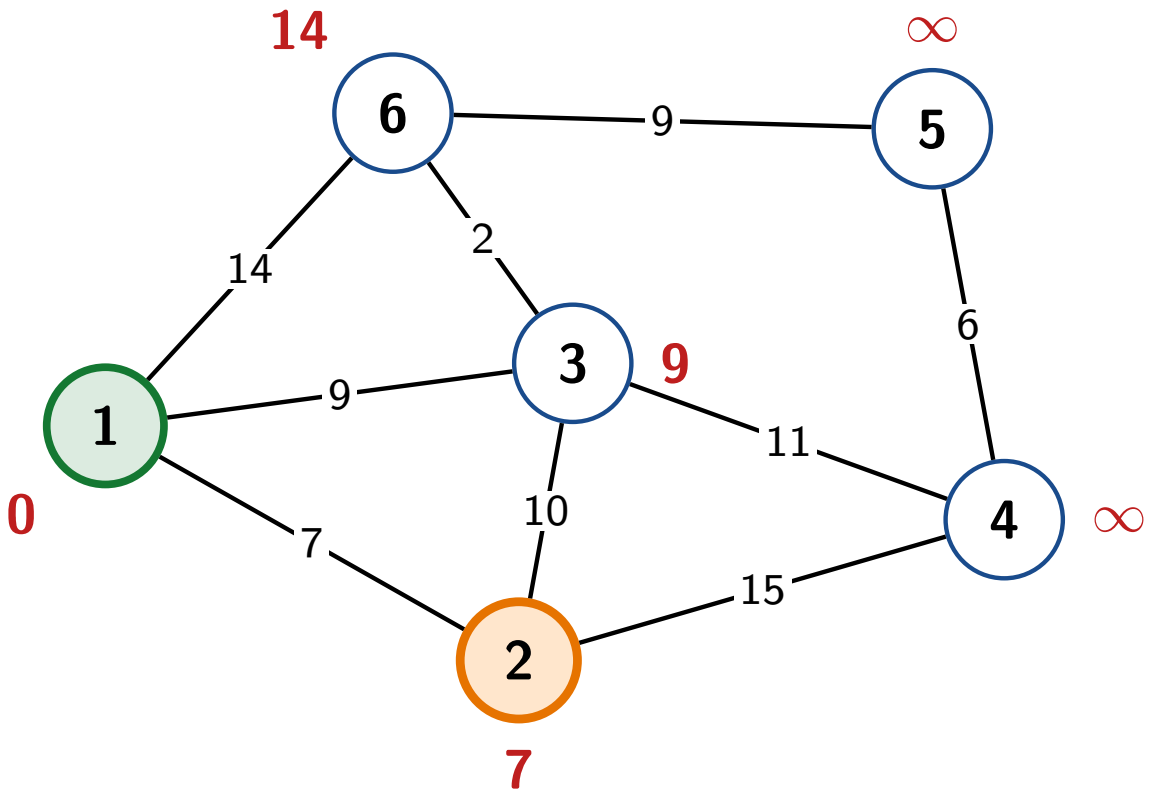
Dijkstra : exemple (source = 1)



$Q = \{2, 3, 4, 5, 6\}$

Relaxation du voisin 6 : $0 + 14 = 14$.

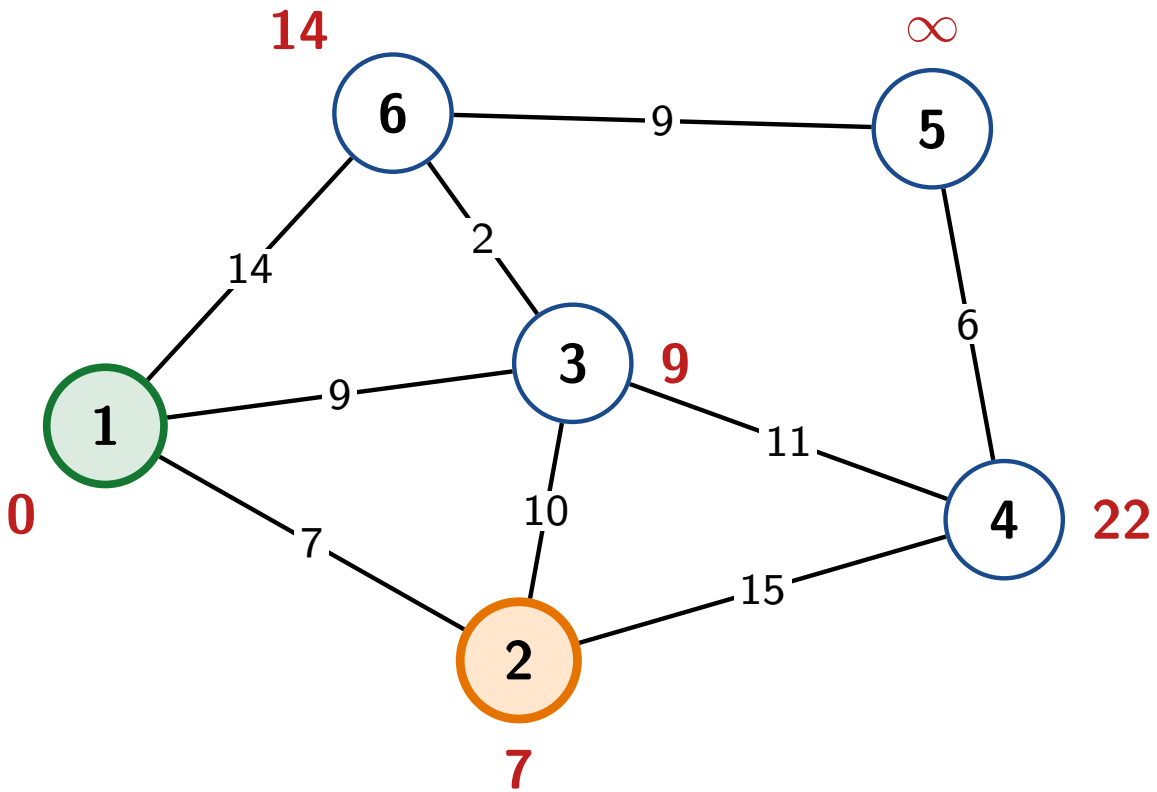
Dijkstra : exemple (source = 1)



$Q = \{3, 4, 5, 6\}$

On extrait 2 (dist 7). Voisin 3 : $7 + 10 = 17 > 9$, pas de changement.

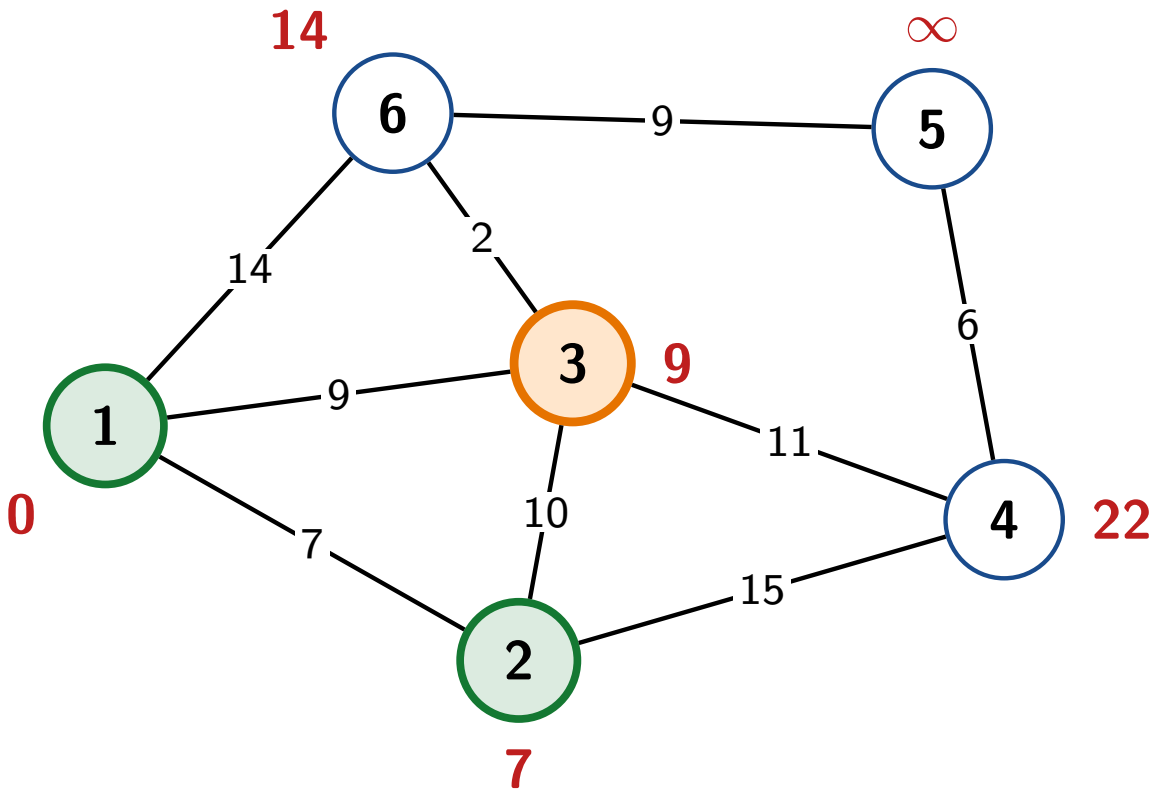
Dijkstra : exemple (source = 1)



$Q = \{3, 4, 5, 6\}$

Relaxation du voisin 4 : $7 + 15 = 22$.

