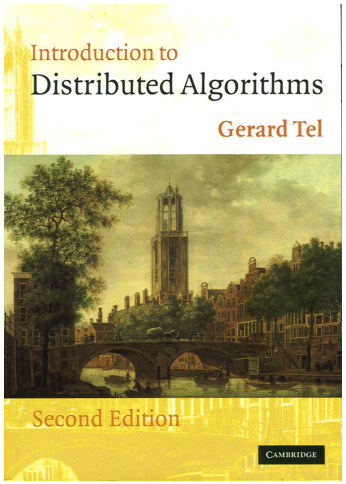
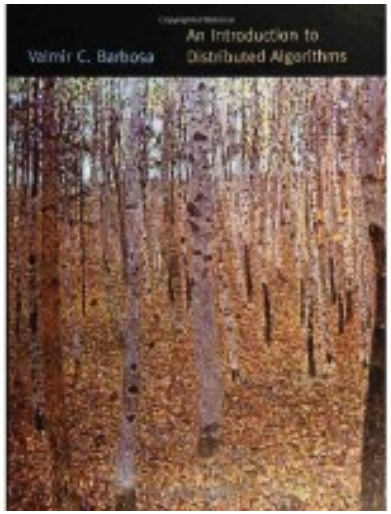
Cours APD : algorithmique parallèle et distribuée.

Johanne Cohen

PRISM/CNRS, Versailles, France.

Références





- Gérard Tel
  Introduction to Distributed Algorithms
  publié par Cambridge University Press
- Valmir C. Barbosa
  An Introduction to Distributed Algorithms
  publié par The MIT Press

Descriptif du cours

- Intervenant : Johanne Cohen

  `Johanne.Cohen@prism.uvsq.fr`

- 6 séances de 3 heures en salle S106

- Un examen :

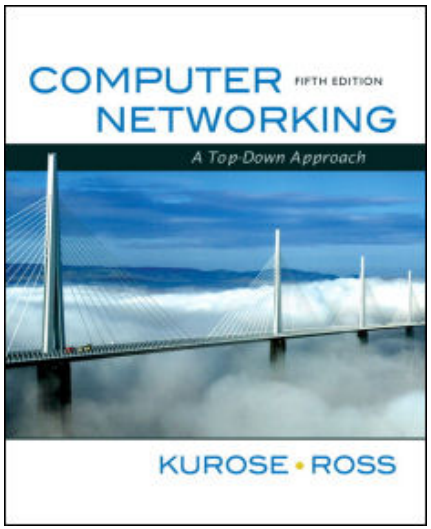
- Support de cours : transparents + exercices sur la page

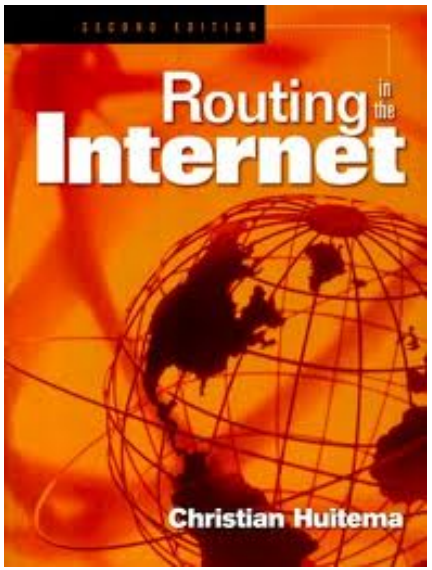
  `http://www.prism.uvsq.fr/ joco`

Calcul distribué de plus court chemins.

Johanne Cohen

Références





- Jim Kurose, Keith Ross
  Computer Networking,
  publié par Addison-Wesley Longman
- Christian Huitema
  Routing in the Internet
  publié aux éditions Eyrolles.

Plan

Comment trouver son chemin

Objectif: Trouver un algorithme qui calcule le trajet le plus court.

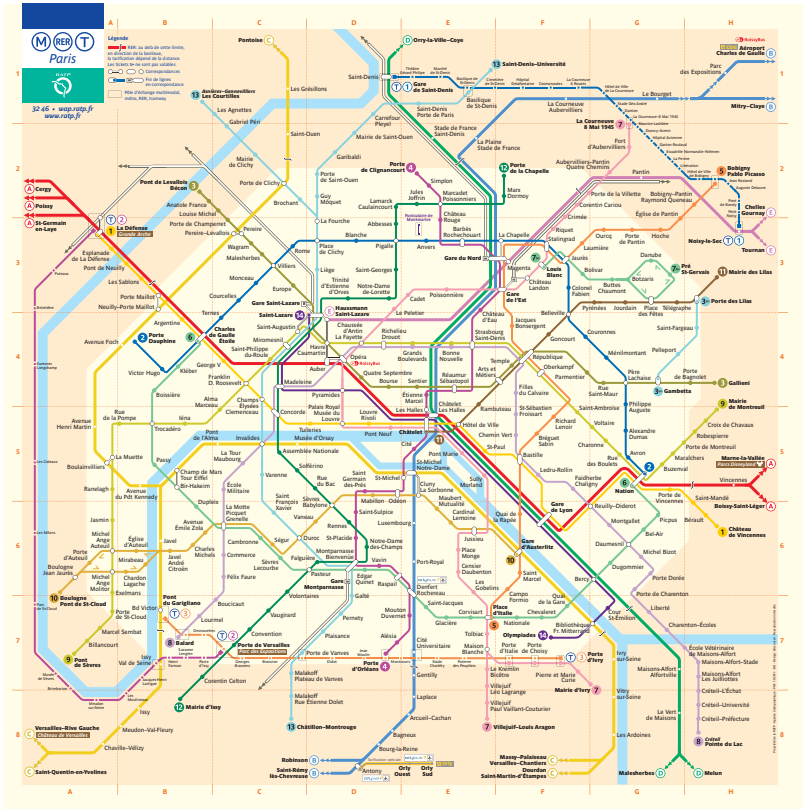


Figure: Plan des lignes parisiennes (*source : site de la RATP)*

Comment trouver son chemin

Modélisation sous forme de graphe.

Définition : Un graphe G est donné par un couple $G = (V, E)$, où

- V : ens. de sommets

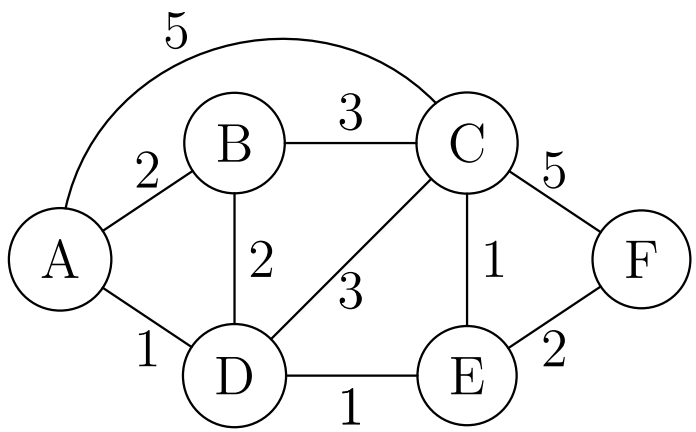
$$V = \{A, B, C, D, E, F\}$$

- E : ens. d'arêtes $(E \subseteq V \times V)$

$$(B, C) \in E$$

Un graphe pondéré est un graphe où chaque arête e a un poids $w(e)$

$$w(B, C) = 3$$



Comment trouver son chemin

Modélisation sous forme de graphe.

Plan des lignes = graphe pondéré

- une station = un sommet
- un tunel entre deux stations = une arête
- Coût = distances interstations, le nombre de stations, ...

Définition : Un graphe G est donné par un couple $G = (V, E)$, où

- V : ens. de sommets

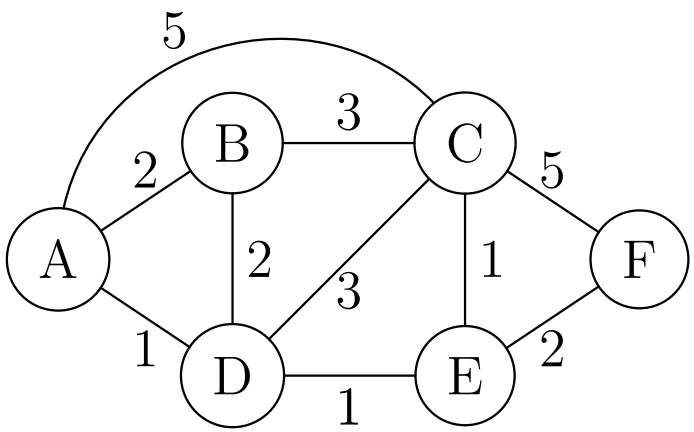
$$V = \{A, B, C, D, E, F\}$$

- E : ens. d'arêtes $(E \subseteq V \times V)$

$$(B, C) \in E$$

Un graphe pondéré est un graphe où chaque arête e a un poids $w(e)$

$$w(B, C) = 3$$



Plan

Problème du plus courts chemins

Problème du plus cours chemins

- Entrée : un graphe pondéré, un sommet singulier.
- Objectif : Construire l'arbre couvrant de plus court chemins.

Problème du plus courts chemins

Problème du plus cours chemins

- Entrée : un graphe pondéré, un sommet singulier.
- Objectif : Construire l'arbre couvrant de plus court chemins.

