

# Algorithmique Distribuée Avancée

## Algorithmes de vagues

Laurent PHILIPPE

Master 2 Informatique  
Université de Franche-Comté  
UFR des Sciences et Techniques

2018/2019

# Les vagues

## Principe

- Un algorithme de vagues sert à prendre une décision après avoir parcouru l'ensemble des processus
- Il s'applique (s'intègre) à un calcul distribué

## Généralités

- Cas d'utilisation possibles :
  - Algorithmes de terminaison
  - Diffusion/Collecte d'information
  - Démarrage de programmes distribués
  - Algorithmes de synchronisation
- Souvent dépendant de la topologie du réseau.

# Les vagues

## Modèle de calcul

- Le calcul distribué est noté  $C$
- Le calcul est un ensemble d'évènements, partiellement ordonné par une relation de causalité  $\preceq$
- Le nombre d'évènements du calcul  $C$  est  $|C|$
- Les évènements du processus  $p$  sont notés  $C_p$
- Pour chaque processus, il existe un évènement interne appelé *decide*

# Les vagues

## Définition

Un **algorithme de vagues** échange un nombre fini de messages pour prendre une décision qui dépend causalement d'un évènement dans chaque processus.

## Propriétés

Un algorithme de vagues satisfait les trois propriétés suivantes :

- 1 **Terminaison** : chaque calcul est fini :  $\forall C : |C| < \infty$
- 2 **Décision** : chaque calcul contient au moins un évènement *decide* :  
 $\forall C : \exists e \in C : e$  est un évènement de décision
- 3 **Dépendance** : dans chaque calcul chaque évènement *decide* est précédé par un évènement dans chaque processus :  $\forall C : \forall e \in C : (e \text{ est un évènement } decide \Rightarrow \forall q \in P \exists f \in C_q : f \preceq e)$

# Les vagues

## Définitions

- Un calcul dans un algorithme de vagues = une **vague**
- Une vague = flot de contrôle qui passe par tous les processus d'un calcul et a pour but de prendre une décision
- Lancée à l'initiative d'un initiateur  $\alpha$ , déclenchée par une condition (évènement local), elle retourne à l'initiateur
- Les processus non-initiateurs (**suiveurs**) sont impliqués dans l'algorithme par la réception d'un message qui déclenche l'exécution de l'algorithme
- On appelle **train de vagues** une succession de vagues.

# Les vagues

## Propriétés des algorithmes

- Centralisé vs. Distribué : l'algorithme est démarré par exactement un initiateur ou par plusieurs.
- Topologie, déterminante pour la définition de l'algorithme
- Données initiales :
  - Identité du processus : chq processus connaît son nom unique
  - Identité des voisins : chq processus connaît ses voisins
  - Direction des liens : lorsque que les liens ne sont pas full-duplex ou lorsqu'une direction leur est attribuée dynamiquement
- Nombre des décisions, dans certains algorithmes le nombre de processus exécutant un évènement décide peut varier
- Complexité, en nombre de messages, calculs, taille des données échangées

# Les vagues

## Primitives de vague

Pour caractériser une vague, on définit 4 primitives :

- lancer (pour  $\alpha$ );
- retour (pour  $\alpha$ );
- visite (pour  $\forall i \in \mathbb{P}$  et  $i \neq \alpha$ );
- faire\_suivre (pour  $\forall i \in \mathbb{P}$  et  $i \neq \alpha$ );

## Propriétés

On note  $x_i^p$  l'exécution de la  $p$ -ème vague sur le processus  $i$  alors :  
 $\forall p > 0, \forall i \neq \alpha : \text{lancer}_\alpha^p \rightarrow \text{visite}_i^p \rightarrow \text{faire\_suivre}_i^p \rightarrow \text{retour}_\alpha^p$   
et  $\forall p > 0 : \text{retour}_\alpha^p \rightarrow \text{lancer}_\alpha^{p+1}$

## Algorithme générique d'une vague

Soit une vague avec  $v$  valeur initiale et  $v'$  résultat après la vague

Pour  $c_\alpha$  :

**début**

**répéter**

        calcul locaux ...

$v \leftarrow \dots$

        lancer( $v$ )

        retour( $v'$ )

**jusqu'à** *une condition sur  $v'$*

visite( $v$ ) : pour  $c_i$  avec  $i \neq \alpha$

**début**

    calcul locaux ...

$v' \leftarrow \dots$

    faire\_suivre( $v'$ )



# Sommaire

- 1 Algorithmes de vagues
  - Vagues sur un réseau complet
  - Vagues sur un anneau
  - Vagues sur un arbre
  - Vagues sur un réseau quelconque
- 2 Algorithme de traversée
- 3 Résultats élémentaires

# Les vagues

## Hypothèses

- La topologie réseau est :
  - Fixe, aucun changement pendant l'algorithme
  - Orientation : échanges bidirectionnels ou non
  - Connectée, il y a un chemin entre chaque paires de processus
- Les échanges de messages sont asynchrones, fiables
- Pas de données, ni d'horloge partagées

# Une vague sur un réseau complet

## Algorithme

- Simple : consultation (*polling*)
- Envoie un message à chaque voisin
- Attend réponse à chaque message
- Décide après la réception de tous les messages

# Une vague sur un réseau complet

## Exercice

Écrire l'algorithme de polling

# Une vague sur un réseau complet

## Algorithme avec les 4 primitives

**var**  $rec_\alpha$  : entier **init** 0 (\* Seulement pour l'initiateur \*)

lancer( $v$ ) : pour  $c_\alpha$

**début**

$v \leftarrow \dots$

**for**  $q \in Neigh_\alpha$  **do**

└ envoi jeton( $v$ ) à  $q$

**while**  $rec_\alpha < \#Neigh_\alpha$  **do**

└ recevoir(jeton( $v'$ ))