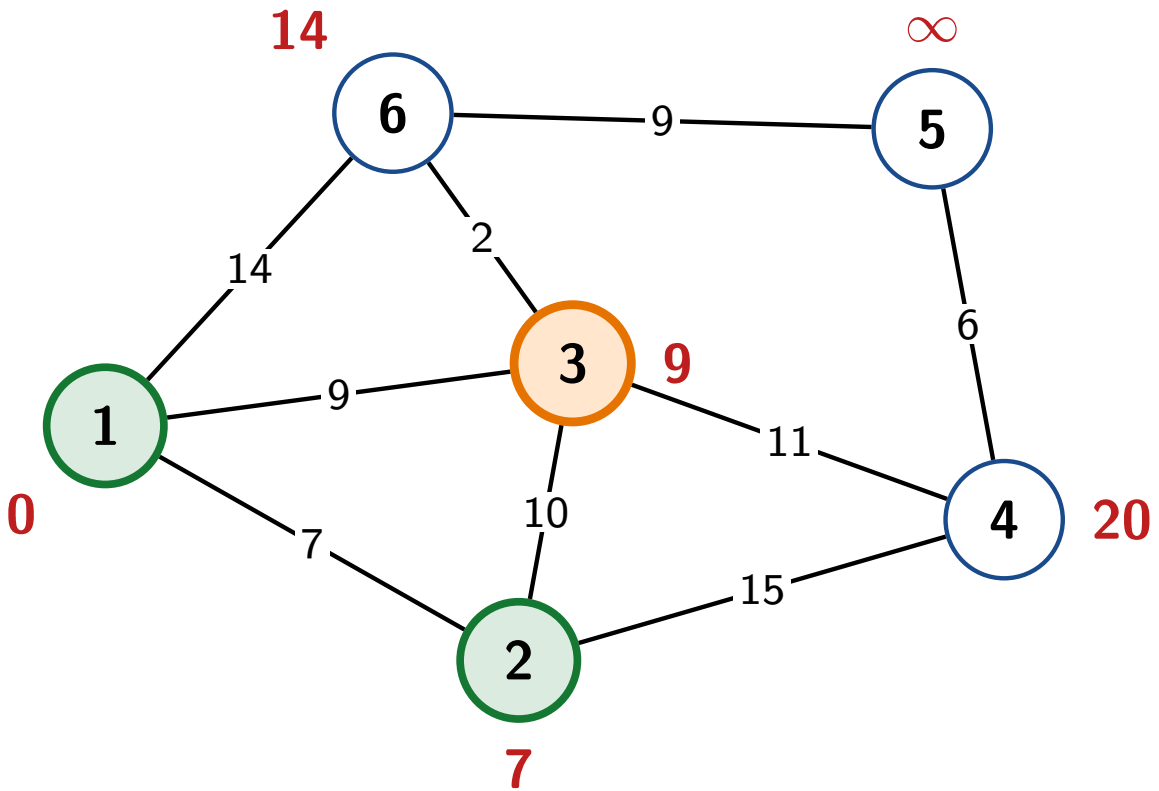
Dijkstra : exemple (source = 1)



$Q = \{4, 5, 6\}$

On extrait 3 (dist 9).

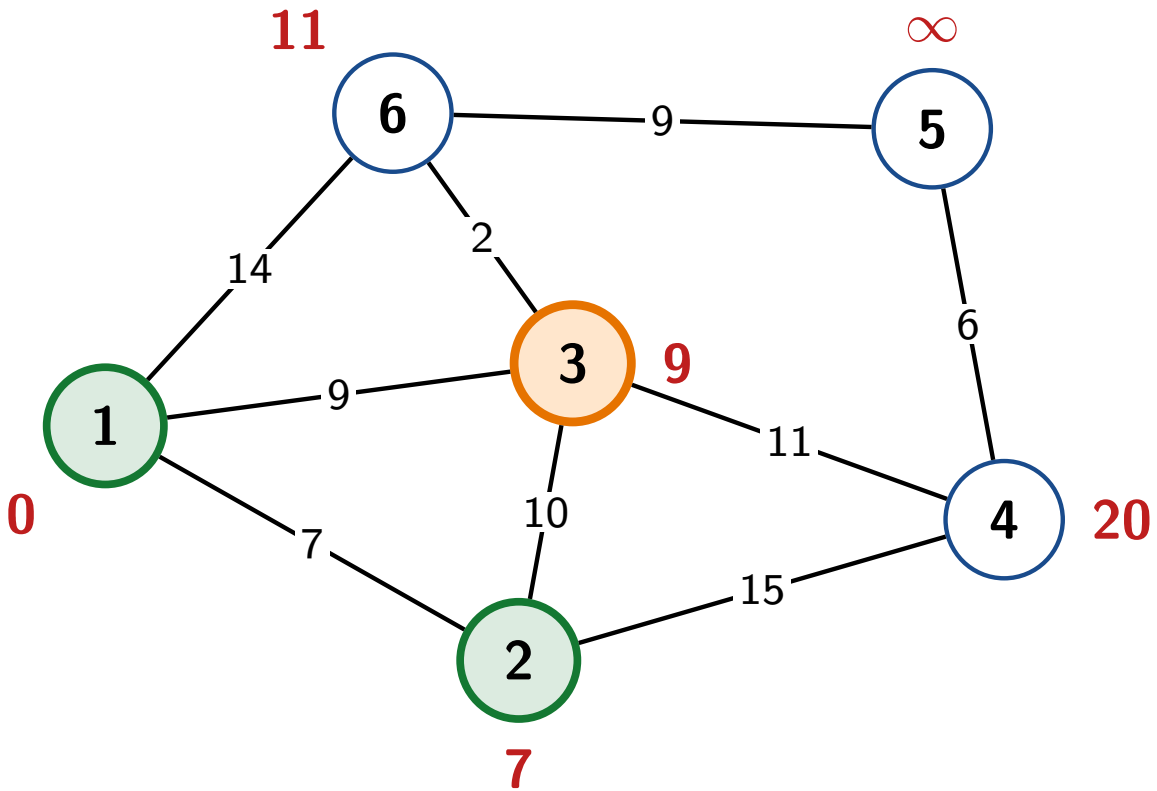
Dijkstra : exemple (source = 1)



$Q = \{4, 5, 6\}$

Relaxation du voisin 4 : $9 + 11 = 20 < 22$, mise à jour.

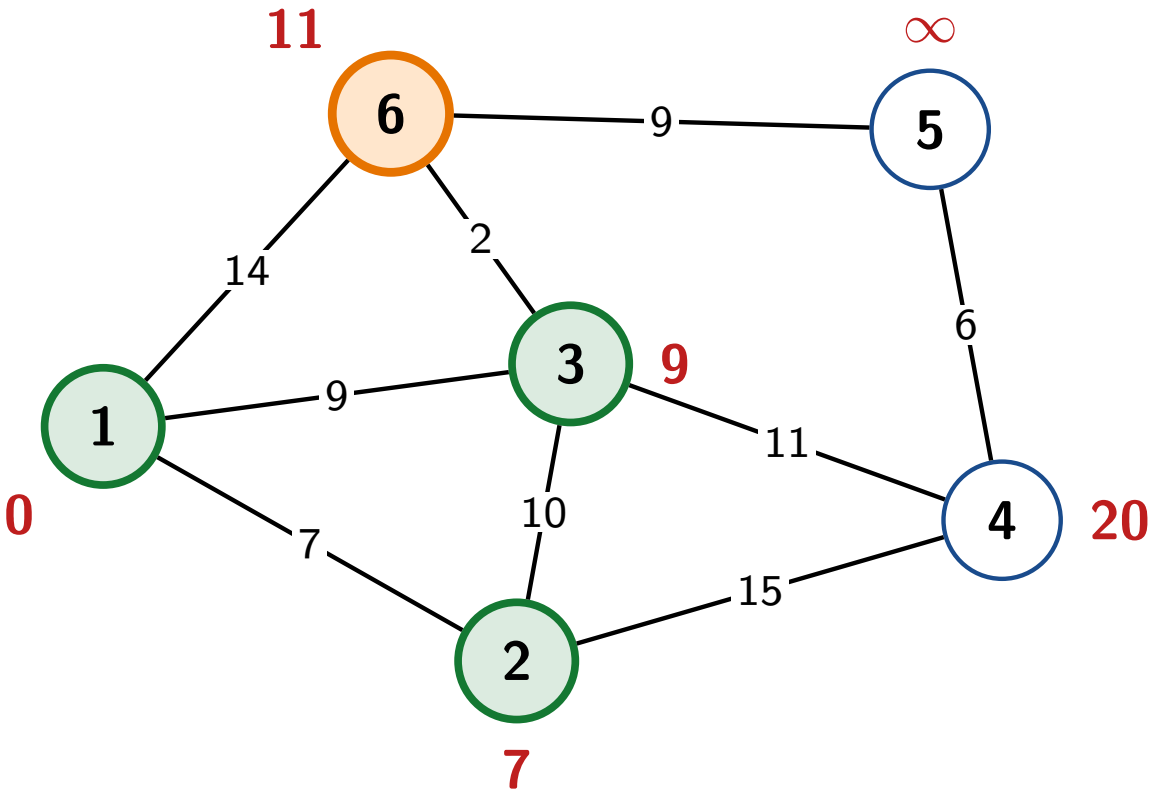
Dijkstra : exemple (source = 1)



$Q = \{4, 5, 6\}$

Relaxation du voisin 6 : $9 + 2 = 11 < 14$, mise à jour.