Algorithme centralisé

- Un algorithme centralisé a une connaissance complête de l'entrée du problème :

$$\text{Entrée : } \begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$$

- Un algorithme centralisé retourne une solution complête

$$\text{Sortie : } \begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$$

Algorithme centralisé de Dijkstra

Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$

Sortie : $\begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$

1. Initialisation de la source s
   - 1.1 $d[s] \leftarrow 0$; $\pi[s] \leftarrow s$
   - 1.2 Pour chaque sommet v de V faire $\begin{cases} \pi(v) \leftarrow NIL \\ d(v) \leftarrow \infty^+ \end{cases}$

Algorithme centralisé de Dijkstra

Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$

Sortie : $\begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$

1. Initialisation de la source s
   - 1.1 $d[s] \leftarrow 0$; $\pi[s] \leftarrow s$
   - 1.2 Pour chaque sommet v de V faire $\begin{cases} \pi(v) \leftarrow NIL \\ d(v) \leftarrow \infty^+ \end{cases}$

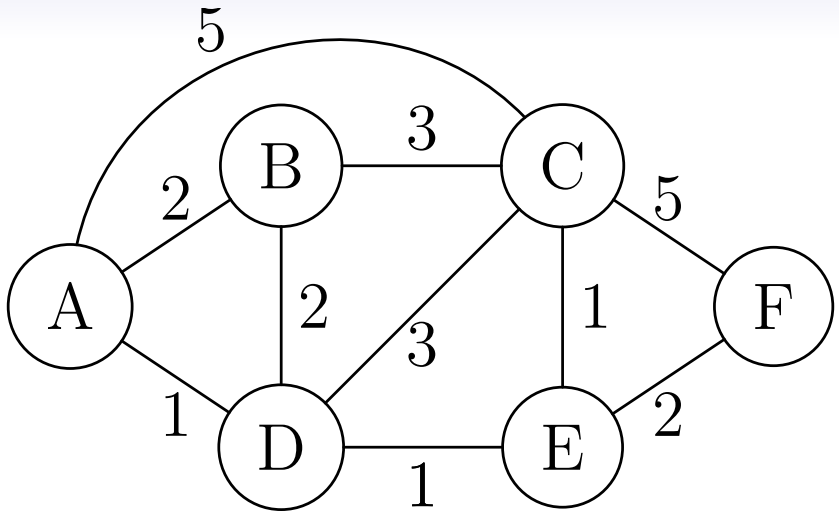
Algorithme centralisé de Dijkstra

Entrée : $\begin{cases} \text{Un graphe } G = (V, E) \text{ avec une source } s \\ \text{Une fonction de poids } w : E \to \mathbb{R}^+ \end{cases}$

Sortie : $\begin{cases} \text{Un vecteur distance } d \\ \text{Une fonction père } \pi : V \to V \end{cases}$

1. Initialisation de la source s
   - 1.1 $d[s] \leftarrow 0$; $\pi[s] \leftarrow s$
   - 1.2 Pour chaque sommet v de V faire $\begin{cases} \pi(v) \leftarrow NIL \\ d(v) \leftarrow \infty^+ \end{cases}$
2. $\mathcal{Q} \leftarrow V$; $\mathcal{S} \leftarrow \emptyset$;
3. Tant que $(\mathcal{Q} \neq \emptyset)$ faire
   - 3.1 $u \leftarrow$ Extraire-Le-Minimum$(\mathcal{Q}, d)$;
   - 3.2 $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$;
   - 3.3 Pour chaque sommet v voisin de u faire
     Si $(d[v] > d[u] + w(u, v))$ alors $\begin{cases} d[v] \leftarrow d[u] + w(u, v) \\ \pi(v) \leftarrow u \end{cases}$
4. retourner d et π

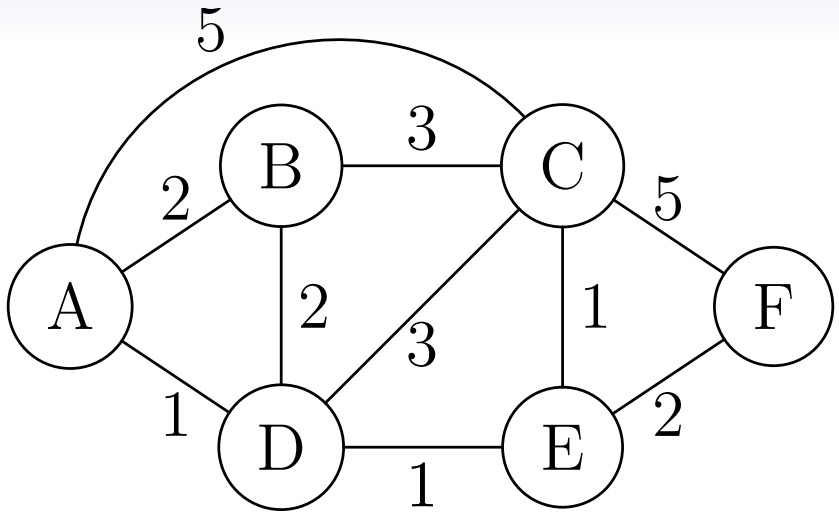
Exemple : la source est le sommet A



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$

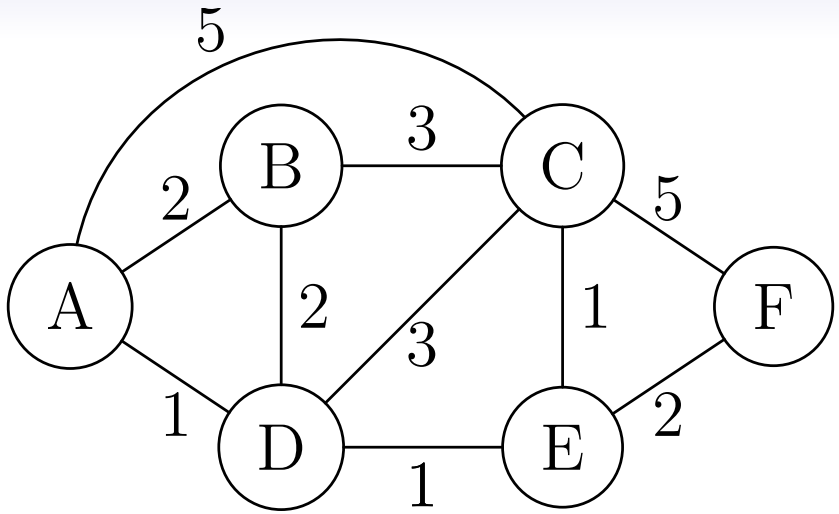
Exemple : la source est le sommet A



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$

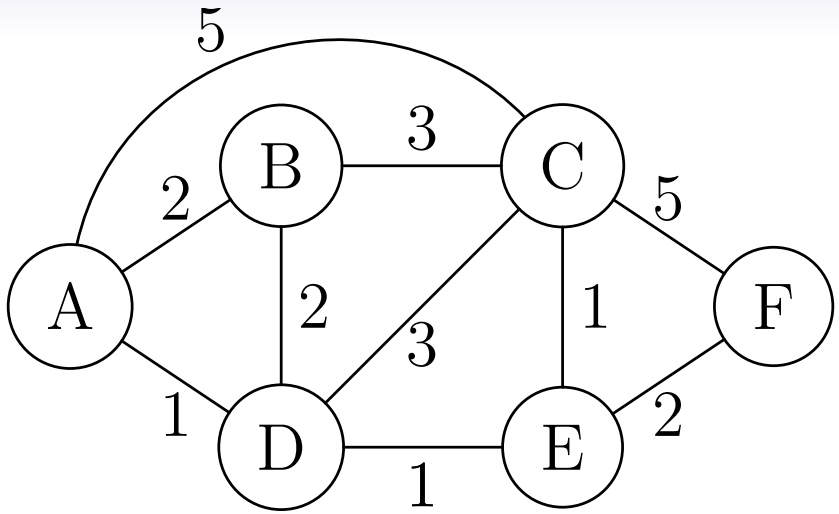
Exemple : la source est le sommet A



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$

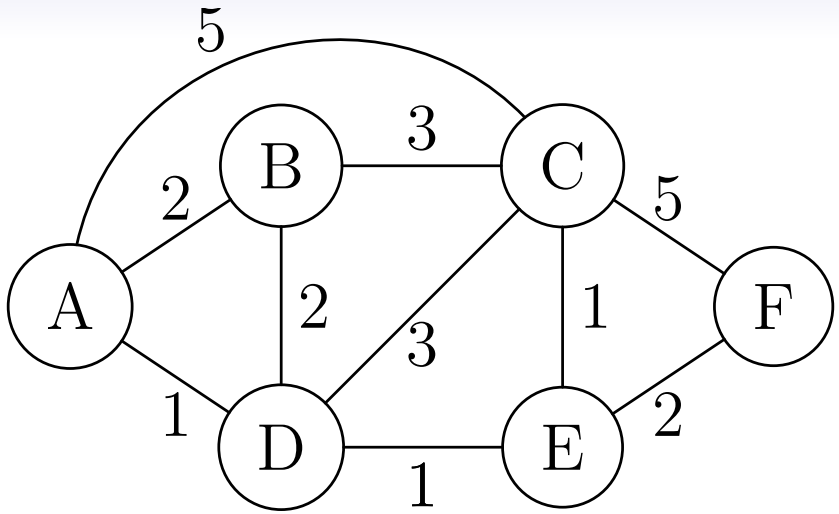
Exemple : la source est le sommet A



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$

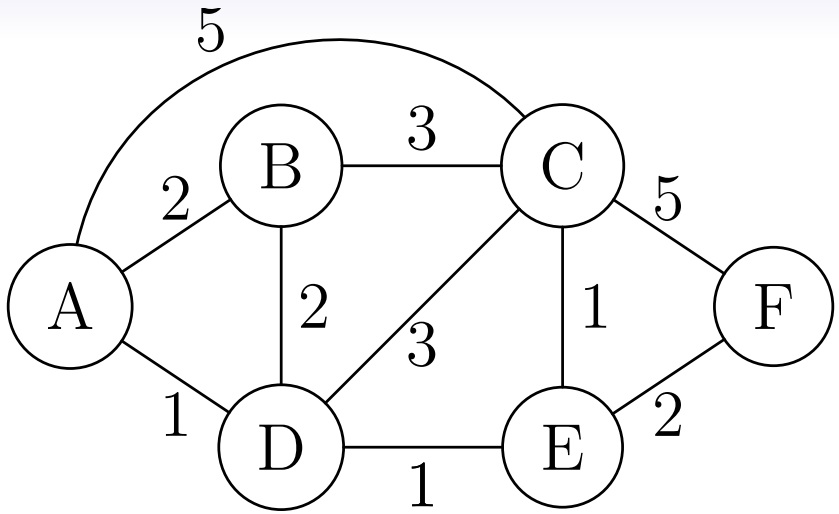
Exemple : la source est le sommet A



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
4	$\{CF\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$