└  $rec_\alpha = rec_\alpha + 1$

└ decide( $v'$ )

visite( $v$ ) : pour  $c_i$  avec  $i \neq \alpha$

**début**

└ recevoir le jeton( $v$ ) de  $\alpha$

$v' \leftarrow \dots$

└ faire\_suivre( $v'$ ) : envoyer  
le jeton( $v'$ ) à  $\alpha$

# Une vague sur un anneau

## Principe

- Anneau unidirectionnel : tous les processus  $i$  connaissent leur suivant  $succ_i$
- La visite des différents processus se fait de manière séquentielle
- La vague est matérialisée par le passage d'un jeton
- L'algorithme est centralisé

# Une vague sur un anneau

## Exercice

Écrire l'algorithme de l'anneau

# Une vague sur un anneau

## Algorithme avec les 4 primitives

lancer( $v$ ) : pour  $c_\alpha$

**début**

┌  $v \leftarrow \dots$   
└ envoyer le jeton( $v$ ) à  $succ_\alpha$

retour( $v$ ) : pour  $c_\alpha$

**début**

┌ recevoir le jeton( $v'$ )  
└ decide( $v$ )

visite( $v$ ) : pour  $c_i$  avec  $i \neq \alpha$ ;

**début**

┌ recevoir le jeton( $v$ ) ;

faire\_suivre( $v'$ ) : pour  $c_i$  avec  
 $i \neq \alpha$ ;

**début**

┌  $v' \leftarrow \dots$  ;  
└ envoyer le jeton( $v'$ ) à  $succ_i$ ;



# Une vague sur un anneau

## Preuve que l'algorithme sur un anneau est un algorithme de vague

Soit  $p_0$  l'initiateur, chaque processus envoie au plus 1 message donc, au total, l'algorithme échange au plus  $N$  messages et le calcul est **fini**.

Une configuration terminale est atteinte en un nombre fini d'étapes. Dans cette configuration :

- $p_0$  a déjà envoyé le jeton, il a dépassé l'instruction *envoyer* dans son programme.
- Aucun jeton n'est en transit sinon il pourrait être reçu et la configuration ne serait pas terminale
- Aucun processus ne possède le jeton sinon il n'aurait pas franchi l'instruction *envoyer*
- De ce fait  $p_0$  a reçu le jeton et, puisque nous sommes en configuration terminale, il a exécuté l'instruction *decide* et la **décision** est vérifiée

La réception et l'envoi du jeton pour les processus  $p \neq p_0$  précèdent la réception de  $p_0$  donc la condition de **dépendance** est vérifiée.

# Une vague sur un arbre

## Algorithme

- La racine de l'arbre est  $c_\alpha$ ,
- Les messages circulent dans les deux sens,
- Un processus agrège les informations reçues à l'aide de la fonction  $V' = \text{agregé}(V)$ , où  $V$  est l'ancienne valeur et  $V'$  la nouvelle,
- Un processus remonte l'information vers son père si il a reçu tous les retours de tous ses fils,
- Exercice : écrire l'algorithme de vague pour les arbres dans le cas où l'initiateur est la racine et dans le cas où l'algorithme est initié par toutes les feuilles

# Une vague sur un arbre

Algorithme initié par la racine

lancer( $v$ ) : pour  $c_\alpha$

**début**

$v \leftarrow \dots$   
 $\forall x \in \text{fils}_\alpha$  : envoyer aller( $v$ ) à  $c_x$

retour( $V'$ ) : pour  $c_\alpha$

**début**

$\forall x \in \text{fils}_\alpha$  recevoir( $V'$ ) de  $c_x$   
 decide( $V'$ )

visite( $v$ ) : pour  $c_i$  avec  $i \neq \alpha$

**début**

recevoir aller( $v$ ) de  $c_{\text{pere}_i}$

**si**  $|\text{fils}_i| \neq 0$  **alors**

$\forall x \in \text{fils}_i$  : envoyer aller( $v$ ) à  $c_x$

**sinon**

$v' \leftarrow \dots$

envoyer retour( $v'$ ) à  $\text{pere}_i$

faire\_suivre( $v'$ ) : (pour  $i \neq \alpha$ )

**début**

$\forall x \in \text{fils}_i$  recevoir valeur  $w$  de  $c_x$

$v' \leftarrow \text{agregation}(v', w)$

envoyer retour( $v'$ ) à  $\text{pere}_i$

Note : la valeur  $v$  transmise en montée de l'arbre peut être changée par le processus.

## Une vague sur un arbre

Algorithme d'arbre initié par (toutes) les feuilles :

**var**  $req_p[q], \forall q \in Neigh_p$  : booléen **init** faux  
 (\*  $req_p[q]$  est vrai si  $p$  a reçu un message de  $q$  \*)

*visite*( $V$ ) :

**début**

**tant que**  $\#(q : req_p[q] = faux) > 1$  **faire**

└─ reçoit  $\langle tok \rangle$  de  $q$ ;  $rec_p[q] = vrai$

(\* il n'y a plus qu'un  $q_0$  avec  $req_p[q_0]$  est faux \*)

envoie  $\langle tok \rangle$  à  $q_0$  tel que  $rec_q[q_0]$  est faux

reçoit  $\langle tok \rangle$  de  $q_0$ ;  $rec_p[q_0] = vrai$

**si**  $p$  est initiateur **alors** décide

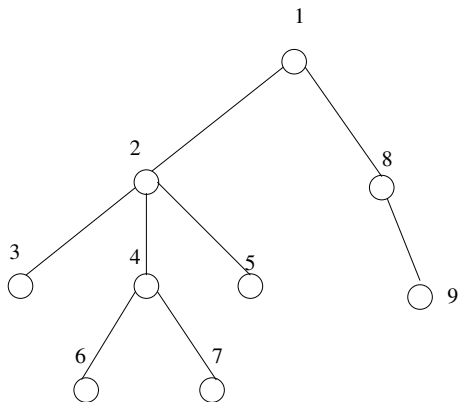
(\* Retour de la décision aux autres processus initiateurs \*)