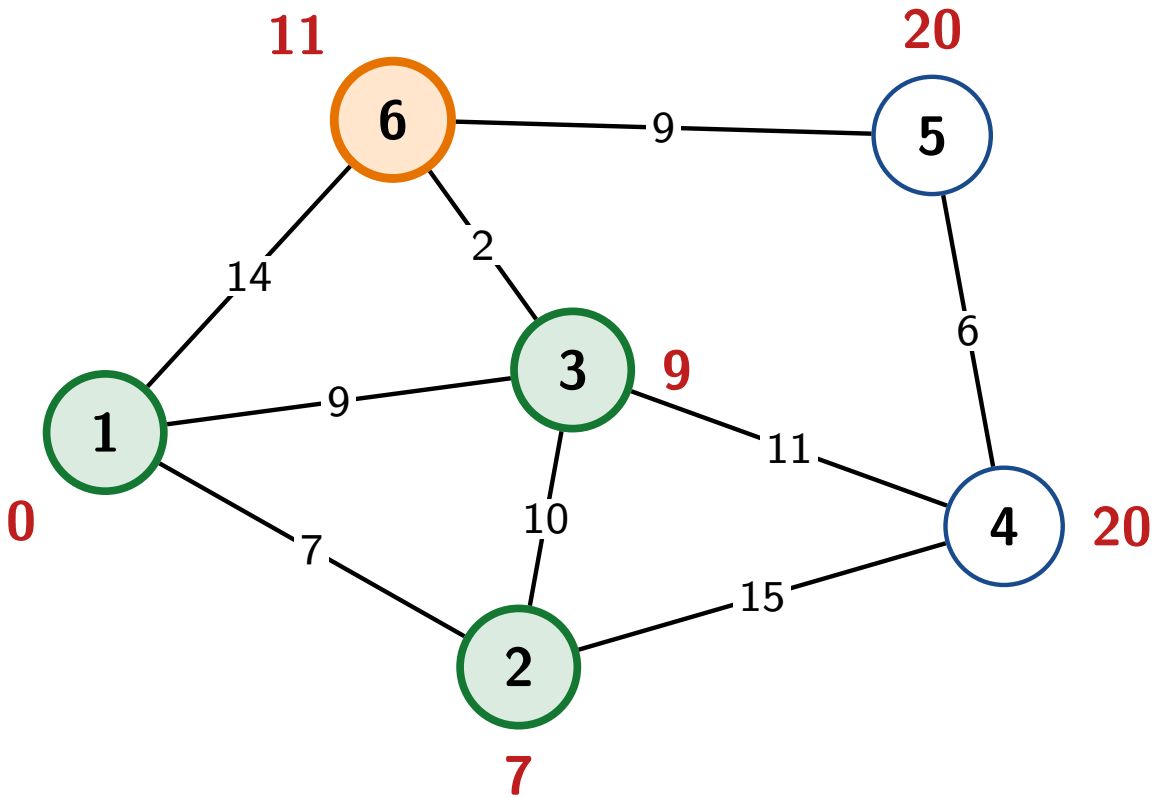
Dijkstra : exemple (source = 1)



$Q = \{4, 5\}$

On extrait 6 (dist 11).

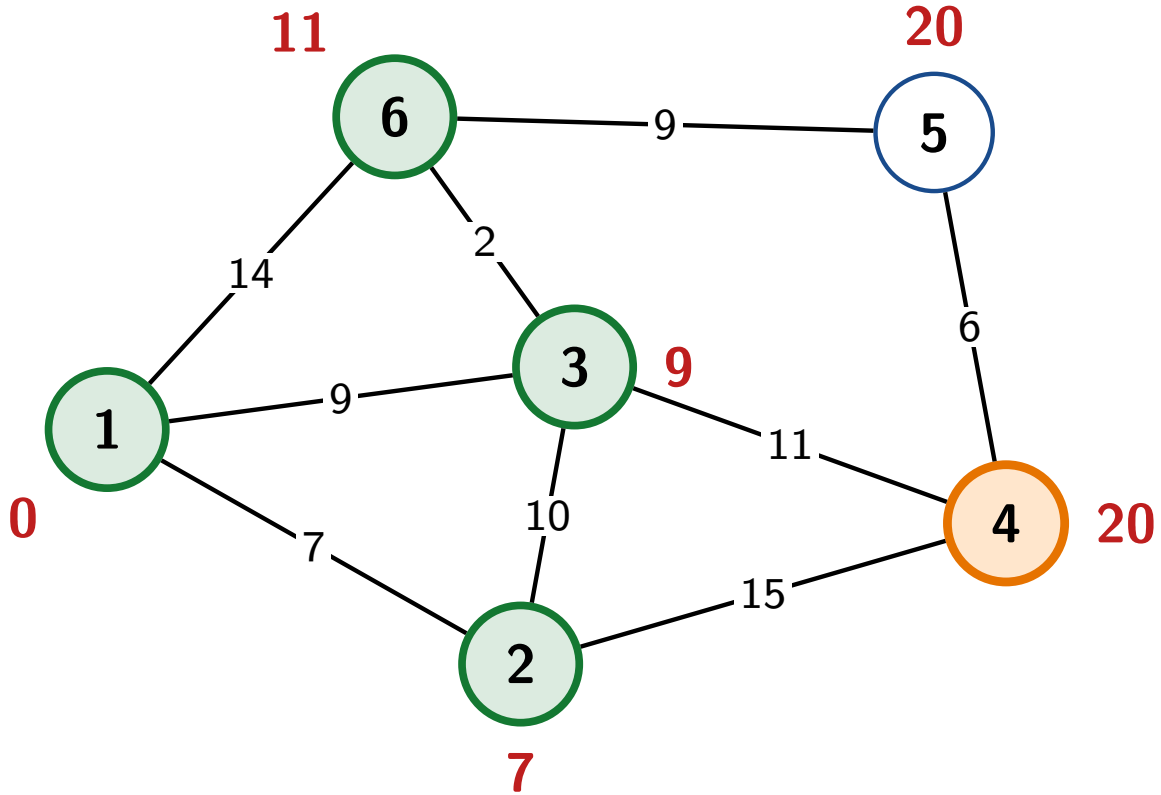
Dijkstra : exemple (source = 1)



$Q = \{4, 5\}$

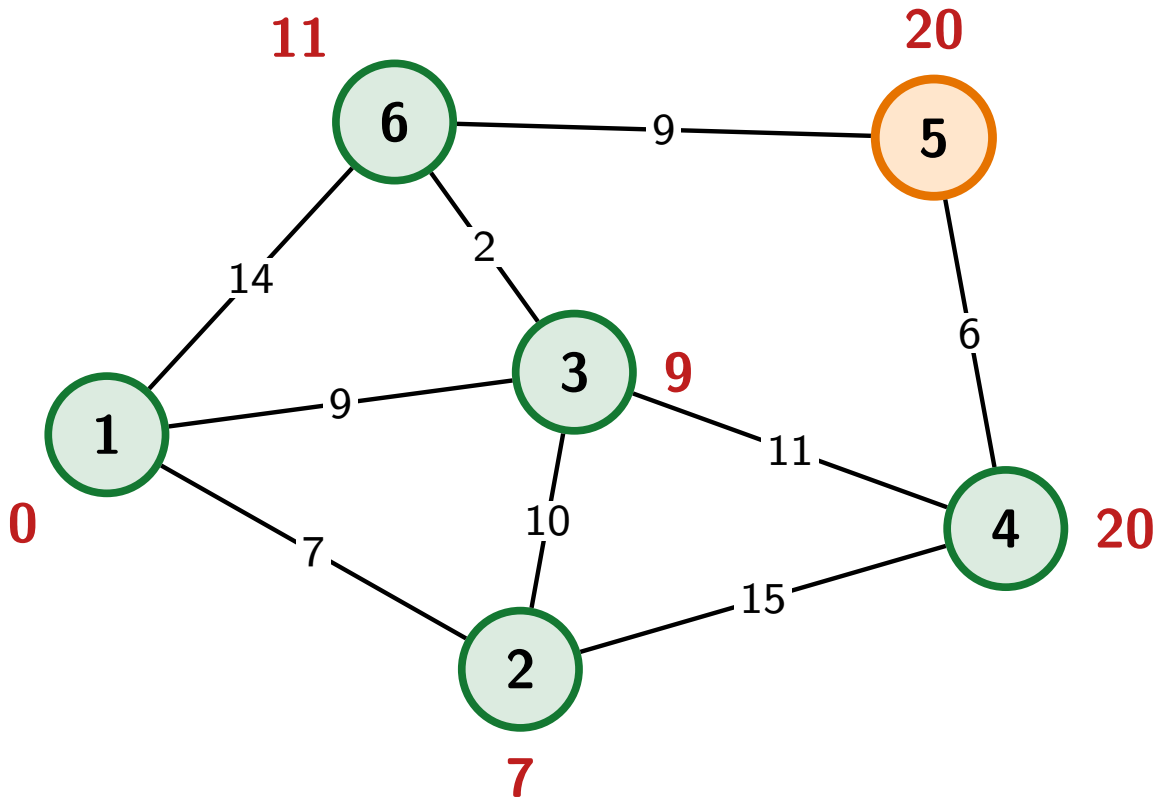
Relaxation du voisin 5 : $11 + 9 = 20$.