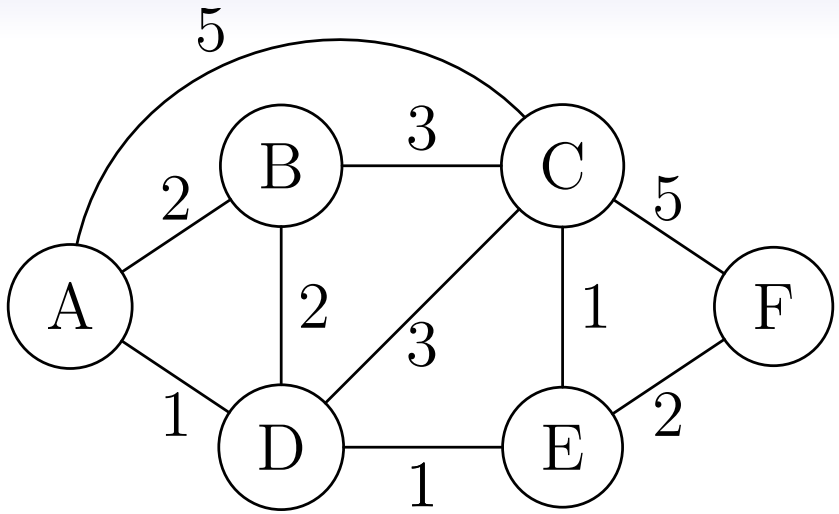
Exemple : la source est le sommet A



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
4	$\{CF\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$
5	$\{F\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$

Exemple : la source est le sommet A



les couples correspondent à $(d(.), \pi(.))$

	$\mathcal{Q}$	A	B	C	D	E	F
I	$\{ABCDEF\}$	$(0, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
1	$\{BCDEF\}$	$(0, A)$	$(2, A)$	$(5, A)$	$(1, A)$	$(\infty, \emptyset)$	$(\infty, \emptyset)$
2	$\{BCEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
3	$\{CEF\}$	$(0, A)$	$(2, A)$	$(4, D)$	$(1, A)$	$(2, D)$	$(\infty, \emptyset)$
4	$\{CF\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$
5	$\{F\}$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$
6	$\emptyset$	$(0, A)$	$(2, A)$	$(3, E)$	$(1, A)$	$(2, D)$	$(4, E)$

Complexité de l'algorithme : $O(|V|^2)$

Plan

Problème de la construction de l'arbre des plus courts chemins

Algorithme centralisé

Algorithme distribué

- Pourquoi rendre le problème distribué ?
- Comment le rendre distribué ?
- Quel type d'algorithmes ?
  - Connaissance du voisinnage
  - Diffusion du voisinnage
  - Gestion de la panne

Plus précisément

Algorithme distribué

- Pourquoi rendre le problème distribué ?
- Comment le rendre distribué ?
- Quel type d'algorithmes ?
  - Connaissance du voisinnage
  - Diffusion du voisinnage
  - Gestion de la panne

Problème de routage.

Comment transiter les messages ?

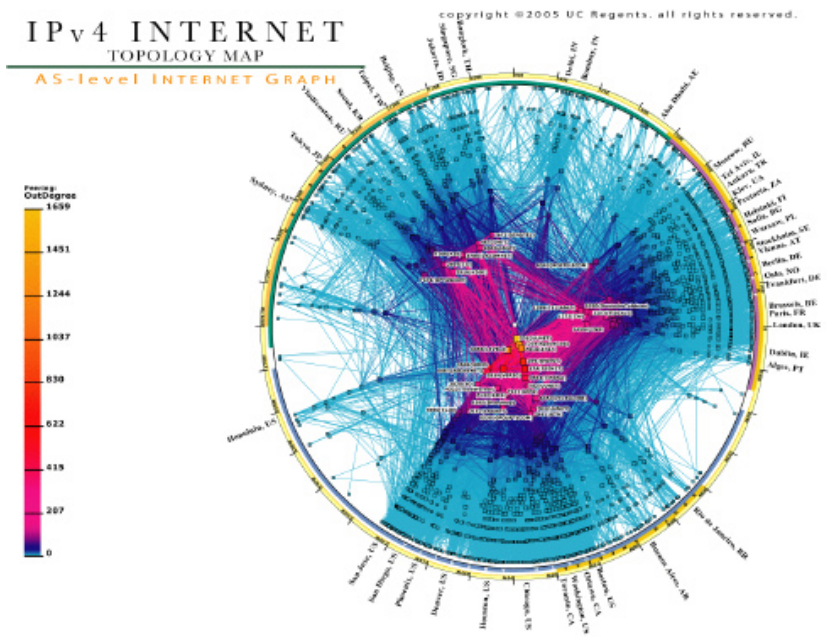


Figure: "Graphe" de l'Internet *(source : le site de CAIDA)*

Problème de routage.

Comment transiter les messages ?

En sachant que chaque nœud a une vision partielle du réseau

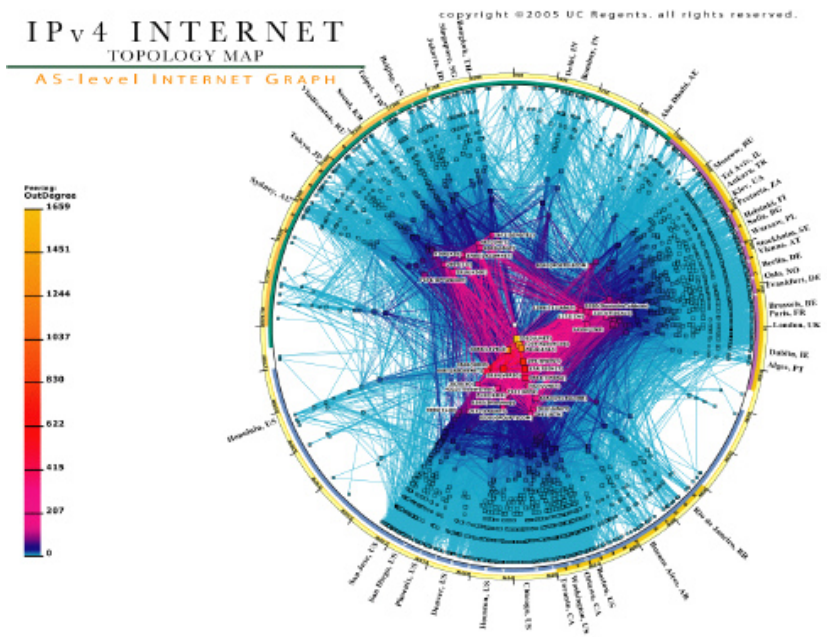


Figure: "Graphe" de l'Internet *(source : le site de CAIDA)*

Comment transiter les messages

Modélisation sous forme de graphe.

Réseau = graphe

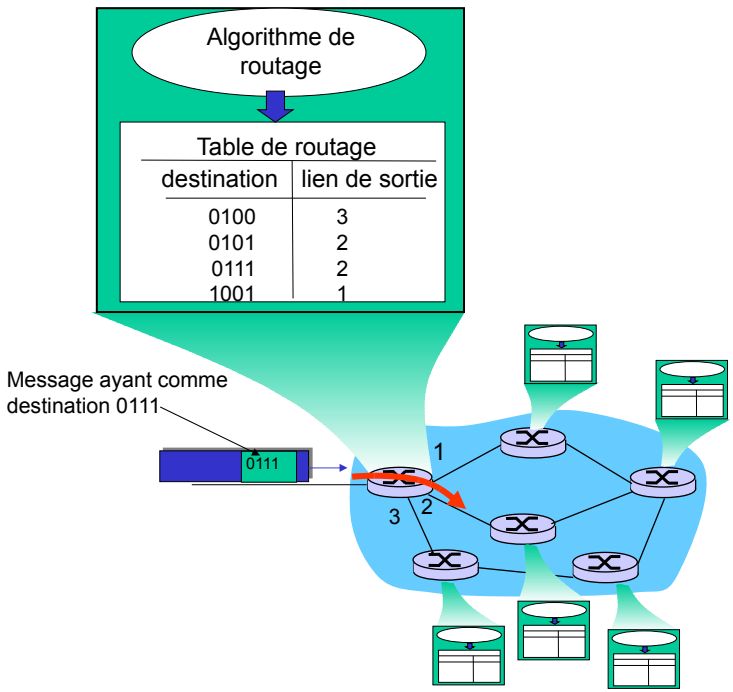
- routeurs = sommets
- liens physiques = arêtes
- Coût = charge, délai, congestion

Aiguillage au sein d'un routeur :