**pour**  $q \in Neigh_p, q \neq q_0$  **faire** envoie  $\langle tok \rangle$  à  $q$

# Une vague sur un arbre

## Exercice

Dérouler l'algorithme des feuilles avec l'arbre suivant



# Algorithmes de vagues sur un réseau quelconque

## Principe

- Le réseau est un graphe
- Les processus sont connectés entre eux de manière quelconque
- Chaque processus connaît uniquement ses voisins

## Caractéristique

Propriétés du réseau influent sur les performances de l'algorithme

# Algorithme Echo

## Exercice

- L'algorithme est centralisé (lancé par un seul initiateur),
- L'initiateur diffuse des messages  $\langle tok \rangle$ ,
- A la réception du premier message, le père du processus est défini,
- Chaque processus compte les messages reçus,
- Lorsqu'il a reçu autant de messages que de voisins, il renvoie le message  $\langle tok \rangle$  à son père,
- Écrire l'algorithme,

# Algorithme Echo

```
var  $rec_p$  : entier init 0 (* Compte le  
nombre de messages reçus *)  
     $father_p$  :  $\mathbb{P}$  init udef
```

Initiateur :

**début**

```
┌   pour  $q \in Neigh_p$  faire  
├   │   envoie < tok > à  $q$   
├   tant que  $rec_p < \#Neigh_p$  faire  
├   │   reçoit < tok >  
├   │    $rec_p = rec_p + 1$   
└   decide()
```

Non-initiateur :

**début**

```
┌   reçoit < tok > de  $q$ ;  
├    $father_p = q$ ;  $rec_p = rec_p + 1$ ;  
├   pour  $q \in Neigh_p, q \neq father_p$  faire  
├   │   envoie < tok > à  $q$   
├   tant que  $rec_p < \#Neigh_p$  faire  
├   │   reçoit < tok > de  $q$ ;  
├   │    $rec_p = rec_p + 1$ ;  
└   envoie < tok > à  $father_p$ ;
```



# Algorithmes de vagues sur un réseau quelconque

## Remarque

- Définition implicite d'un arbre de recouvrement.

# L'algorithme Phase

## Généralités

- Sur un réseau quelconque, pour des **liens unidirectionnels** : les voisins de  $p$  sont répartis en deux ensembles :
  - in-voisins de  $p$  : ceux qui peuvent envoyer un message à  $p$
  - out-voisins de  $p$  : ceux qui peuvent recevoir un message de  $p$
- Algorithme décentralisé : un ou plusieurs initiateurs
- Nécessite pour les processus de connaître le diamètre du réseau  $D$
- Mais l'algorithme fonctionne aussi avec une constante  $D' > D$  à condition que tous les processus disposent de la même valeur, il est donc suffisant de connaître une borne supérieure
- Plus efficace si la valeur de  $D$  est exacte

# L'algorithme Phase

## Principe

- Chaque processus envoie exactement  $D$  messages à chaque out-voisin
- Le  $i + 1$  ième message pour chaque out-voisin n'est envoyé qu'après avoir reçu  $i$  messages de chacun des in-voisins

# Algorithme Phase

**cons**  $D$  : entier (\* le diamètre du réseau \*)  
**var**  $rec_p[q]$  :  $0..D$  **init**  $0 \forall q \in In_p$  (\* Nombre de messages reçus de  $q$  \*)  
 $env_p$  :  $0..D$  **init**  $0 \forall q \in Out_p$  (\* Nombre de messages envoyés de  $p$  \*)  
 $In_p$  et  $Out_p$  (\* Voisins de  $p$  \*)

**début**

```

if  $p$  is initiateur then
    pour  $q \in Out_p$  faire
        envoie  $\langle tok \rangle$  à  $q$ 
         $env_p = env_p + 1$ 

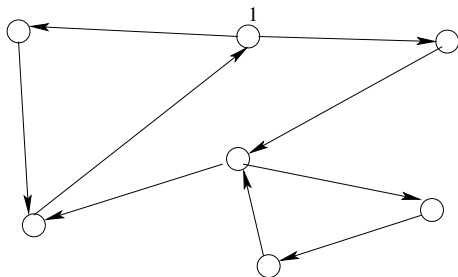
    tant que  $\min_q(rec_p[q]) < D$  faire
        reçoit  $\langle tok \rangle$  de  $q_0$ 
         $rec_p[q_0] = rec_p[q_0] + 1$ 
        if  $(\min_q(rec_p[q]) \geq env_p) \wedge (env_p < D)$  then
            for  $q \in Out_p$  do envoie  $\langle tok \rangle$  à  $q$ 
             $env_p = env_p + 1$ 
    
```

*decide*

# Algorithme Phase

## Exercice

Soit le graphe orienté suivant :  
Calculer le diamètre du graphe

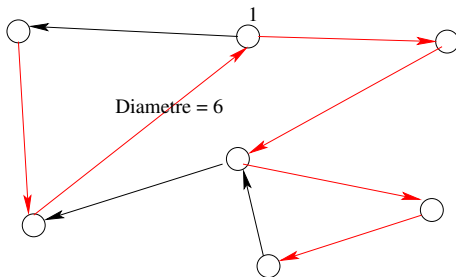


# Algorithme Phase

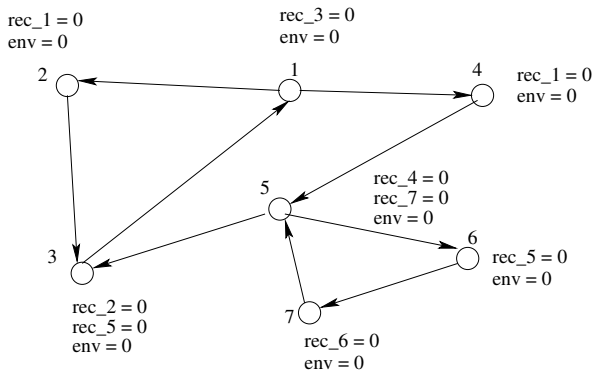
## Exercice

Soit le graphe orienté suivant :