Dijkstra : exemple (source = 1)



On extrait 4 (dist 20). Voisin 5 : $20 + 6 = 26 > 20$, pas de changement.

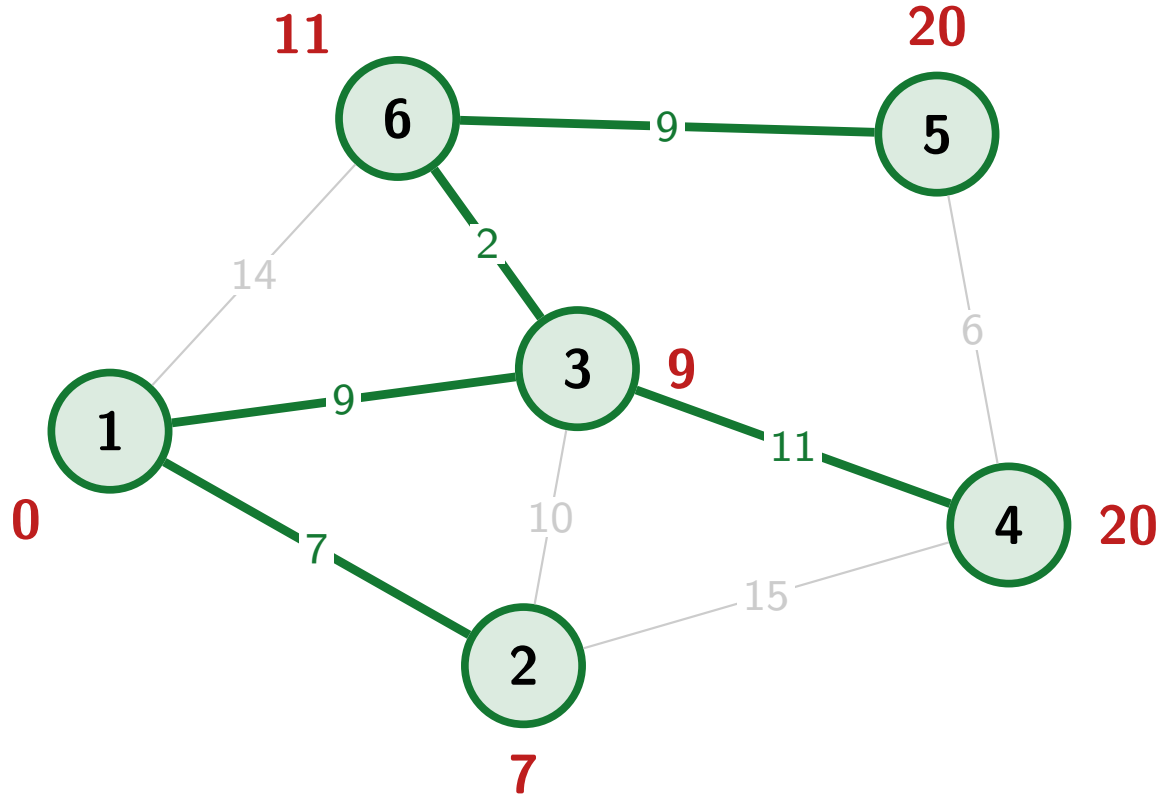
Dijkstra : exemple (source = 1)



$Q = \{\}$

On extrait 5 (dist 20). $Q = \emptyset$: **terminé**.

Dijkstra : exemple (source = 1)



L'arbre des plus courts chemins depuis 1 (via le tableau `previous`).

Pause exercice — Wooclap

États de liens



Code : **DJTYCHK**

Connectez-vous à `wooclap.com` avec le code de la séance.

OSPF : Open Shortest Path First

Protocole de type États de liens (RFC 2328, ou RFC 5340 pour la version 3 avec IPv6) :

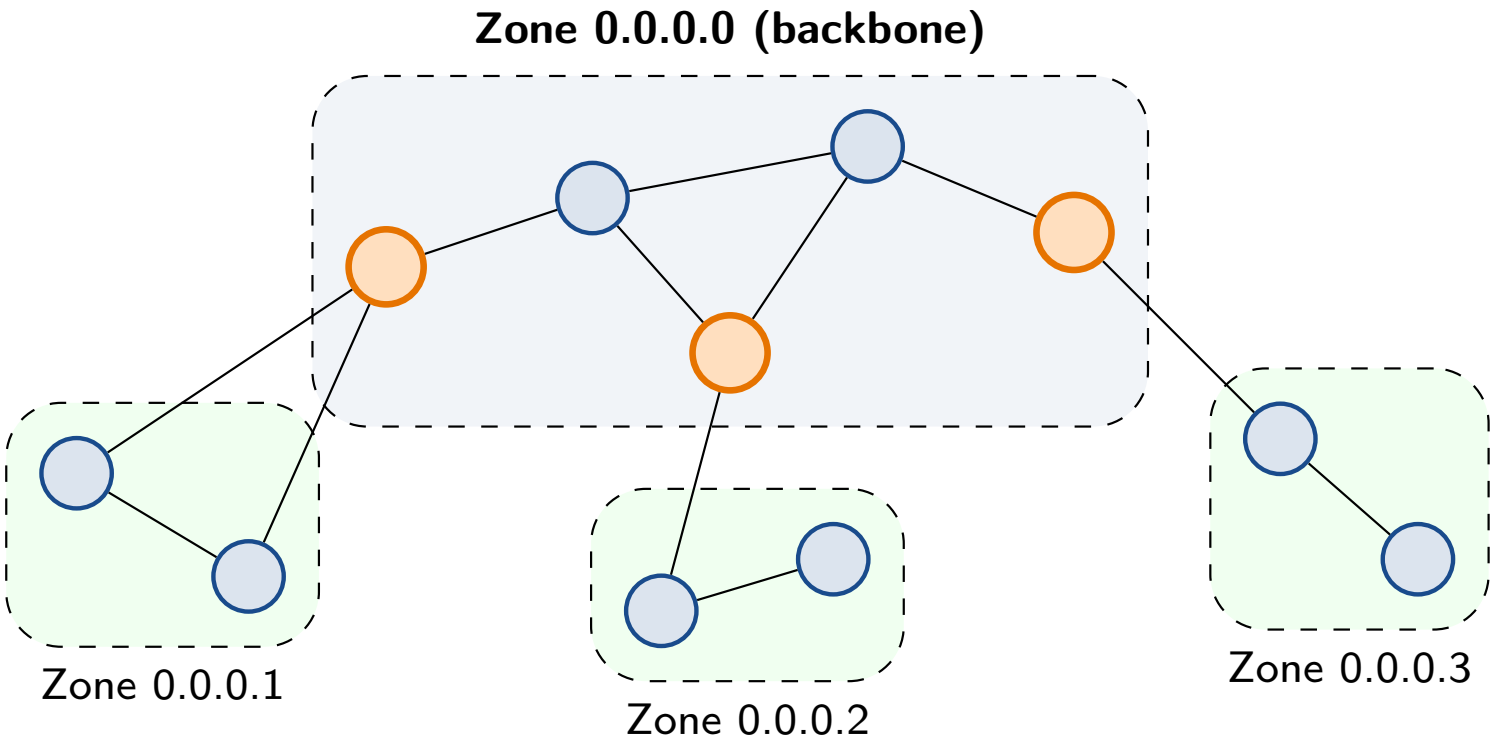
- directement dans IP, protocole 89
- découpage en zones
- plusieurs types de messages
- métrique 10^8/bande passante

Zones OSPF (version simple)

Pour gérer des réseaux composés d'un grand nombre de routeurs, on découpe l'ensemble des routeurs en zones numérotées sans chevauchement:

- épine dorsale (backbone) : zone 0
- autres zones toutes connectées à l'épine dorsale
- échanges simplifiés aux frontières entre zones
- routage à états de liens interne à chaque zone
- uniquement une route par défaut vers l'extérieur dans les zones non-backbone

Zones OSPF : schéma



Les routeurs **de bordure de zone** appartiennent à la fois au backbone et à une zone ; ils simplifient les échanges aux frontières.