- Calcul des routes

  stoquées dans une table

- Sélection d'un port de sortie en fonction d'une destination



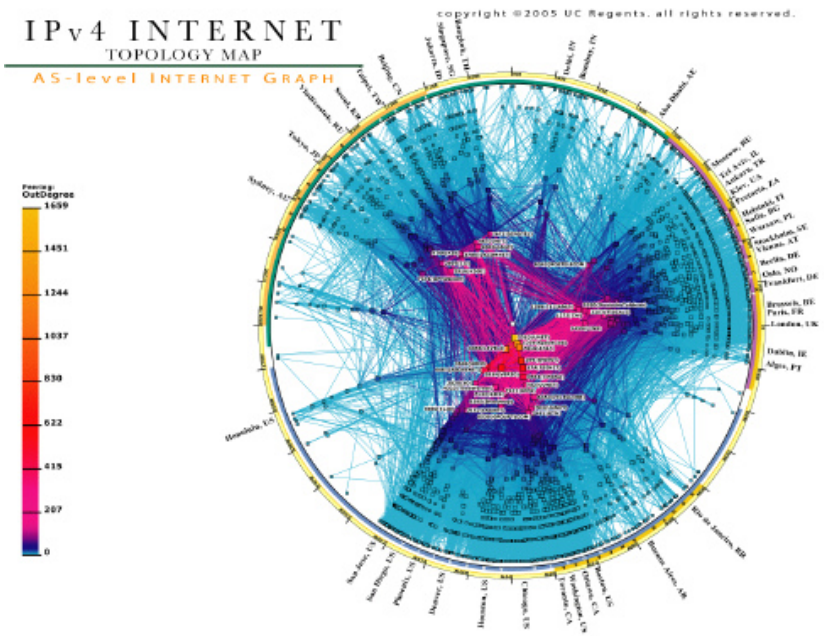
Plus précisément

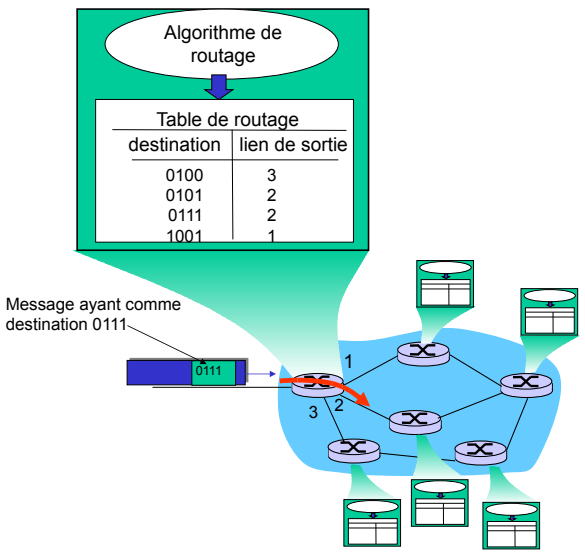
- Algorithme distribué
  - Pourquoi rendre le problème distribué ?
  - Comment le rendre distribué ?
  - Quel type d'algorithmes ?
    - Connaissance du voisinnage
    - Diffusion du voisinnage
    - Gestion de la panne

Comment le rendre distribué ?

Calculer les tables de routages en chaque nœud du réseau en tenant en compte :

- des changements locaux de coût de liens, des pannes, d'une méconnaissance du système.





Les systèmes distribués

Un système distribué est un ensemble de N sites (ou processus) et un système de communication.

Chacun de ces sites agit comme un automate : il réalise des opérations qui modifient son état.

Motivation :

- Echange d'information
- Partage de ressource
- Réplication de données.
- Parallélisation.
- Modularité.

Un site/un processus

- Elément effectuant une tâche.
  - Exécution d'un ensemble d'instructions (atomiques)
  - Une instruction correspond à un événement local
- Il possède et connaît
  - un état local (ens. des données, valeurs de ces variables)
  - un identifiant unique (en général)
- Il ne possèdent aucune connaissance sur les autres processus.

Un site/un processus

- Elément effectuant une tâche.
  - Exécution d'un ensemble d'instructions (atomiques)
  - Une instruction correspond à un événement local

Ici, un sommet du graphe

- Il possède et connaît
  - un état local (ens. des données, valeurs de ces variables)
  - un identifiant unique (en général)

Ici, sa mémoire et son identifiant

- Il ne possèdent aucune connaissance sur les autres processus.

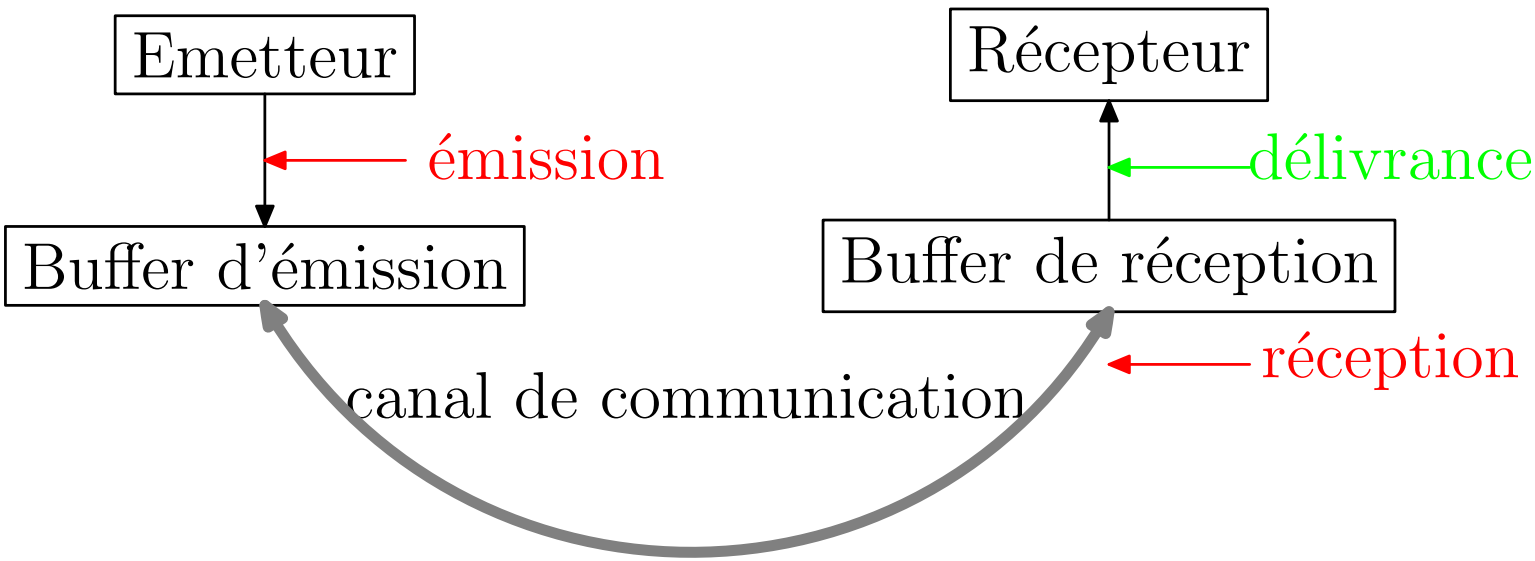
Ici, il y a aucune connaissance sur ces voisins,
uniquement qu'il possède des liens de communications

Un système de communications

Échange d'information entre sites : envoi de messages.

Canal de communication

- Canal logique de communication point à point
  - pour la communication entre deux processus
  - Transit de messages sur un canal



Un système de communications

Échange d'information entre sites : envoi de messages.

Canal de communication

- Uni ou bi-directionnel
- Fiable ou non
- Ordre de réception par rapport à l'émission
- Synchrone ou asynchrone
  - Synchrone : l'émetteur et le récepteur se synchronisent pour réaliser l'émission et/ou la réception
  - Asynchrone : pas de synchronisation entre émetteur et récepteur
- Taille des tampons de message cotés émetteur et récepteur

En résumé :

Un système distribué est composé

- de plusieurs sites ayant un état local (variables locales,...)
- d'un système de communications permettant l' échange d'informations entre sites.
- d'une configuration ou état global= ensemble des états locaux de tous les sites et de tous les canaux de communications.

Un système est modélisé par un état qui évolue selon les événements survenus.

3 types d'événements

local	changement de l'état d'un processus
du à une communication	émission d'un message
	réception d'un message

Plus précisément

- Algorithme distribué
  - Pourquoi rendre le problème distribué ?
  - Comment le rendre distribué ?
  - Quel type d'algorithmes ?
    - Connaissance du voisinnage
    - Diffusion du voisinnage
    - Gestion de la panne

Quel type d'algorithmes ?

- Rendre l'algorithme centralisé en un algorithme distribué.
  1. Algorithme de Dijkstra
  2. Algorithme de Bellman-Ford

Quel type d'algorithmes ?

- Rendre l'algorithme centralisé en un algorithme distribué.
  1. Algorithme de Dijkstra
     - Il faut avoir une connaissance complête du réseau
  2. Algorithme de Bellman-Ford

Quel type d'algorithmes ?

- Rendre l'algorithme centralisé en un algorithme distribué.
  1. Algorithme de Dijkstra
     - Il faut avoir une connaissance complête du réseau
  2. Algorithme de Bellman-Ford
     - Il faut avoir une connaissance incomplête du réseau

Quel type d'algorithmes ?

- Rendre l'algorithme centralisé en un algorithme distribué.
  1. Algorithme de Dijkstra
     - Il faut avoir une connaissance complête du réseau
  2. Algorithme de Bellman-Ford
     - Il faut avoir une connaissance incomplête du réseau
       chaque sommet va connaître sa distance entre lui et les autres.

Quel type d'algorithmes ?

- Rendre l'algorithme centralisé en un algorithme distribué.
  1. Algorithme de Dijkstra
     - Il faut avoir une connaissance complête du réseau
     - Cela correspond au protocole de routage OSPF (RFC 2328)
  2. Algorithme de Bellman-Ford
     - Il faut avoir une connaissance incomplête du réseau

       chaque sommet va connaître sa distance entre lui et les autres.
     - Cela correspond aux protocoles de routage RIP (RFC 1058) et BGP (RFC 4271)

Premier Algorithme

Basé sur l'algorithme de Dijkstra

Une fois que la topologie est connue, l'algorithme de Disjkstra est exécuté afin de connaitre le chemin le plus court pour aller à un sommet arbitraire.

résultat = une table de routage incluant les chemins et les ports permettant d'atteindre chaque noeud

Premier Algorithme

Basé sur l'algorithme de Dijkstra

Une fois que la topologie est connue, l'algorithme de Disjkstra est exécuté afin de connaitre le chemin le plus court pour aller à un sommet arbitraire.

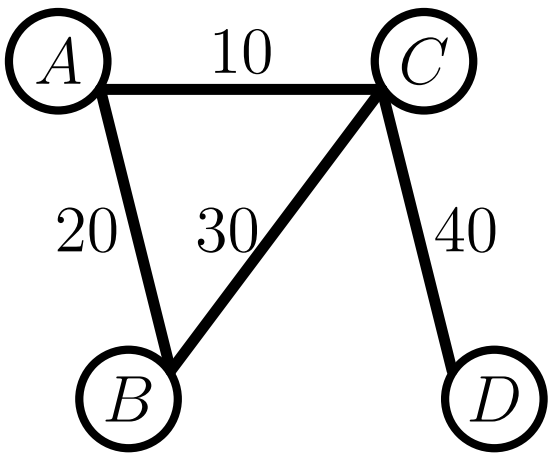
résultat = une table de routage incluant les chemins et les ports permettant d'atteindre chaque noeud

Comment connaître la topologie ?

