Vagues

Cours du 3 février

des vagues…

Topologie: déterminer la topologie du réseau

Tables de routages

Connectivité

Parcours d'un jeton (traversée : vague particulière sans concurrence des messages)

Exercices…

on suppose que tous les processus ont des identités. On veut un algorithme distribué qui calcule sur un noeud (tous les noeuds) la liste de toutes les identités. (décision: cette liste est calculée)

(1) montrer qu'un tel algorithme est nécessairement une vague

(2) partant d'un algorithme de vague donner un algorithme permettant de calculer la liste des identités

Vagues: anneau

jeton sur un anneau

initiateur i:

- S_i {une seule fois}-> send <> au suivant
- R_i {réception de <>} -> recevoir <>; Rec_i=True
- D_i {Rec_p}-> décider

autres p:

R_p {réception de <>} -> recevoir <>; Rec_p=True

S_p {Rec_p}-> send <> au suivant

En général, on notera e_p le premier événement de la vague sur p: si p est un initiateur cet événement est une émission, sinon c'est une réception

e_i: exécution de S_i

g_p: exécution de R_p

d_i: exécution de D_i

f_p: exécution de S_p

g_p: exécution de R_i o

En notant e_p le premier événement de la vague sur p ($p \neq i$) on a $e_p = g_p$

$\forall p : g_p \prec f_p \wedge f_p \prec g_{p+1}$ et $e_i \prec f_i$ et $g_i \prec d_i$ d'où

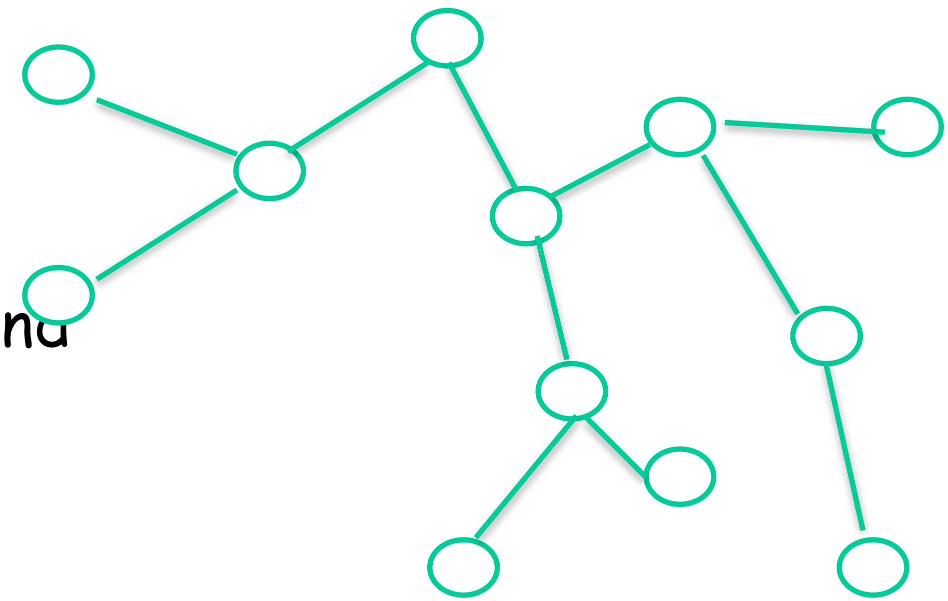
$e_i \prec g_{i+1} \prec f_{i+1} \cdots \prec g_{i-1} \prec f_{i-1} \prec g_i \prec d_i$ et

$$\boxed{\forall q : e_i \preceq e_q \preceq d_i}$$

Vagues : arbre

Arbre avec des liens bidirectionnels

- initiateurs: toutes les feuilles
- un noeud transmet <> quand il a reçu <> de tous ses voisins sauf un
- décision: celui qui a reçu <>de tous ses voisins (un ou deux décideurs)



Arbre

- Code pour chaque processus p:
  - $R_p : \{\text{message} < > \ \text{de } q\} \longrightarrow$
    $recevoir < > ; Rec_p[q] := true$
  - $S_p : \{\exists q \in Voisins(p) \wedge \forall r \neq q : Rec_p[q] = True$
    $\wedge \neg fait\} \longrightarrow \ send < > \ \text{to } \ q$
    $fait := true$
  - $D_p : \{\forall q \in Voisins(p) : Rec_p[q]\}$
    $\longrightarrow decide$

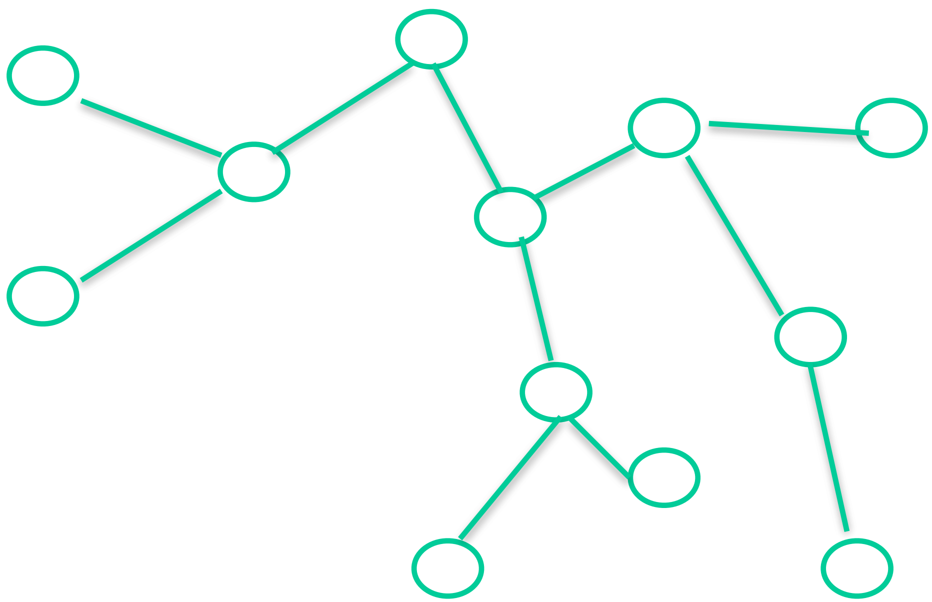
- Code pour chaque processus p:
  - R_p : {message $<>$ de q} $\longrightarrow$