Message HELLO

- Permet de découvrir les routeurs voisins sur les liens locaux
- Envoyés régulièrement (période Hello, e.g. toutes les 10 secondes)
- Voisin déclaré mort après un temps long sans recevoir de message Hello (période Dead, e.g. 40 secondes)

Messages OSPF

- DataBase Description : envoi de sa topologie sur demande pour initialisation
- Link State Request : demande de renseignements complémentaires sur un ou plusieurs liens
- Link State Update : envoi périodique de LSP
- Link State Acknowledgement : à envoyer pour chaque LSP

Link State Update

- Contient tous les Link State Advertisement (=LSP) reçus
- Un accusé de réception par LSA
- Différents types de LSA pour décrire les autres routeurs ou les réseaux connectés par exemple
- Utilisation d'un checksum pour garantir la fiabilité
- Durée de validité pour éviter que des LSA trop vieux soient considérés

Bilan : DV *vs* LS

	Vecteurs de distance	**États de liens**
Connaissance	locale	globale
Algorithme	Bellman-Ford	Dijkstra
Ce qui circule	sa table (distances), aux voisins	sa topologie locale, à tous (inondation)
Protocole	RIP (UDP 520)	OSPF (proto IP 89)
Difficultés	convergence lente, comptage à l'infini, passage à l'échelle	volume des LSP, géré par zones

Pause exercice — Wooclap

OSPF



Code : **OBSUGMJ**

Connectez-vous à `wooclap.com` avec le code de la séance.

Routage externe

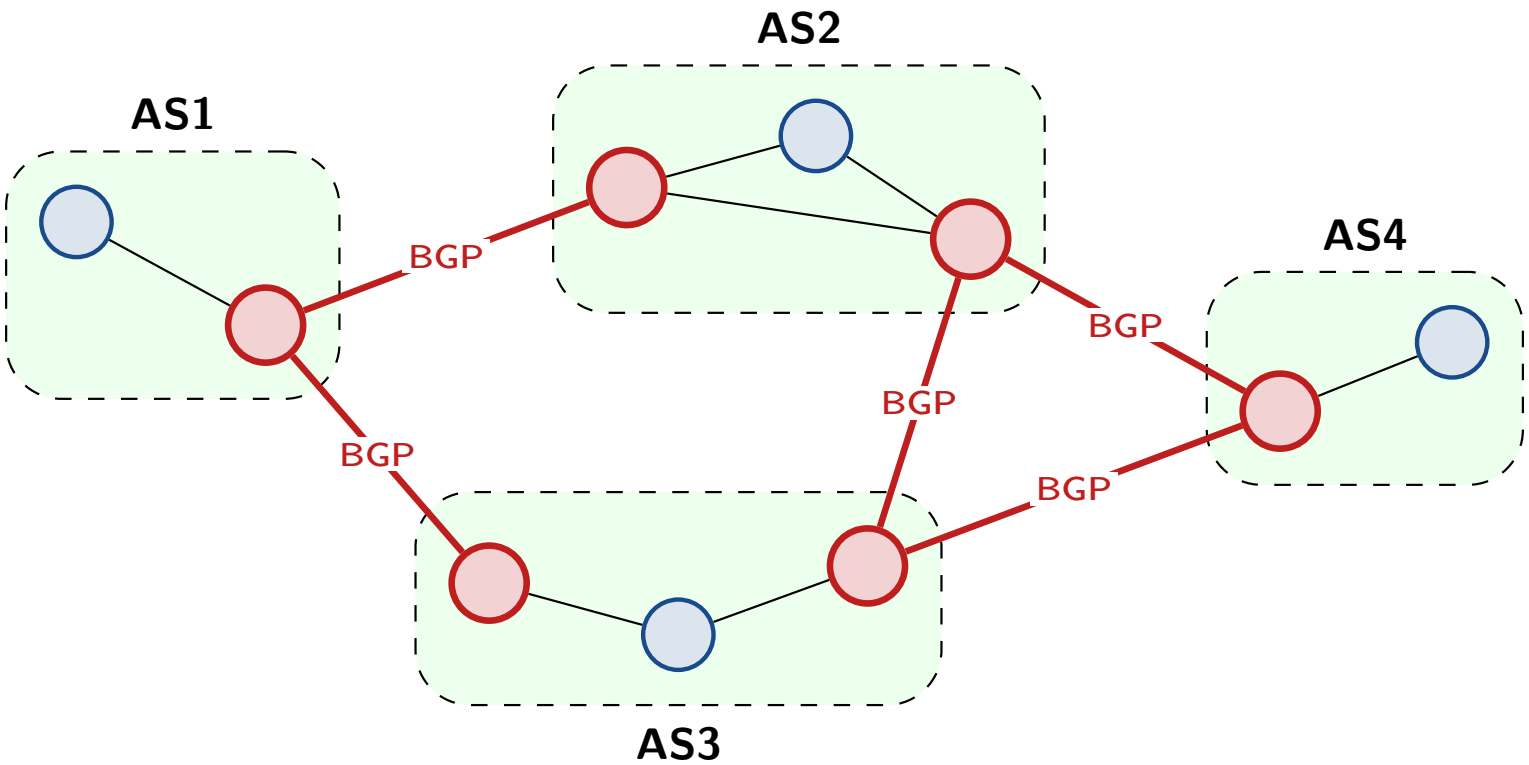
1. Le problème du routage
2. Vecteurs de distance (DV)
3. États de liens (LS)
4. Routage externe

Systèmes autonomes

L'internet est divisé en systèmes autonomes (AS) :

- contrôlés par un organisme (typiquement F.A.I.)
- identifiés par un numéro (ASN) unique fourni par l'IANA (délégation à un RIR)
- politique de routage interne
- autour de 85 000 AS à l'heure actuelle
- ASN codé sur 16 bits, depuis 2007 sur 32 bits

L'internet : un graphe d'AS



À l'intérieur de chaque AS : routage interne (RIP, OSPF,...). Entre les AS : les **routeurs de frontière** échangent des routes.

Types d'AS

Deux types d'AS :

- AS de transit : autorise la traversée par des paquets venant d'autres AS
- AS stub (resp. multihomed) : n'autorise pas la traversée et connecté à un (resp. plusieurs) AS de transit

Protocoles de routage

Deux types de protocoles :

- IGP (Interior Gateway Protocol), e.g. RIP, OSPF, IS-IS (plus confidentiel, LS)...
- EGP (Exterior Gateway Protocol) : Border Gateway Protocol.

BGP

- Moins de partage d'informations qu'à l'intérieur d'un AS
- Enjeux politiques, de sécurité... En particulier, pas de métrique sur le coût interne à un AS
- Protocole entre les routeurs aux frontières des AS
- RFC 4271, TCP port 179
- Table BGP de plus de 500 000 entrées pour IPv4 (limite de certains vieux routeurs atteinte)

Quel routage entre AS ? Les lieux d'échange

Les échanges se font entre routeurs aux frontières des AS :

- soit sur un lien privé entre les 2 routeurs
- soit un point d'interconnexion public auquel sont connectés des routeurs de différents AS

Quel routage entre AS ? Des relations diverses

Les AS ne traitent pas nécessairement d'égal à égal :

- relations Client-Fournisseur (*transit*) : le Client loue le passage par l'AS Fournisseur
- relations entre Pairs (*peering*) : les trafics sont jugés globalement équivalents et les AS font transiter l'ensemble du trafic des autres AS

Quel routage entre AS ? Organisation d'internet

Une hiérarchie historique :

- Tier-1 : groupe de grands FAI en peering. Pas de transit. E.g. Verizon, Deutsche Telekom, Orange
- Tier-2 : FAI nationaux ou régionaux. Peering entre eux. Relation de transit avec des FAI du Tier-1. E.g. Renater, Turk Telecom International
- Tier-3 : FAI locaux ou fournisseurs de contenus. Relation de transit avec des FAI du Tier-2. E.g. Université d'Orléans.

Récemment, quelques AS (Hyper Giants : Google, Amazon, Facebook...) concentrent une grande partie du trafic et ont des accords de peering avec la plupart des autres AS.

Quel routage entre AS ?