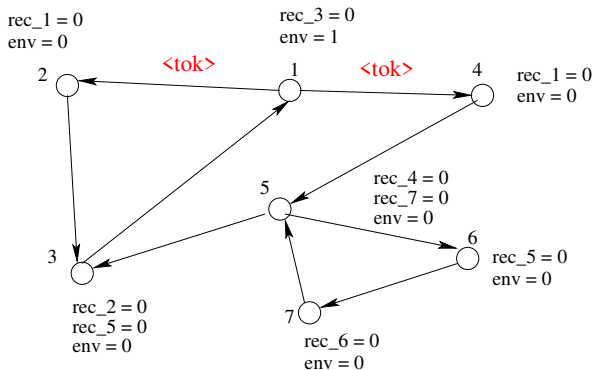
Dérouler l'algorithme *Phase* avec le processus 1 comme initiateur



# Algorithme Phase

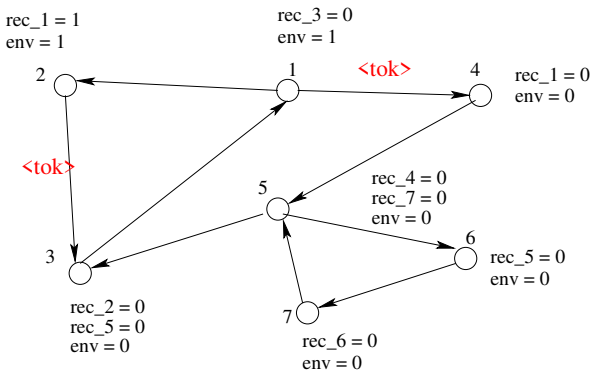


# Algorithme Phase

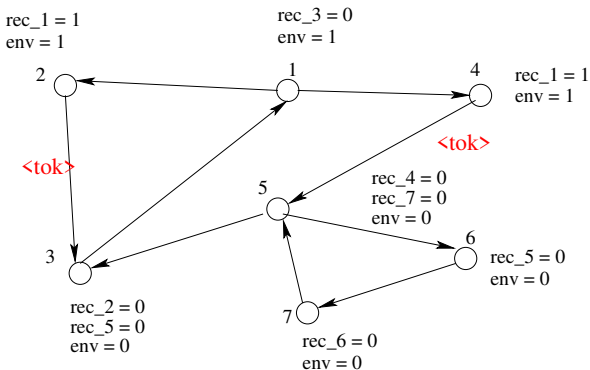




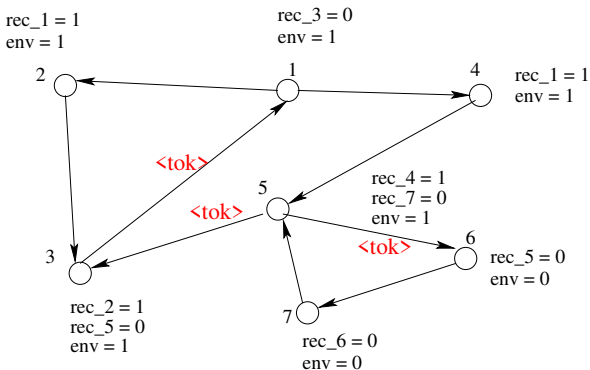
# Algorithme Phase



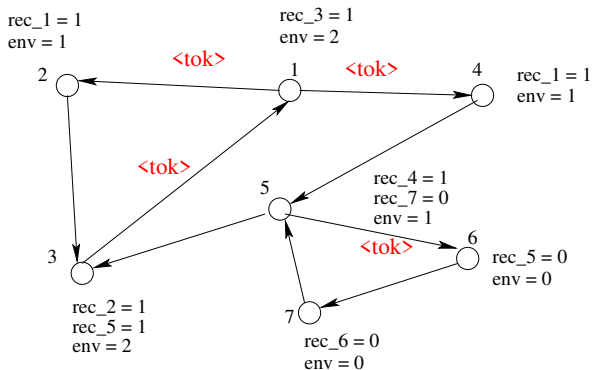
# Algorithme Phase



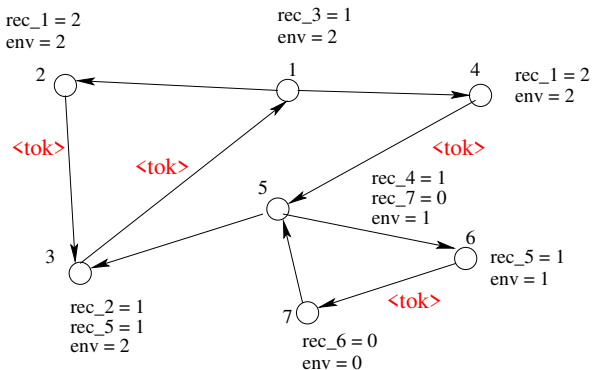
# Algorithme Phase



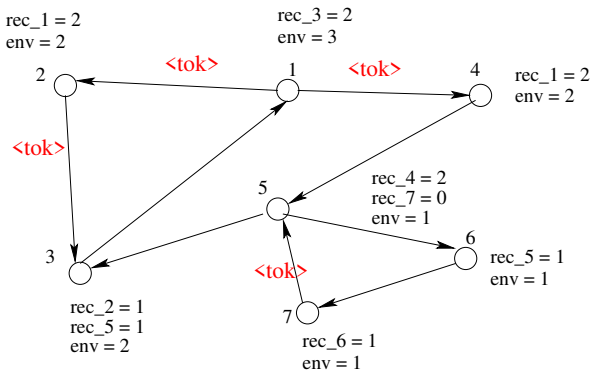
# Algorithme Phase



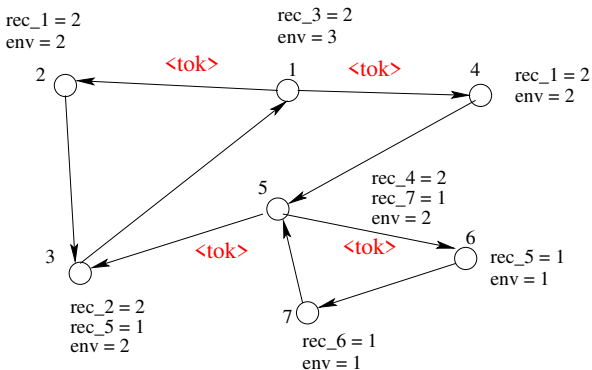
# Algorithme Phase



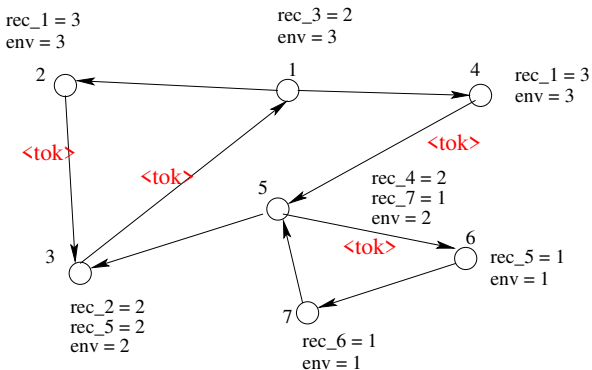
# Algorithme Phase



# Algorithme Phase

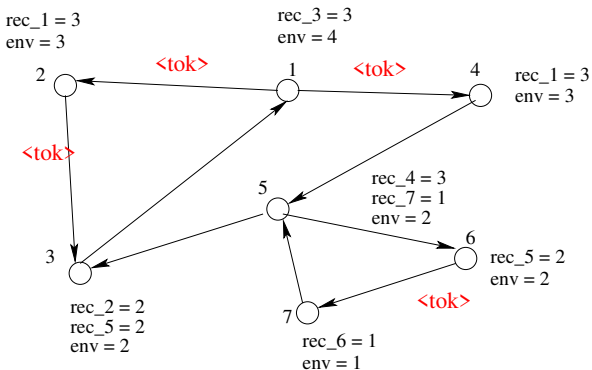


# Algorithme Phase

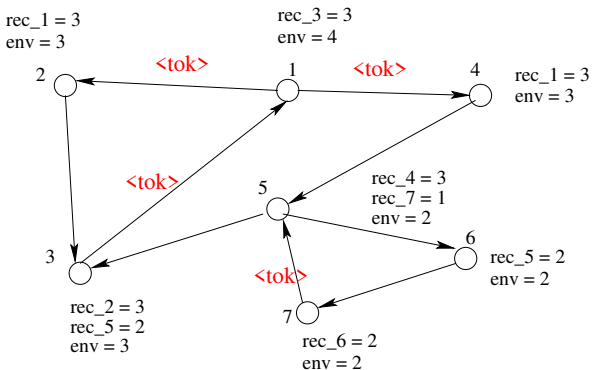




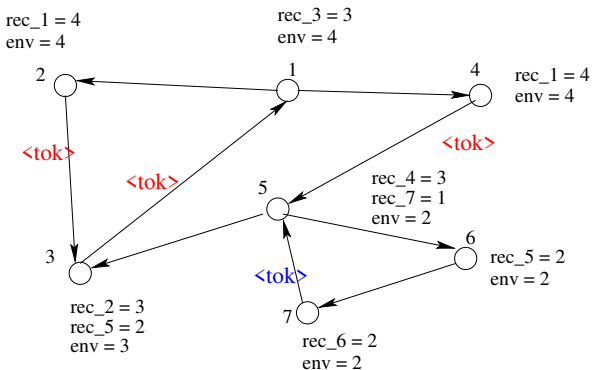
# Algorithme Phase



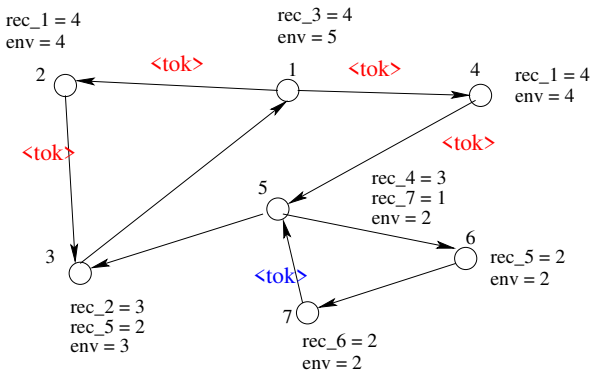
# Algorithme Phase



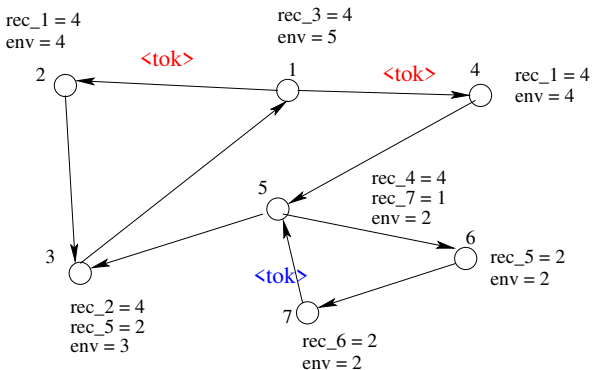
# Algorithme Phase



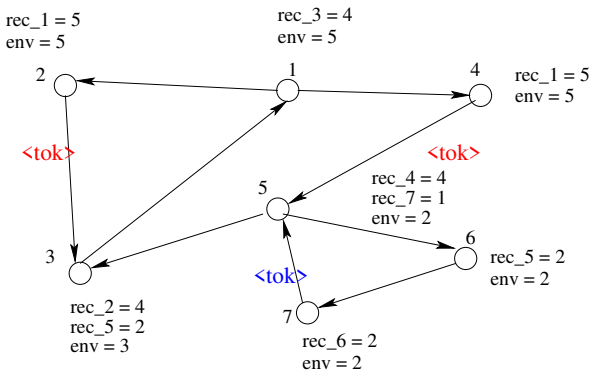
# Algorithme Phase



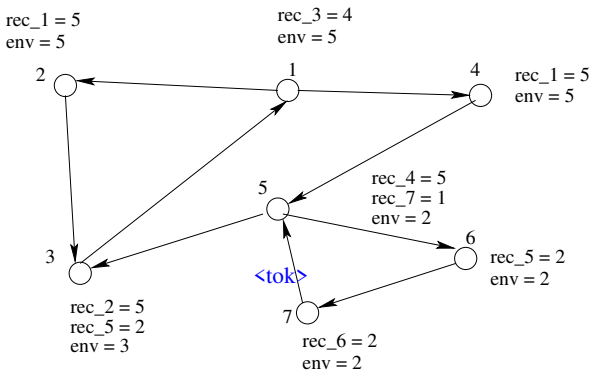