Connaissance de la topologie = une table

La topologie est stoquée sous forme de table.

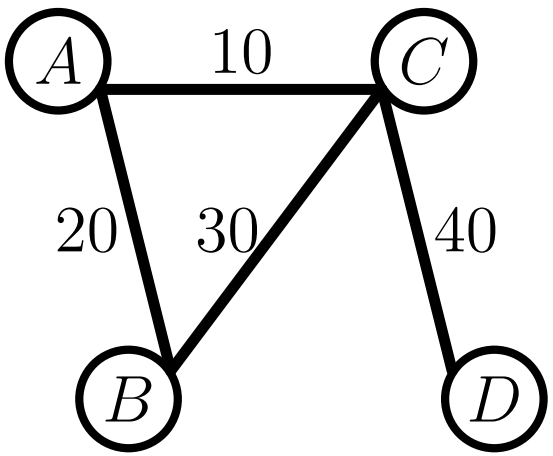
De	Vers	Liens	Coût
A	B	20	1
A	C	10	1
B	A	20	1
B	C	30	1
C	A	10	1
C	B	30	1
C	D	40	1
D	C	40	1



Connaissance de la topologie = une table

La topologie est stoquée sous forme de table.

De	Vers	Liens	Coût
A	B	20	1
A	C	10	1
B	A	20	1
B	C	30	1
C	A	10	1
C	B	30	1
C	D	40	1
D	C	40	1

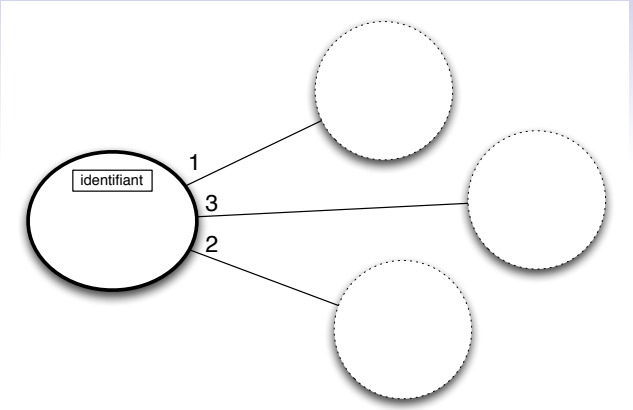


Comment la construire ?

Plus précisément

- Algorithme distribué
  - Pourquoi rendre le problème distribué ?
  - Comment le rendre distribué ?
  - Quel type d'algorithmes ?
    - Connaissance du voisinnage
    - Diffusion du voisinnage
    - Gestion de la panne

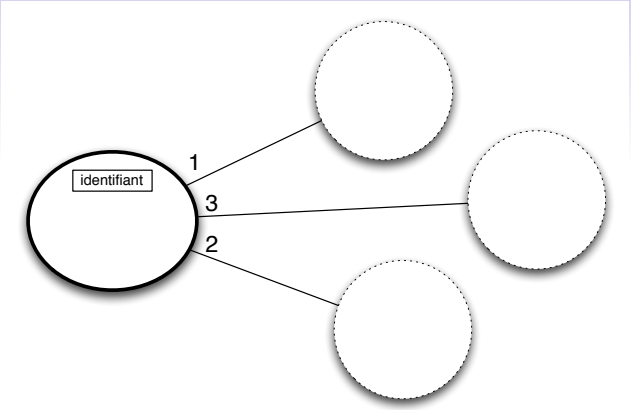
Etat d'un site



1. Au départ,

Connaissance du site : son identifiant + les ports de sortie

Etat d'un site



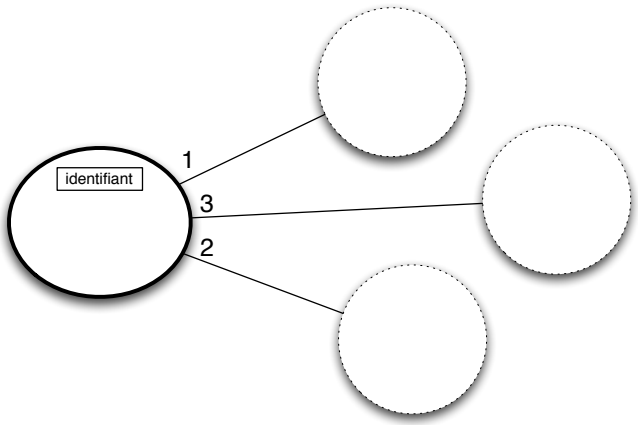
1. Au départ,

Connaissance du site : son identifiant + les ports de sortie

2. Lancement du processus de connaissance du voisinnage

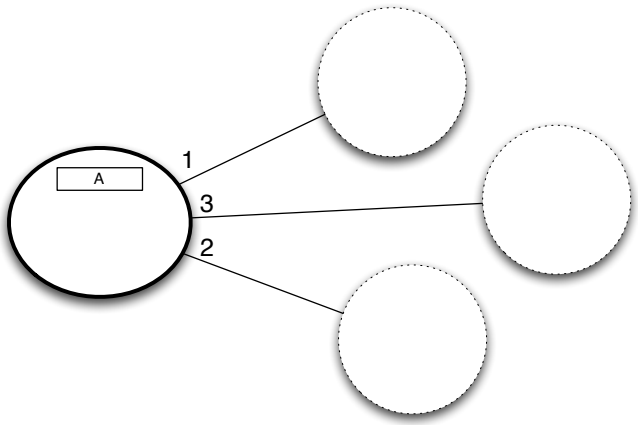
Protocole HELLO

Connaissance du voisinnage : protocole HELLO



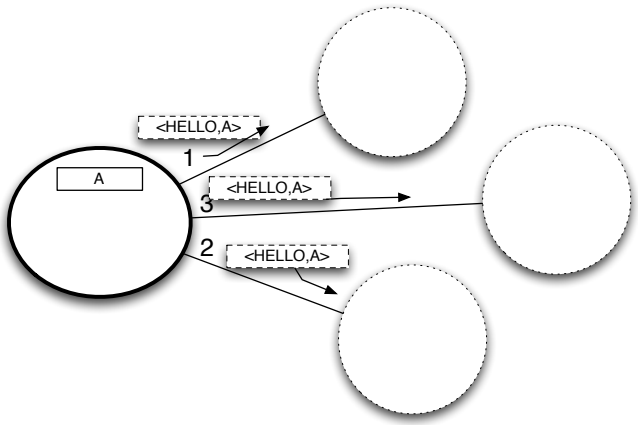
- Lors de processus de la connaissance du voisinage sur le site i :

Connaissance du voisinnage : protocole HELLO



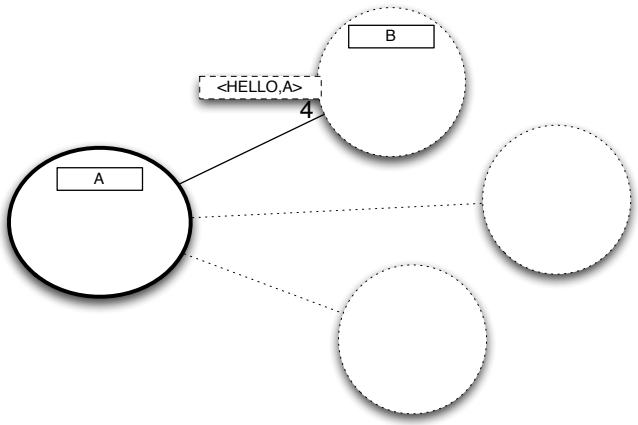
- Lors de processus de la connaissance du voisinage sur le site i :

Connaissance du voisinnage : protocole HELLO



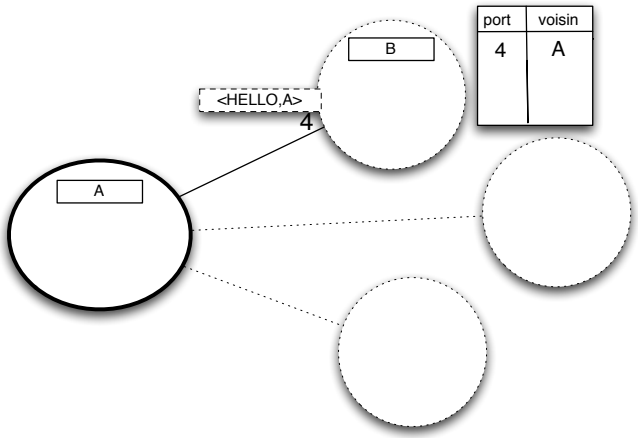
- Lors de processus de la connaissance du voisinage sur le site i :
  - envoyer le message `<HELLO,i>`

Connaissance du voisinnage : protocole HELLO



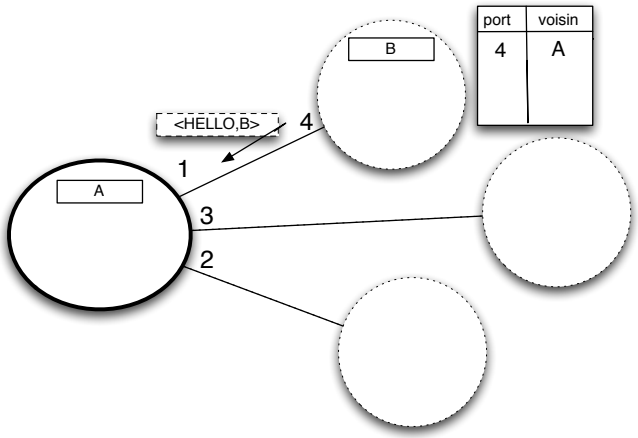
- Lors de processus de la connaissance du voisinage sur le site i :
  - envoyer le message `<HELLO,i>`
- Lors de la récéption d'un message `<HELLO,id>` sur le port ℓ