Nécessité d'implémenter la politique de routage (transit/peering) et de choisir des routes sans connaître les métriques intra-AS:

- échanges de routes entre AS
- filtres en entrée et en sortie
- algorithme de choix de la meilleure route basé sur un système de préférences

Vecteurs de chemins (Path Vector Protocol)

- Chaque routeur communique sa meilleure route vers chaque destination
- Métrique = nombre d'AS traversés
- Mise à jour uniquement en cas de changement de la route

Deux types de messages :

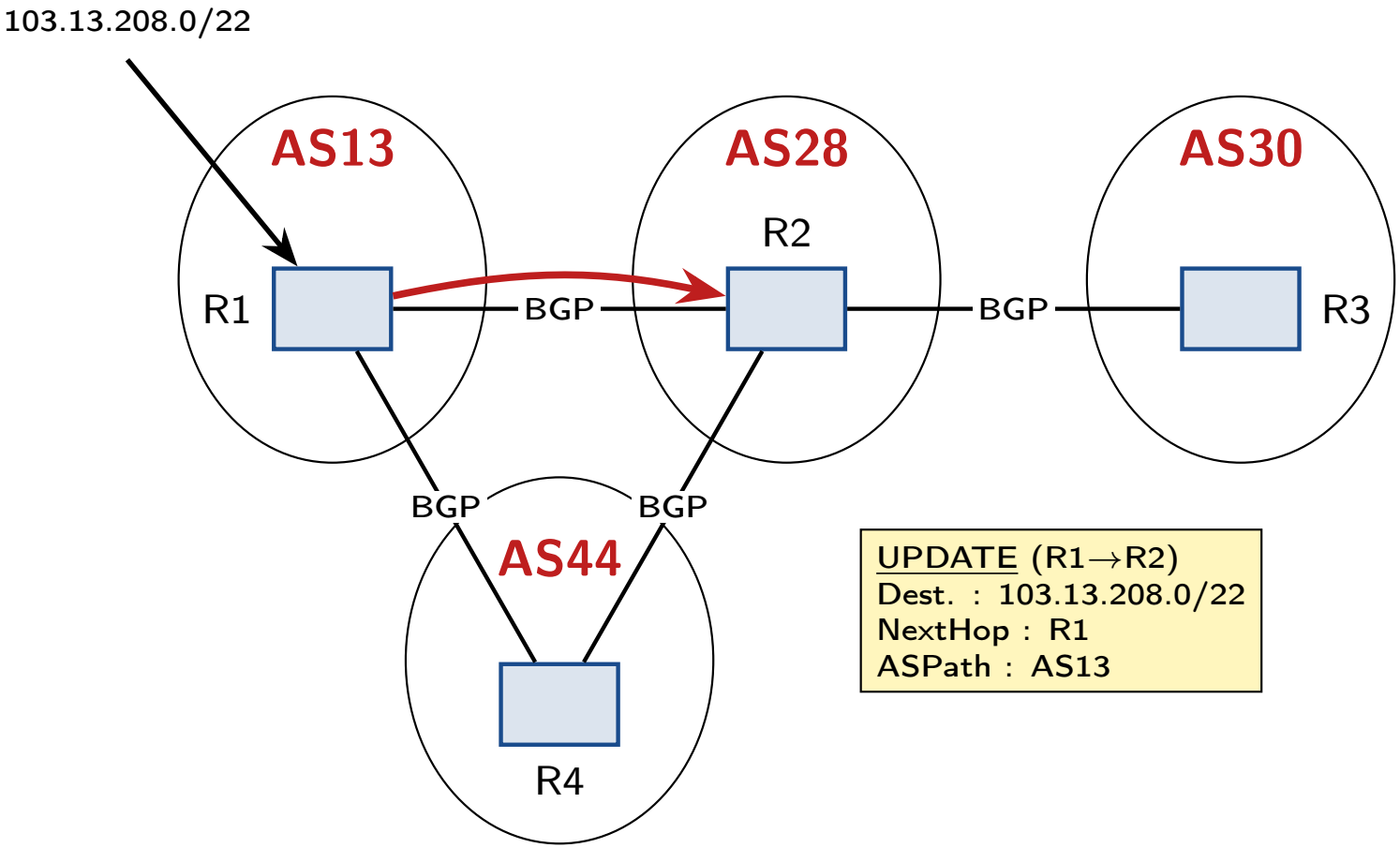
- UPDATE : description d'une route — destination, prochain saut et liste des AS traversés
- WITHDRAW : retrait d'une route précédemment annoncée — destination et liste des AS

Sessions BGP

Les routeurs BGP communiquent 2 par 2 :

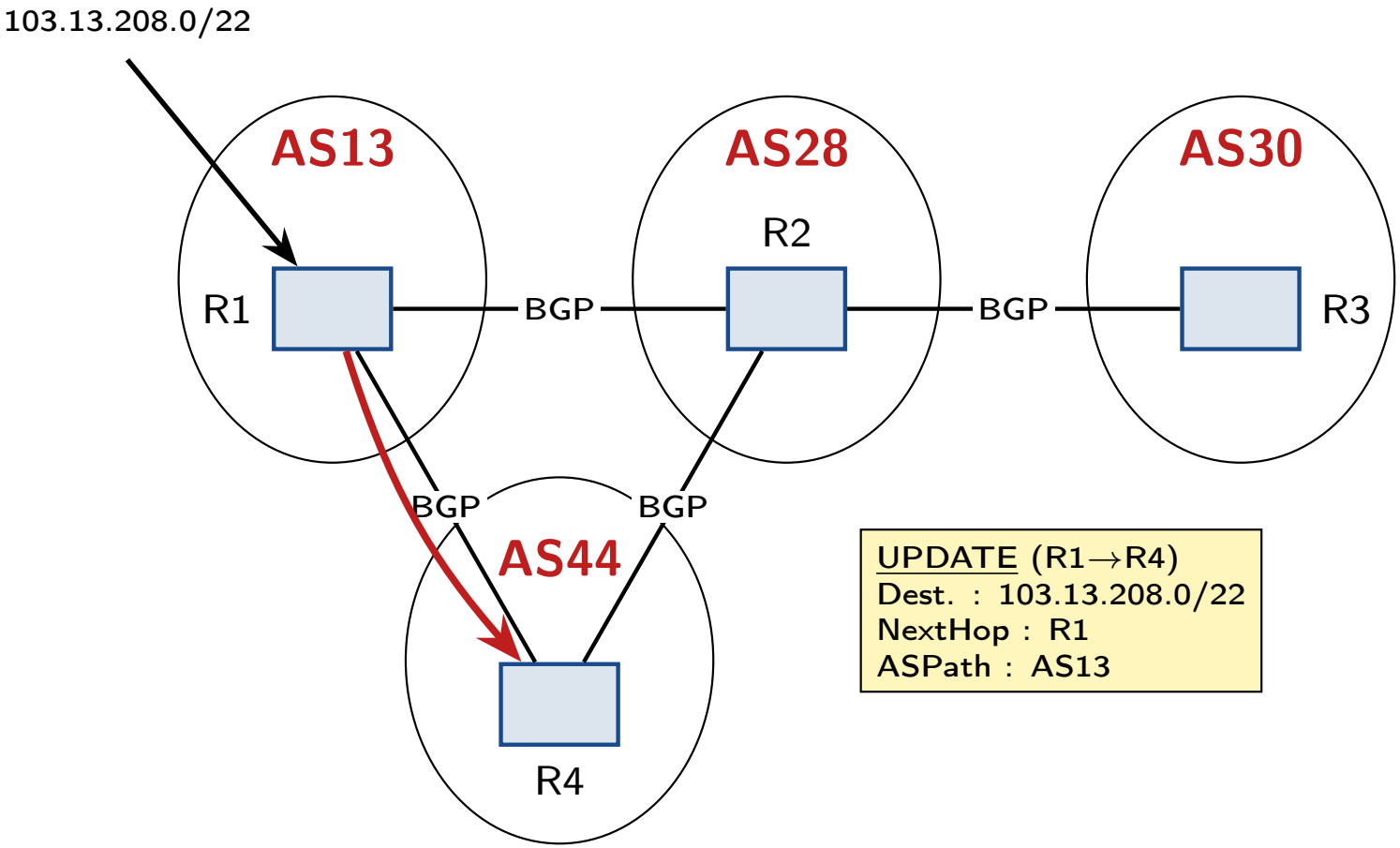
- établissement de la session BGP (dans TCP)
- échange de l'ensemble des routes connues
- durant la session : mise à jour ou retrait en cas de changement de topologie détecté