# Algorithme Phase



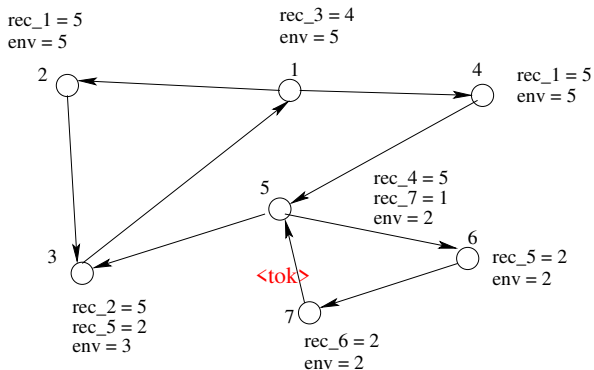
# Algorithme Phase



# Algorithme Phase

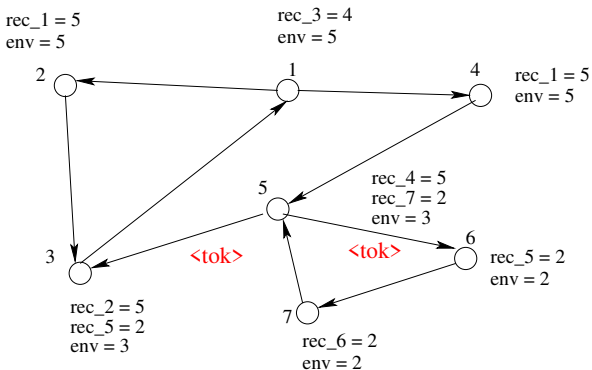


# Algorithme Phase

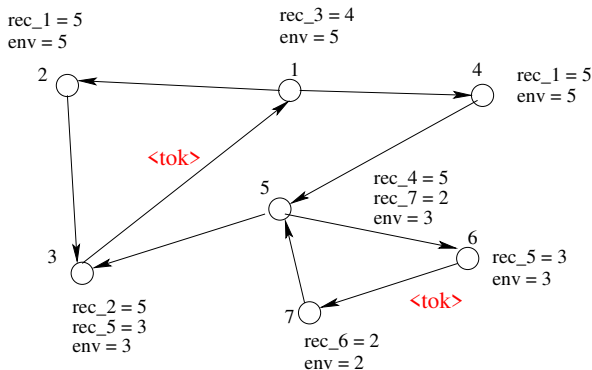




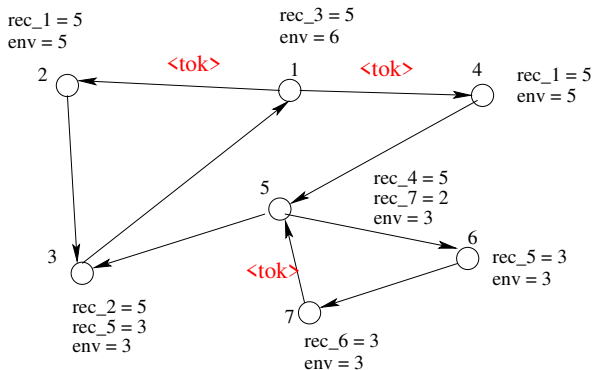
# Algorithme Phase



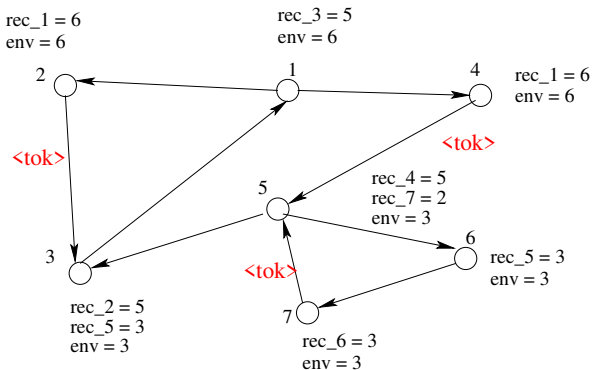
# Algorithme Phase



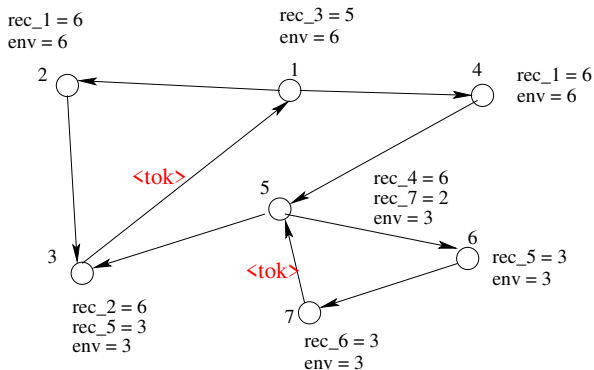
# Algorithme Phase



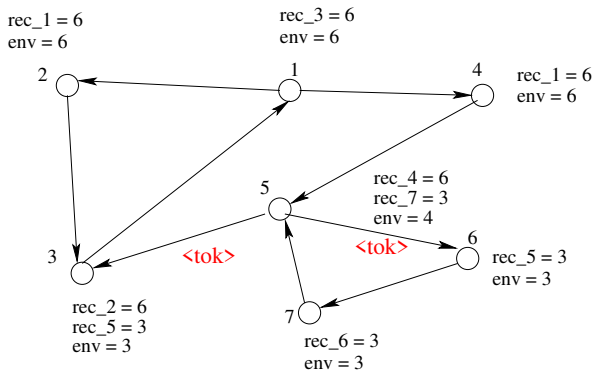
# Algorithme Phase



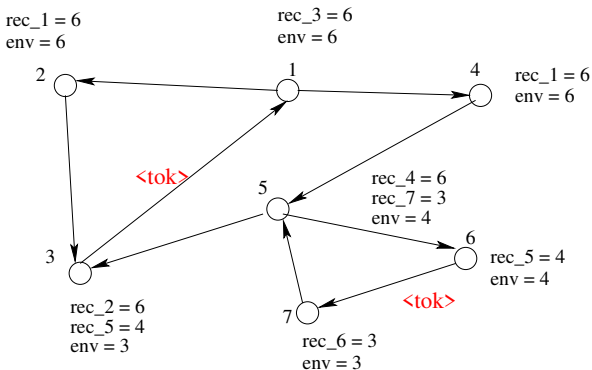
# Algorithme Phase



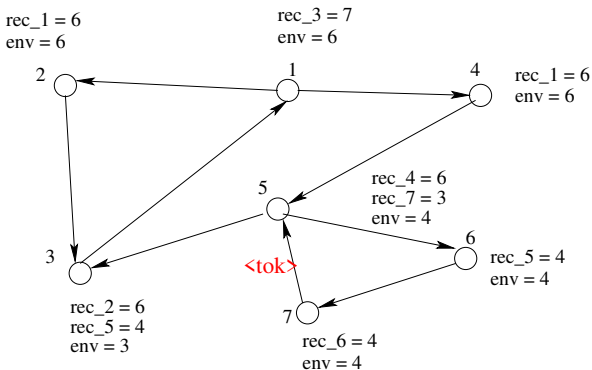
# Algorithme Phase



# Algorithme Phase

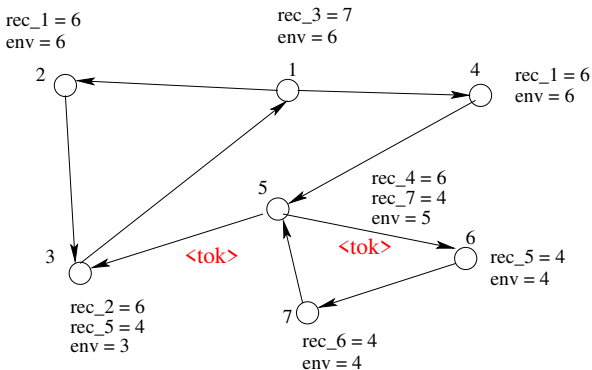


# Algorithme Phase





# Algorithme Phase



# Sommaire

- 1 Algorithmes de vagues
- 2 Algorithme de traversée
- 3 Résultats élémentaires

# Algorithmes de traversée

## Définition

Un algorithme de traversée est un algorithme vérifiant les propriétés suivantes :

- Il y a un seul initiateur qui lance une vague avec exactement 1 message