Connaissance du voisinnage : protocole HELLO



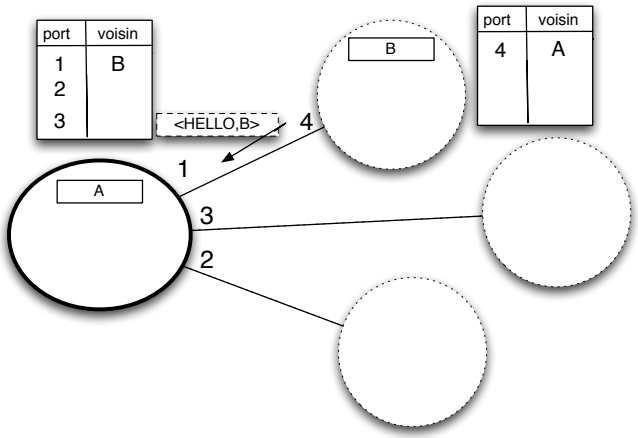
- Lors de processus de la connaissance du voisinage sur le site i :
  - envoyer le message `<HELLO,i>`
- Lors de la récéption d'un message `<HELLO,id>` sur le port ℓ
  - Réactualisation de la table voisinnage

Connaissance du voisinnage : protocole HELLO



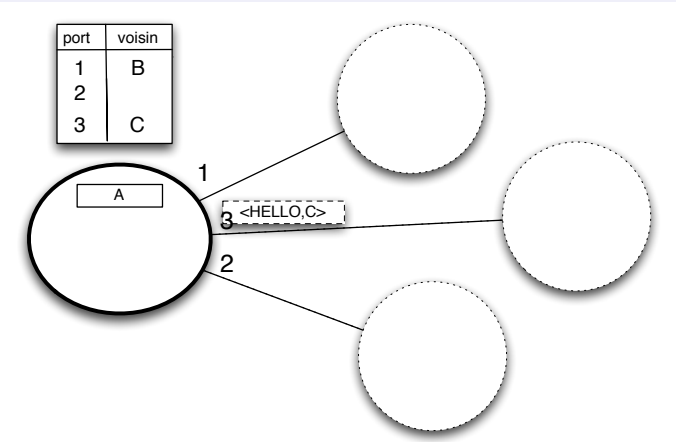
- Lors de processus de la connaissance du voisinage sur le site i :
  - envoyer le message <HELLO,i>
- Lors de la récéption d'un message <HELLO,id> sur le port ℓ
  - Réactualisation de la table voisinnage
  - Envoyer le message <HELLO,i>

Connaissance du voisinnage : protocole HELLO



- Lors de processus de la connaissance du voisinage sur le site i :
  - envoyer le message <HELLO,i>
- Lors de la récéption d'un message <HELLO,id> sur le port ℓ
  - Réactualisation de la table voisinnage
  - Envoyer le message <HELLO,i>

Connaissance du voisinnage : protocole HELLO



Chaque noeud envoie périodiquement sur chaque lien un message HELLO contenant

- son identifiant
- une liste de noeuds (voisins) dont le noeud a reçu un message HELLO

Objectif :

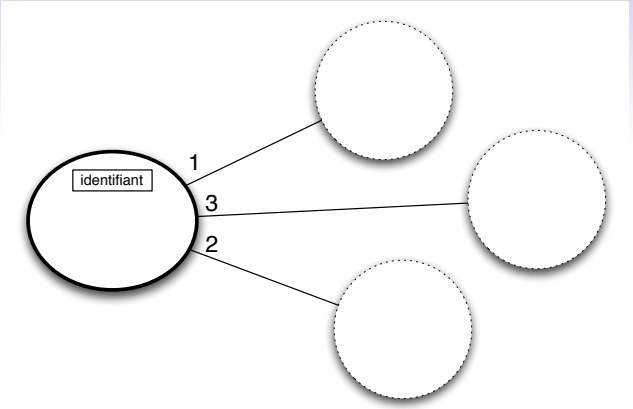
- connaître tous ses voisins
- vérifier l'état du lien de communication

Conséquence : utilisation des connexions *full-duplex*

Plus précisément

- Algorithme distribué
  - Pourquoi rendre le problème distribué ?
  - Comment le rendre distribué ?
  - Quel type d'algorithmes ?
    - Connaissance du voisinnage
    - Diffusion du voisinnage
    - Gestion de la panne

Etat d'un site



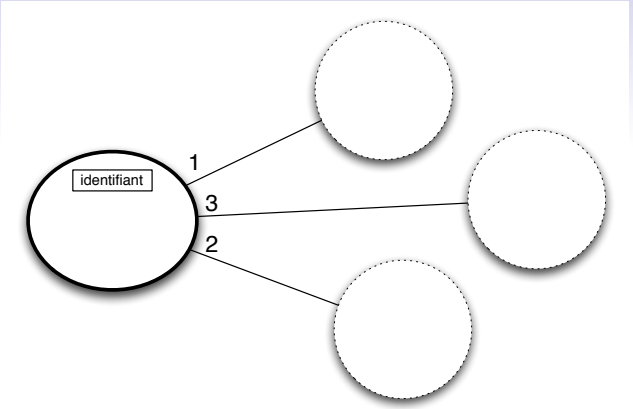
1. Au départ,

   Connaissance du site : son identifiant + les ports de sortie

2. Lancement du processus de connaissance du voisinnage

   Protocole HELLO

Etat d'un site



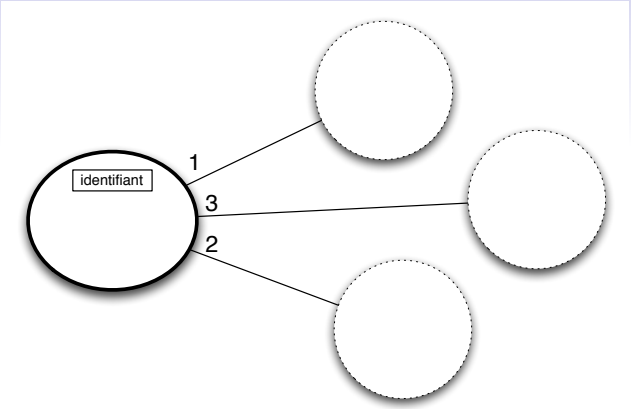
1. Au départ,

   Connaissance du site : son identifiant + les ports de sortie

2. Lancement du processus de connaissance du voisinnage

   Protocole HELLO

Etat d'un site



1. Au départ,

   Connaissance du site : son identifiant + les ports de sortie

2. Lancement du processus de connaissance du voisinnage

   Protocole HELLO

3. Après la connaissance de son voisinnage,

   Faire une diffusion de son voisinnage au reste du système

   protocole d'innondation

Diffusion de son voisinnage au reste du système

Objectif : Informer à tout le reste du réseau,
Par un protocole INONDATION (*Flooding* en anglais)

Lors d'une réception de message <ARETE, A,B,lien ℓ, cout> :

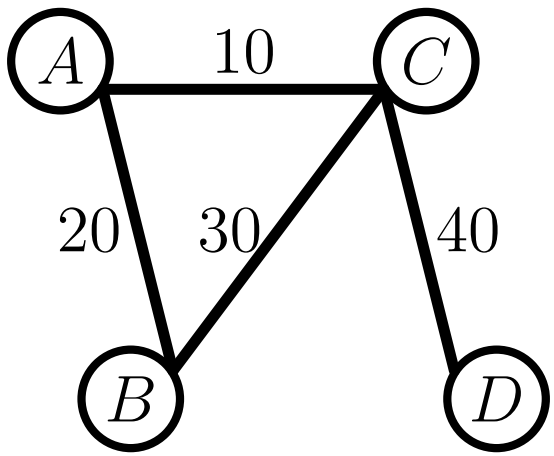
1. Regarder si l'information est dans la base.
2. Si ce n'est pas le cas, alors

   stoquer dans la base puis diffuser cette information

Lors de la diffusion du voisinage de C,

- il envoie à tous ces voisins les messages suivants :

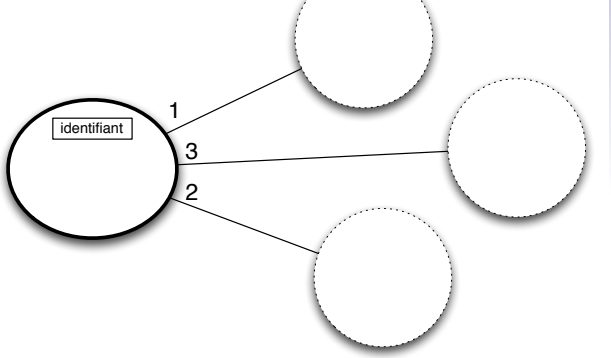
  <ARETE, C,A,lien 10, cout=1>
  <ARETE, C,D,lien 40, cout=1>
  <ARETE, C,B,lien 30, cout=1>



Plus précisément

- Algorithme distribué
  - Pourquoi rendre le problème distribué ?
  - Comment le rendre distribué ?
  - Quel type d'algorithmes ?
    - Connaissance du voisinnage
    - Diffusion du voisinnage
    - Gestion de la panne

Etat d'un site



1. Au départ,

   Connaissance du site : son identifiant + les ports de sortie

2. Lancement du processus de connaissance du voisinnage

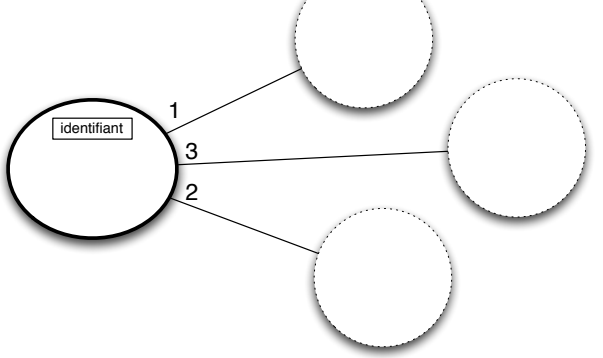
   Protocole HELLO

3. Après la connaissance de son voisinnage,

   Faire une diffusion de son voisinnage au reste du système

   protocole d'innondation

Etat d'un site



1. Au départ,

   Connaissance du site : son identifiant + les ports de sortie

2. Lancement du processus de connaissance du voisinnage

   Protocole HELLO

   - Remarque : le protocole Hello permet periodiquement afin de détecter les pannes et les nouveaux voisins.