Propagation d'une route BGP : 103.13.208.0/22



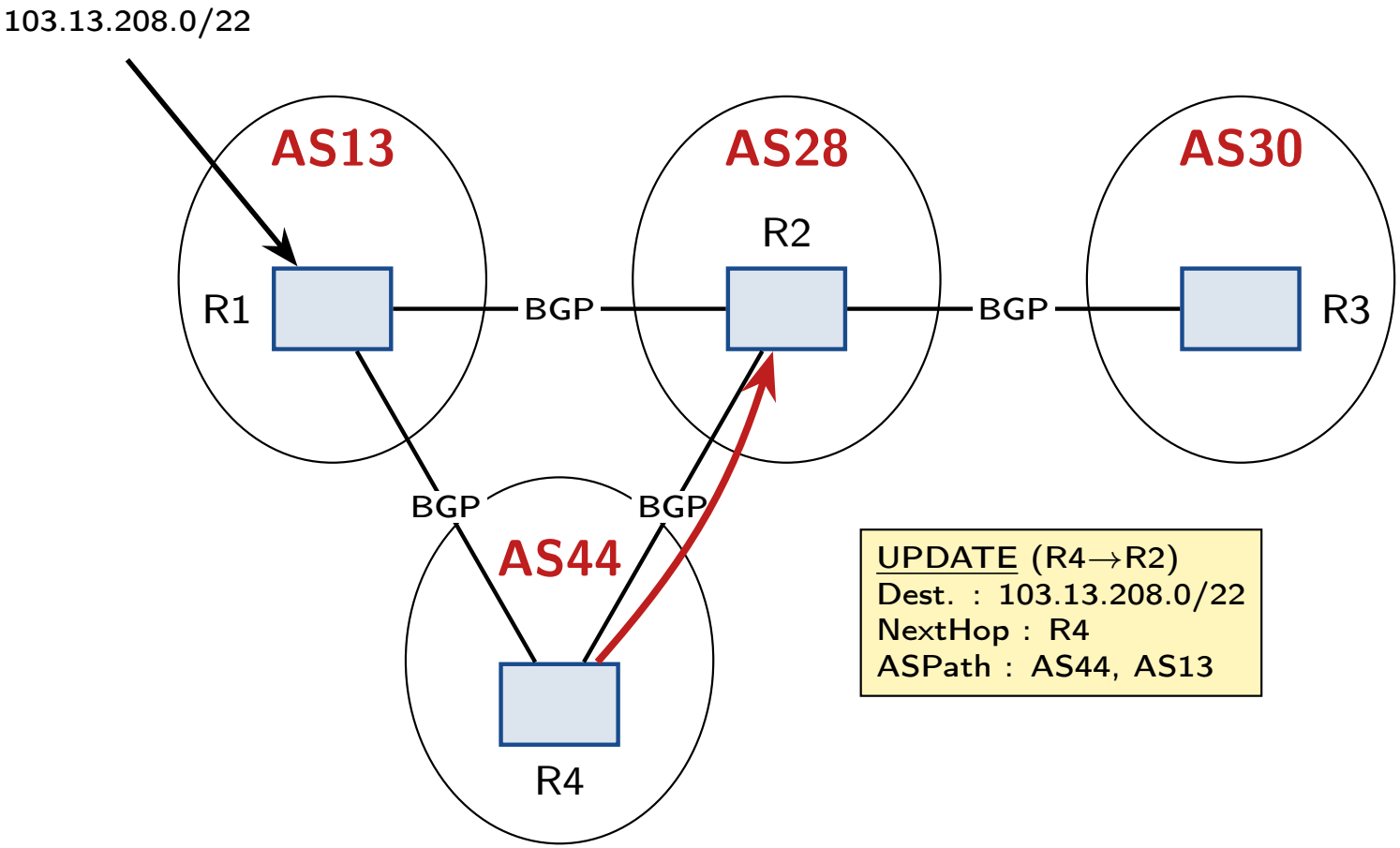
R1 (AS13) annonce à R2 le préfixe qu'il héberge : chemin = AS13.

Propagation d'une route BGP : 103.13.208.0/22



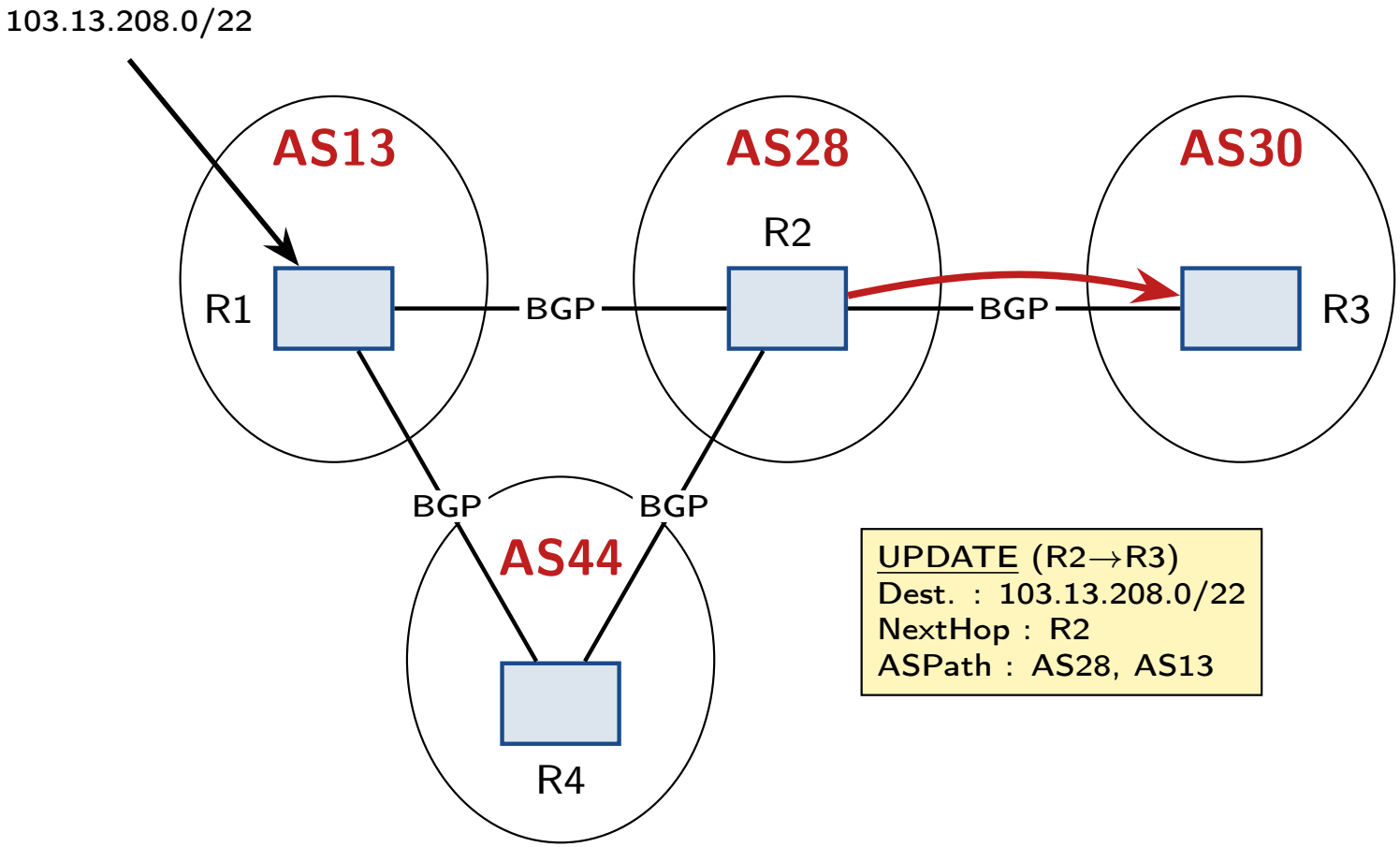
R1 annonce aussi la route à R4 (AS44).

Propagation d'une route BGP : 103.13.208.0/22



R4 relaie la route en préfixant son AS au chemin : AS44, AS13.

Propagation d'une route BGP : 103.13.208.0/22



R2 annonce à R3 sa meilleure route (2 AS traversés, via R1) : AS28, AS13.

Types de routes

- Routes statiques : configurées à la main
- Routes apprises par le protocole IGP (OSPF ou RIP)
- Routes apprises par les autres routeurs BGP

La politique de routage s'applique essentiellement aux routes apprises par BGP.

Filtre d'entrée : implémentation de la politique

Le filtre d'entrée :

- sélectionne pour chaque routeur voisin les routes acceptables
- attribue à chaque telle route un champ `local-pref` : plus la préférence est grande, plus la route est intéressante

La préférence dépend en général du coût et de la capacité

Exemple

- route proposée par un pair : `local-pref` = 100
- route proposée par un client : `local-pref` = 200
- route proposée par un fournisseur : `local-pref` = 50

BGP decision process

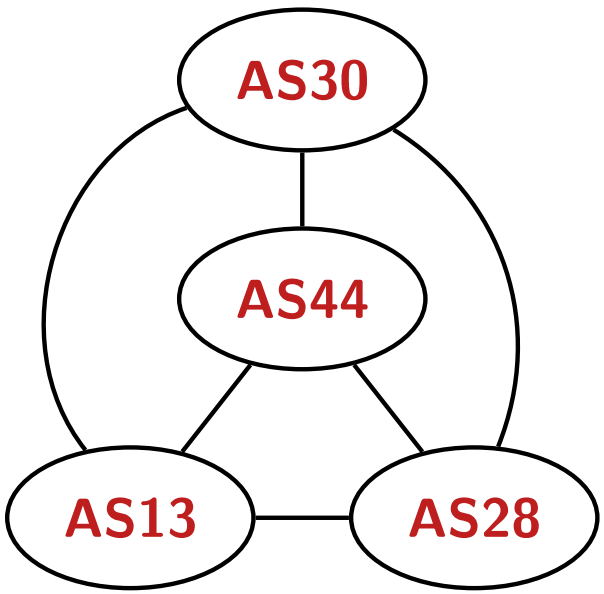
Une fois les routes et leur attribut `local-pref` connus, choix de la meilleure route :

1. plus grande préférence
2. plus petit vecteur de chemin (nombre d'AS traversés)
3. règle supplémentaire pour départager

En sortie, le filtre s'applique:

- sur la route sélectionnée par le processus de décision
- indépendamment pour chaque pair

Problèmes de routage



AS13, AS28 et AS30 préfèrent passer par leur voisin « de droite » que par AS44.