- Un processus qui reçoit un message, soit envoie un message soit décide
- L'algorithme termine chez l'initiateur
- Lorsque l'algorithme est terminé, chaque processus a au moins envoyé un message

# Algorithmes de traversée ou de parcours

## Propriétés

Un algorithme de traversée (*traversal algorithms*) est un algorithme vérifiant les propriétés suivantes :

- Dans une configuration atteignable d'un algorithme de traversée il y a : soit un message en transit, soit un processus qui a reçu un message et qui n'en a encore pas envoyé.
- En général le message est un jeton qui passe de processus en processus

## Définition

Les algorithmes de parcours (*traversal algorithms*) forment une sous-classe des algorithmes de vagues où :

- Il n'y a qu'un seul initiateur, qui est le seul processus qui décide
- Tous les évènements sont totalement ordonnés par ordre de causalité

# Algorithmes de traversée

## Définition

- Parmi les algorithmes de traversée, la classe des algorithmes  $f$ -traversants est définie par : à chaque vague au moins  $\min(N, x + 1)$  processus ont été visités après  $f(x)$  passages de jeton.
- Par exemple l'algorithme sur l'anneau est un algorithme  $x$ -traversant car  $x + 1$  processus ont traité le jeton après  $x$  étapes et tous les processus l'ont fait après  $N$  étapes.

# Algorithmes de traversée

## Traversée d'un réseau complet

La traversée d'un réseau complet peut-être réalisée par une consultation séquentielle :

- Les messages sont envoyés un par un
- Le voisin suivant n'est consulté que lorsque le voisin courant a répondu au message
- Parcours des voisins un par un

# Algorithmes de traversée

## Théorème

L'algorithme de consultation séquentielle est un algorithme  $2x$ -traversant pour les réseaux complets

### Preuve :

- Quand l'algorithme termine chez l'initiateur, chaque processus a envoyé une réponse.
- Le  $(2x - 1)$  ième message est la consultation de  $q_x$  et le  $2x$  ième message est sa réponse.
- Ainsi quand  $2x$  messages ont été échangés les  $x + 1$  processus  $p, q_1, \dots, q_x$  ont été visités.