3. Après la connaissance de son voisinnage,

   Ou après la détection d'un changement dans le voisinnage

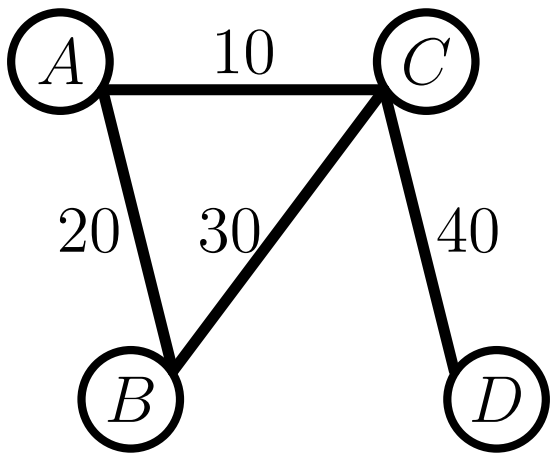
   Faire une diffusion de son voisinnage au reste du système

   protocole d'innondation

Lors d'une panne

Si le lien 30 tombe en panne, après la détection, C alertera ses voisins par l'envoi du msg <C,B,30, cout $= \infty >$

Alors, un noeud peut recevoir les messages

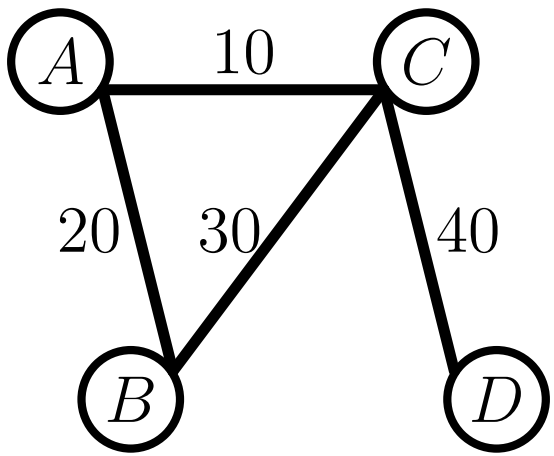


Lors d'une panne

Si le lien 30 tombe en panne, après la détection, C alertera ses voisins par l'envoi du msg <C,B,30, cout $= \infty >$

Alors, un noeud peut recevoir les messages

- dans cet ordre
  1. <ARETE, C, B, lien 30, cout=1>
  2. <ARETE, C, B, lien 30, cout $= \infty >$

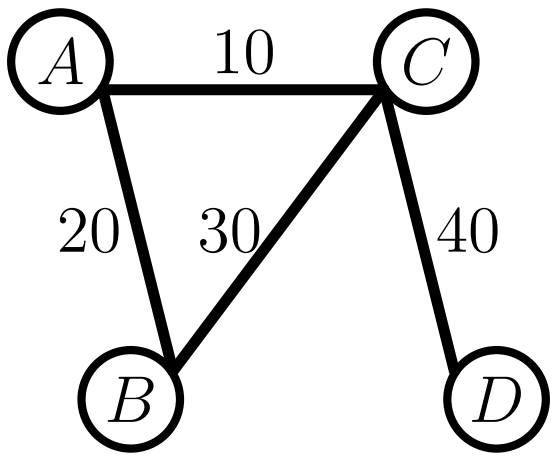


Lors d'une panne

Si le lien 30 tombe en panne, après la détection, C alertera ses voisins par l'envoi du msg <C,B,30, cout $= \infty >$

Alors, un noeud peut recevoir les messages

- dans cet ordre
  1. <ARETE, C, B, lien 30, cout=1>
  2. <ARETE, C, B, lien 30, cout $= \infty >$
- ou dans cet ordre
  1. <ARETE, C, B, lien 30, cout $= \infty >$
  2. <ARETE, C, B,lien 30, cout$= 1 >$

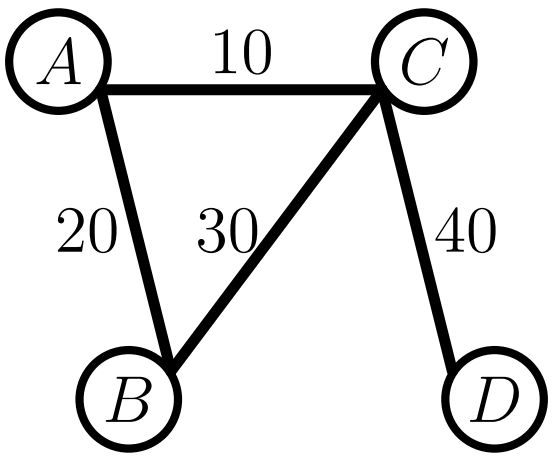


Lors d'une panne

Si le lien 30 tombe en panne, après la détection, C alertera ses voisins par l'envoi du msg <C,B,30, cout $= \infty >$

Alors, un noeud peut recevoir les messages

- dans cet ordre
  1. <ARETE, C, B, lien 30, cout=1>
  2. <ARETE, C, B, lien 30, cout $= \infty >$
- ou dans cet ordre
  1. <ARETE, C, B, lien 30, cout $= \infty >$
  2. <ARETE, C, B,lien 30, cout$= 1 >$



ATTENTION : il risque d'avoir des vieux messages transitant dans le réseau

Nécessité de datation (sur 32 bits)

Protocole d'INNODATION

Lors d'une réception de message <ARETE, A, B, lien ℓ, coût, nbr> :

1. Regarder si l'information est dans la base.
2. Si ce n'est pas le cas, alors

   stoquer dans la base puis diffuser cette information

3. Sinon, si le nombre dans la base $<$ que celui du message, alors

   réactualiser la base, puis diffuser cette information

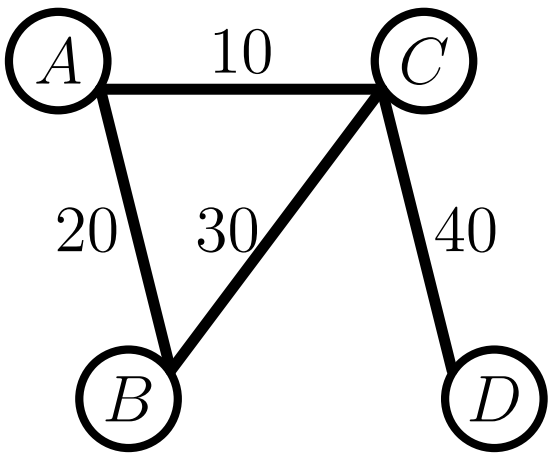
4. Sinon, si le nombre dans la base $>$ que celui du message, alors

   renvoyer l'information stoquée dans la base au destinataire du message.

Lors d'une panne

Avant la panne du lien 30

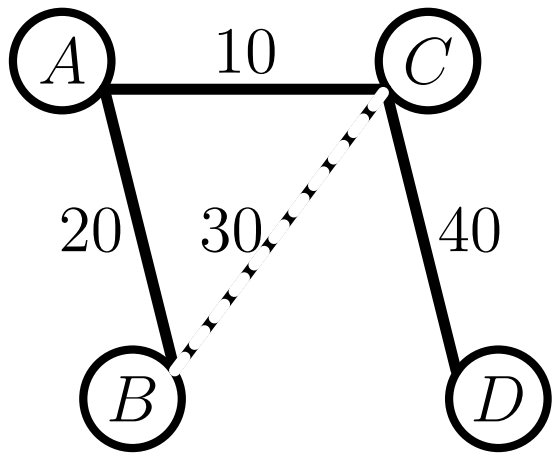
De	Vers	Liens	Coût	Nbr
A	B	20	1	1
A	C	10	1	1
B	A	20	1	1
B	C	30	1	1
C	A	10	1	1
C	B	30	1	1
C	D	40	1	1
D	C	40	1	1



Lors d'une panne

Après la panne du lien 30

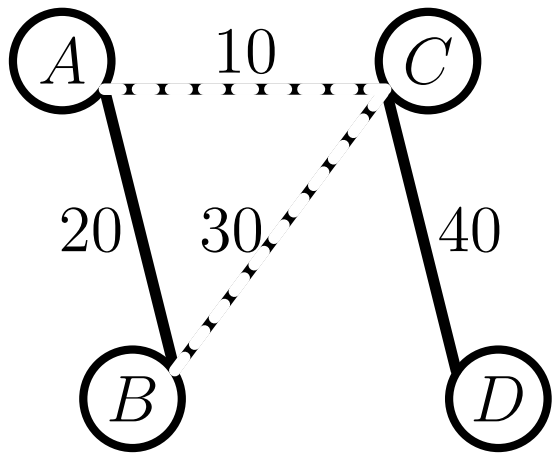
De	Vers	Liens	Coût	Nbr
A	B	20	1	1
A	C	10	1	1
B	A	20	1	1
B	C	30	∞	2
C	A	10	1	1
C	B	30	∞	2
C	D	40	1	1
D	C	40	1	1



Lors d'une panne provoquant une séparation

Après la panne du lien 10

De	Vers	Liens	Distance	Nbr
A	B	20	1	1
A	C	10	∞	2
B	A	20	1	1
B	C	30	∞	2
C	A	10	∞	2
C	B	30	∞	2
C	D	40	1	1
D	C	40	1	1



Les deux réseaux distincts évoluent séparément.

Que se passe-t-il lors d'une nouvelle fusion ?

Lors de la fusion de 2 réseaux

Lors d'une fusion : rétablissement d'un lien connectant (A-C)

- Synchronisation des deux bases par l'échange complète des deux bases

  sur chaque information : conservation de l'information la plus récente.
  insérer les nouvelles informations

- Diffusion des nouvelles informations ou des modifications par le protocole d'inondation.