Les préférences locales des uns et des autres peuvent être incompatibles : la politique de chaque AS est choisie indépendamment.

Les routes BGP dans un AS

En pratique, chaque AS a plusieurs routeurs aux frontières, qui doivent s'échanger les routes reçues de l'extérieur.
Problème : les protocoles IGP (OSPF, RIP,...) ne sont pas conçus pour :

- trop de routes
- pas d'attributs `local-pref` ou *vector path*.

Solution : 2 sous-protocoles

eBGP

- on communique seulement la meilleure route
- des filtres sont définis pour chaque session

iBGP

- on communique les routes de chaque routeur d'entrée
- pas de filtre

Quels routeurs pour iBGP ?

Trois options :

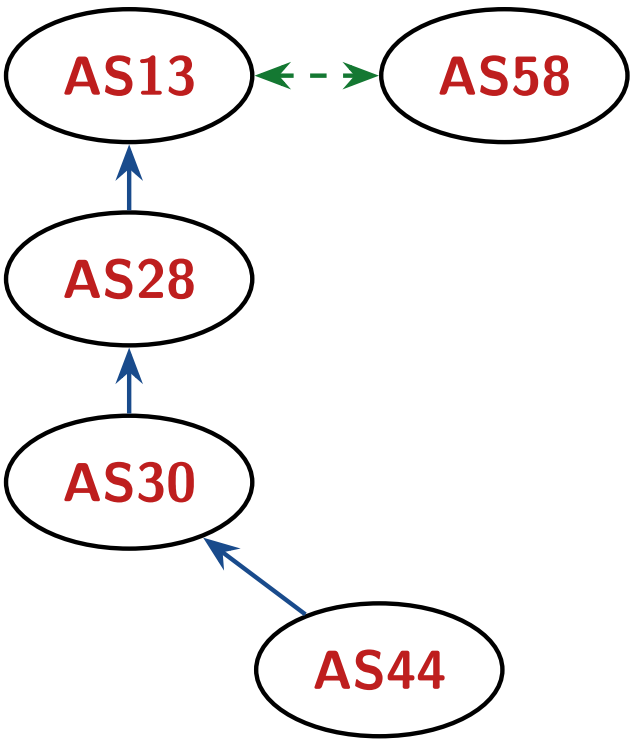
1. Utiliser IGP. Problèmes divers
2. Créer des tunnels entre routeurs frontière pour les sessions iBGP, en utilisant IGP. MPLS
3. Faire participer tous les routeurs de l'épine dorsale de l'AS à BGP. Les routeurs internes apprennent les routes En revanche, un routeur ne communique jamais une route apprise sur une session iBGP Inutile, tous les routeurs de l'épine dorsale sont connectés à tous les routeurs frontière : graphe des sessions iBGP complet sur l'épine dorsale.

Algorithme de décision (complet)

Critères par ordre d'importance :

1. Route avec le plus grand attribut `local-pref`
2. Route avec le plus petit vecteur de chemin
3. Plus petite route interne (Multi Exit Discriminator)
4. Route apprise par eBGP plutôt qu'iBGP
5. Plus proche premier saut
6. Pour départager : plus petite IP

Attribut `local-pref`

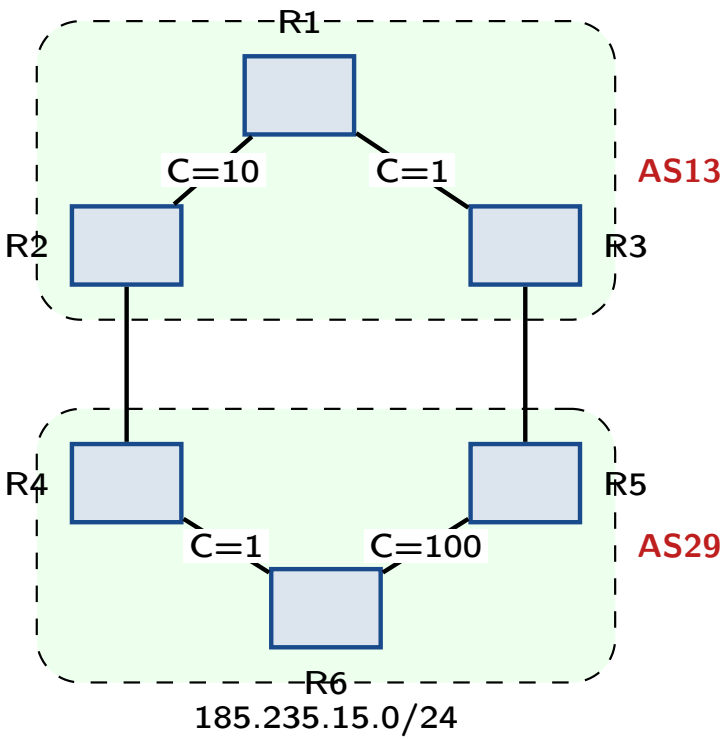


AS13 préfère passer par son client AS28.

Accord Client $\rightarrow$ Fournisseur

Accord de peering

Route interne (MED)

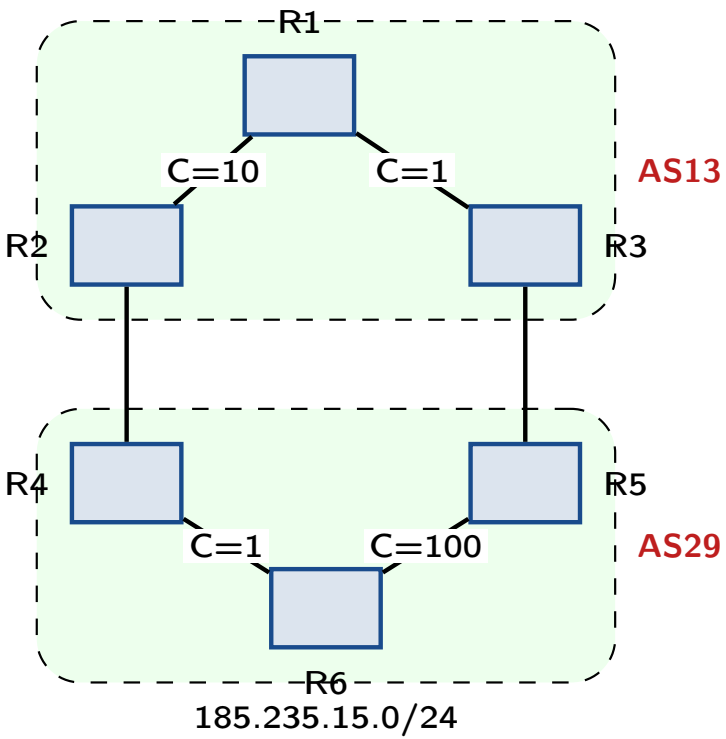


Pour le cas de plusieurs entrées dans AS29, AS29 communique la métrique interne.

R1 a 2 routes :

- R2, AS29, MED=1 ;
- R3, AS29, MED=100.

Route interne (MED)



Pour le cas de plusieurs entrées dans AS29, AS29 communique la métrique interne.

R1 a 2 routes :

- R2, AS29, MED=1 ;
- R3, AS29, MED=100.

Pause exercice — Wooclap

Routage externe



Code : **TJZDNQN**

Connectez-vous à `wooclap.com` avec le code de la séance.

Ce qu'il faut retenir

- Le routage remplit automatiquement les tables de routage : c'est le plan de contrôle
- À l'intérieur d'un AS (IGP) :
  - vecteurs de distance (Bellman-Ford) → RIP : simple, mais convergence lente, comptage à l'infini
  - états de liens (Dijkstra) → OSPF : inondation de la topologie, zones
- Entre AS (EGP) : BGP, vecteurs de chemins, politique de routage (filtres, `local-pref`, MED), eBGP/iBGP

Droits

La plupart des images proviennent, sont adaptées ou inspirées de CNP3 et sont diffusées sous licence CC-BY-SA-3.0.
Le texte est en partie une libre adaptation des transparents de CNP3 diffusés sous licence CC-BY-SA-3.0.
`http://inl.info.ucl.ac.be/CNP3`

Merci à Nicolas Ollinger.

Cette version : réorganisation du chapitre 3 du cours de M. Delacourt (Université d'Orléans), figures redessinées en TikZ, même licence.