# Algorithmes de traversée

## Algorithme de Tarry

- Algorithme de traversée de réseaux quelconques
- Basé sur un jeton
- Il est formulé à partir de deux règles :
  - R1 : un processus ne transmet jamais le jeton deux fois sur le même lien
  - R2 : un processus suiveur transmet le jeton à son père (le voisin dont il a reçu le jeton) seulement si il n'y a plus de lien permettant de respecter R1



# Algorithme de Tarry

**var**  $util_p[q]$  : booléen **init** faux  $\forall q \in Vois_p$  (\* Indique si le voisin a été visité \*)  
 $pere_p$  : processus **init** indef

Pour l'initiateur, exécute  
**début**

```
┌  $pere_p = p$   
├ choisir  $q \in Vois_p$   
├  $util_p[q] = vrai$   
└ envoi( jeton ) à  $q$ 
```

A la réception du jeton en  $p$  depuis  $q_0$

**début**

```
┌ if  $pere_p = indef$  then  $pere_p = q_0$   
├ if  $\forall q \in Vois_p : util_p[q]$  then decide  
├ else  
├ ┌ if  $\exists q \in Vois_p : (q \neq father_p \wedge !util_p[q])$   
├ │ then  
├ │ ┌ choisit  $q \in Vois_p - pere_p / !util_p[q]$   
├ │ │  $util_p[q] = vrai$   
├ │ │ envoi( jeton ) à  $q$   
├ │ else  
├ │ ┌  $util_p[father_p] = vrai$   
├ │ │ envoi( jeton ) à  $pere_p$ 
```

# Sommaire

- 1 Algorithmes de vagues
- 2 Algorithme de traversée
- 3 Résultats élémentaires

## Propriété structurelle des vagues

### Lemme 1

Chaque évènement d'un calcul est précédé par un évènement dans un initiateur :  $\forall e \in C, \exists p, \text{ initiateur}, \text{ et } f \in C_p : f \preceq e$

**Preuve** : par l'absurde

On choisit  $f$ , un élément minimal de l'historique de  $p : f \preceq e$  et  $\nexists f' : f' \preceq f$ . Cet élément existe car l'historique est finie

On en déduit que  $f$  est le premier élément de  $p$ , sinon  $f'$  existerait.

Or le premier évènement d'un suiveur est un évènement de réception qui doit être précédé par un évènement d'émission.

Donc si  $p$  est un suiveur il existe un évènement avant  $f$ , ce qui contredit l'hypothèse  $\Rightarrow p$  est un initiateur

## Arbre de recouvrement pour vague à 1 initiateur

### Lemme 2

Soit  $C$  une vague avec 1 initiateur  $p$  et, pour chaque suiveur  $q$  soit  $father_q$  le voisin par lequel  $q$  reçoit le message du premier évènement. Alors le graphe  $T = (P, E_T)$  avec  $E_T = \{qr : q \neq p \wedge r = father_q\}$  est un arbre de recouvrement dont la racine est  $p$

#### Preuve :

- Connexité : par définition un algorithme de vague parcourt tous les processus, donc chaque processus est bien relié à au moins un autre processus, celui par lequel il a reçu le message.
- Structure d'arbre : comme il y a un noeud de plus dans  $P$  que d'arêtes dans  $E_T$ , il suffit de montrer que  $T$  ne possède pas de cycle. Ce qui est vérifié car, pour  $e_q$  premier évènement de  $q$  ;  $qr \in E_T \Rightarrow e_r \preceq e_q$  et  $\preceq$  est un ordre partiel.

## Un évènement *envoi* précède toute décision

### Lemme 3

Dans la clause *Dépendance* l'évènement  $f$  peut-être un évènement *envoi* pour tous les processus autres que celui où l'évènement *decide* à lieu :  
soit  $C$  une vague et  $d_p \in C$  un évènement *decide* de  $p$  alors  
 $\forall q \neq p : \exists f \in C_q : f \preceq d_p \wedge f$  est un évènement *envoi*

#### **Preuve :**

Comme  $C$  est une vague il existe un  $f \in C_q : f \preceq d_p$ . On choisit  $f$  le dernier évènement de  $C_q$  qui précède  $d_p$ . La définition de la causalité implique qu'il existe une séquence  $f = e_0, \dots, e_k = d_p$  telle que  $\forall i < k : e_i$  et  $e_{i+1}$  sont soit des évènements consécutifs d'un même processus soit un couple *envoi* – *reçoit*. Comme  $f$  est le dernier évènement de  $q$  qui précède  $d_p$  alors  $e_1$  qui suit  $f$  se déroule dans un autre processus que  $q$  et donc  $f$  est un évènement *envoi*

## Borne inférieure sur la complexité des vagues

### Algorithmes à 1 initiateur

Soit  $C$  une vague avec un initiateur  $p$ , tel qu'un évènement de décision  $d_p$  se produit dans  $p$ . Alors  $C$  implique au moins  $N$  messages.

#### Preuve :

D'après le lemme 1 chaque évènement de  $C$  est précédé par un évènement dans  $p$ . En appliquant le même raisonnement que dans le lemme 3 il est facile de montrer qu'au moins un évènement *envoi* a lieu dans  $p$ .

D'après le lemme 3 un évènement d'envoi a lieu dans chaque processus autre que  $p$ , donc le nombre de message minimum est au moins  $N$

## Borne inférieure sur la complexité des vagues

### Algorithmes sur un réseau quelconque

Soit  $A$  un algorithme de vague pour un réseau quelconque sans connaissance à priori de l'identité des voisins, alors  $A$  échange au moins  $|E|$  messages à chaque calcul.

#### Preuve :

Soit  $C$  une vague dans laquelle moins de  $|E|$  messages sont échangés. Il y a donc une arrête  $xy$  qui n'est pas parcourue. Soit le réseau  $E'$  obtenu par l'insertion d'un processus  $z$  entre  $x$  et  $y$ . Comme les processus n'ont pas connaissance de l'identité de leurs voisins, alors l'état initial de  $x$  et  $y$  dans  $E'$  est le même que dans  $E$ . Donc les évènements se déroulent de la même manière et l'évènement *decide* n'est pas précédé par un évènement dans  $z$ .

# Algorithmes PIF

## Définition

- D'une manière générale les algorithmes de vagues sont ceux dont on a besoin pour diffuser une information à tous les processus d'un réseau et où certains processus doivent recevoir une notification lorsque la diffusion est terminée.
- Propagation of Information with Feedback (PIF)
- La notification de la fin de diffusion est considéré comme l'évènement *decide*



## Référence

- Gerard Tel. *Introduction to distributed Algorithms*, Cambridge Hall.