Récapitulatif : comportement d'un noeud i

Récapitulatif : comportement d'un noeud *i*

1. Lors de son insertion dans le réseau
   - 1.1 envoyer à ses voisins des messages Hello
   - 1.2 Réception des messages Hello de ses voisins
     Apprentissage de son voisinage
   - 1.3 envoyer son voisinnage aux autres sommets

Récapitulatif : comportement d'un noeud *i*

1. Lors de son insertion dans le réseau
   - 1.1 envoyer à ses voisins des messages Hello
   - 1.2 Réception des messages Hello de ses voisins
     Apprentissage de son voisinage
   - 1.3 envoyer son voisinnage aux autres sommets

2. Lors de la détection d'une panne d'un lien voisin
   - 2.1 envoyer le message que le lien est en panne aux autres sommets

Récapitulatif : comportement d'un noeud *i*

1. Lors de son insertion dans le réseau
   - 1.1 envoyer à ses voisins des messages Hello
   - 1.2 Réception des messages Hello de ses voisins
     Apprentissage de son voisinage
   - 1.3 envoyer son voisinnage aux autres sommets

2. Lors de la détection d'une panne d'un lien voisin
   - 2.1 envoyer le message que le lien est en panne aux autres sommets

3. Lors de l'insertion d'un nouveau voisin
   - 3.1 Vérifier si ce n'est pas une fusion des réseaux
   - 3.2 Si oui,
     - 3.2.1 échanger les bases + actualiser les bases
     - 3.2.2 envoyer les nouvelles informations et les modifications aux autres sommets
   - 3.3 Sinon envoyer l'information sur la création d'une arête aux autres sommets

Récapitulatif : comportement d'un noeud *i*

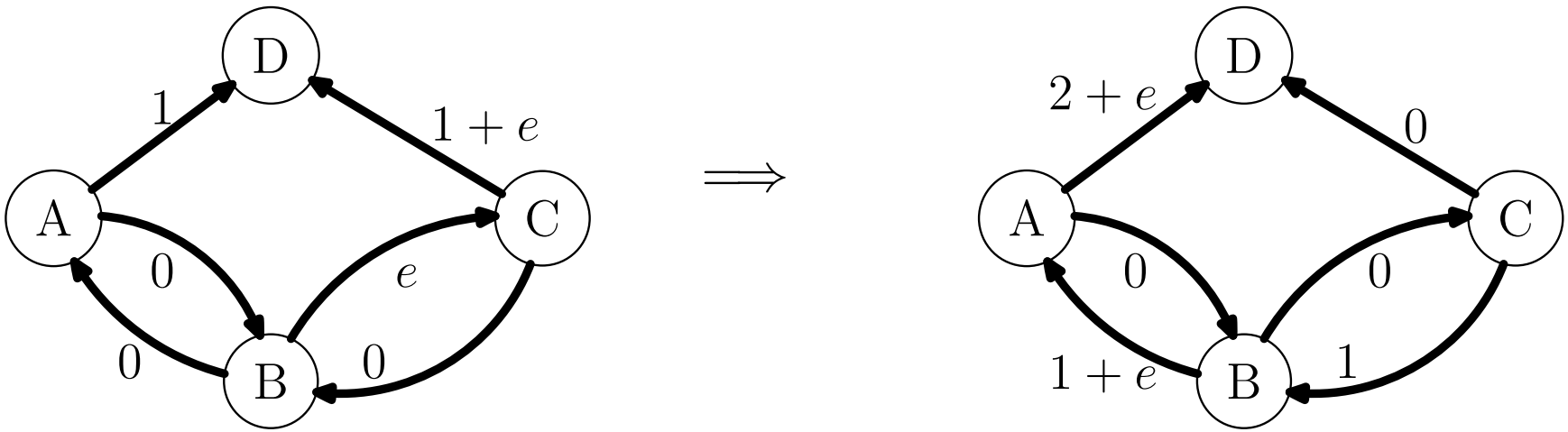
1. Lors de son insertion dans le réseau
   - 1.1 envoyer à ses voisins des messages Hello
   - 1.2 Réception des messages Hello de ses voisins
     Apprentissage de son voisinage
   - 1.3 envoyer son voisinnage aux autres sommets
2. Lors de la détection d'une panne d'un lien voisin
   - 2.1 envoyer le message que le lien est en panne aux autres sommets
3. Lors de l'insertion d'un nouveau voisin
   - 3.1 Vérifier si ce n'est pas une fusion des réseaux
   - 3.2 Si oui,
     - 3.2.1 échanger les bases + actualiser les bases
     - 3.2.2 envoyer les nouvelles informations et les modifications aux autres sommets
   - 3.3 Sinon envoyer l'information sur la création d'une arête aux autres sommets
4. Lors de la modification de la base
   - 4.1 Recalculer la table de routage en fonction de différentes métriques

Différentes métriques

- le nombre de liens
- le débit des liens
- le coût des liens
- la fiabilité des liens
- le temps de transmission des liens
- ....

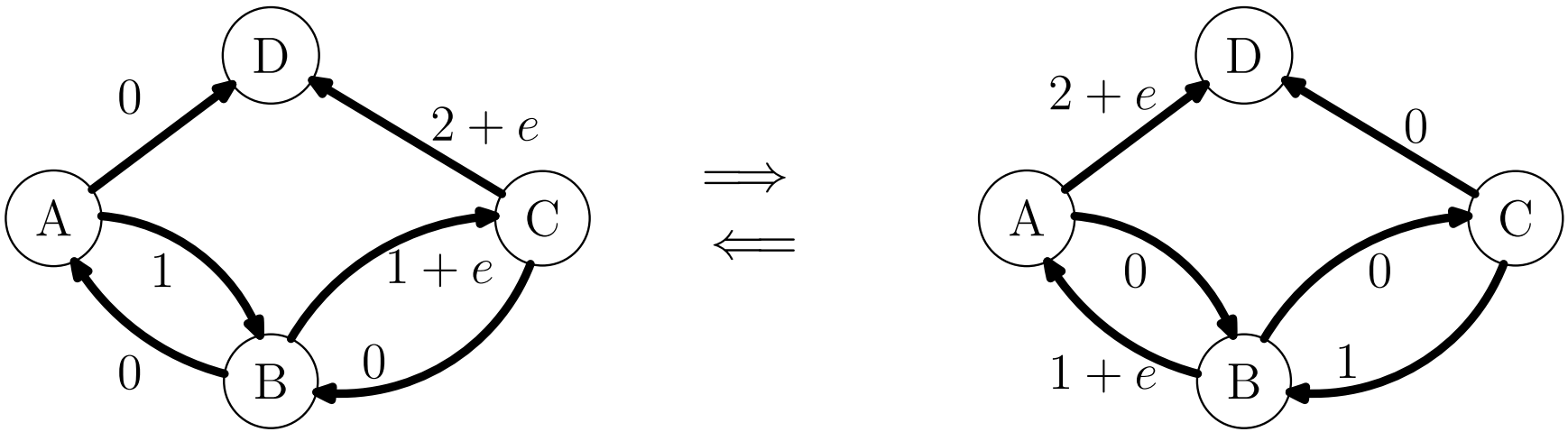
Risque d'oscillation.

Scénario : {
- métrique considérée : la charge du trafic
- trois communications : $B \to D$: e unité
- $A \to D, C \to D$: 1 unité



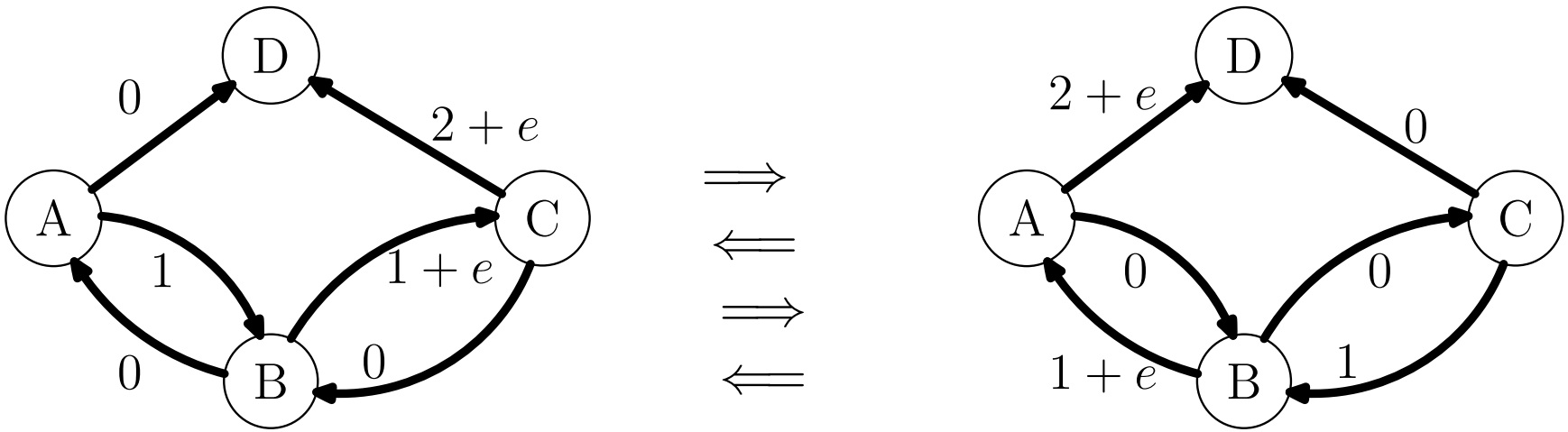
Risque d'oscillation.

Scénario : $\begin{cases} \text{métrique considérée : la charge du trafic} \\ \text{trois communications : } B \to D \text{ : e unité} \\ A \to D, C \to D \text{ : 1 unité} \end{cases}$



Risque d'oscillation.

Scénario : $\begin{cases} \text{métrique considérée : la charge du trafic} \\ \text{trois communications : } B \to D \text{ : e unité} \\ A \to D, C \to D \text{ : 1 unité} \end{cases}$



Instabilité des routes ! ! ! !