    $$recevoir <> ; Rec_p[q] := true$$
  - S_p : $\{ \exists q \in Voisins(p) \wedge \forall r \neq q : Rec_p[q] = True$

    $$\wedge \neg fait\} \longrightarrow send <> \text{ to } q$$

    $$fait = true$$
  - D_p : $\{\forall q \in Voisins(p) : Rec_p[q]\}$

    $$\longrightarrow decide$$



Arbre

T_{pq} l'ensemble des noeuds accessibles à partir de p sans passer par pq

$f_{p,q}$: émission de p vers q

$g_{p,q}$: réception en q de p

e_p: début de la vague sur p

d_p: décision (sur p)

$$T_{pq} = \{p\} \cup \bigcup_{r \in Voisins(p) - \{q\}} T_{rp}$$

$$\Pi = \{p\} \cup \bigcup_{r \in Voisins(p)} T_{rp}$$

arbre

Lemme: $\forall s \in T_{pq} : e_s \preceq g_{pq}$:

- induction sur $\preceq$:
  - si s est une feuille $e_s \preceq f_{sx}$ et $T_{sx} = \{x\}$
  - on a $e_p \preceq f_{pq} \preceq g_{pq}$ (toujours) et $g_{rp} \preceq f_{pq}$ (algorithme)
  - par induction $e_s \preceq g_{rp}$ pour $s \in T_{rp}$ et donc: $e_s \preceq g_{pq}$

$\forall s \in \Pi : e_s \preceq d_p$

- si $s = p$ clair
- si $s \in T_{rp}$ pour $r \in Voisins(p)$
  - $g_{rp} \preceq d_p$ (algorithme)
  - et $e_s \preceq g_{rp}$ (lemme)
  - d'où $e_s \preceq d_p$

arbre

vivacité…

si tous les événements possibles de l'algorithme sont exécutés alors une décision est prise

Supposons qu'il n'y a plus de message à envoyer et qu'il n'y a pas de décision

- Rec_p bits: on en a $2(n-1)$ (2 x nombre d'arêtes) et soit F le nombre de ces bits à False
- si K est le nombre des messages émis (après réception) $F = 2(n-1) - K$
- Si pas de décision les K qui ont émis ont $F_p = 1$ les autres $F_p > 2$ donc $F \geq 2(n-K) + K$ (car $n-K$ n'ont pas émis et K ont émis)
- or $F = 2(n-1) - K < 2(n-K) + K$ d'où contradiction (il y a une décision ou un message à envoyer)

Arbre

- structure d'arbre (avec liens bidirectionnels)
- les feuilles sont initiateurs (comment permettre qu'un noeud quelconque soit initiateur?)
- un ou deux décideurs
- 2(n-1) messages échangés (chaque arête est parcourue 2 fois)
- temps? (comment mesurer le temps)?

Probe-echo

graphe quelconque (connexe) avec des liens bidirectionnels

- Un initiateur envoie à tous
- Un noeud identifie l'origine du premier message comme son père:
  - transmet à tous sauf son père
  - transmet à son père quand il a reçu de tous

  - Dans une première phase on construit un arbre de racine l'initiateur (probe), ensuite on remonte des feuilles vers l'initiateur (écho)

Code

initiateur

$$S_p :< \text{ une seule fois } > \longrightarrow \forall r \in Voisins(p) : send <> \text{ to } r$$
$$R_p :< \text{ message de } q > \longrightarrow receive <>: Rec_p[q] := true$$
$$D_p :< \forall q \in Voisins(p) : Rec_p[q] > \longrightarrow decide$$

les autres

$$R_p :< \text{ message de } q > \longrightarrow receive <>: Rec_p[q] := true$$
$$\text{if } (father_p = NIL)\{$$
$$father_p := q$$
$$\forall r \neq q \in Voisins(p) : send <>$$
$$\}$$
$$S_p :< \bigwedge_{q \in Voisins(p)} Rec_p[q] > \longrightarrow send <> \text{ to } father_p$$

Probe-echo

Probe-echo

- Probe-echo construit un arbre couvrant du graphe de communication pointant vers la racine initiateur ($p \rightarrow q \Leftrightarrow father_q = p$)
  - Tous les sommets sont atteints
  - Pas de cycle (sinon contredit le fait que $father_p$ est le <u>premier</u> noeud duquel p reçoit)
  - Pendant la construction de l'arbre si p est sur une branche de i à p alors $i \preceq p$,
    - soit C une exécution menant à un décision:
      - f_p envoi à $father_p$
      - g_p reception par $father_p$
      - T_p le sous-arbre de racine p
    - On a (par induction- arbre construit)
      - $\forall p : father_p \in C$
      - $\forall s \in T_p : \exists e \in E_s : e \preceq g_p$
  - D'où $\forall p \exists e \in E_p : i \preceq e \preceq d$

Probe-echo

- Graphe quelconque (connexe!) avec liens bidirectionnels
- un initiateur
- construction d'un arbre de racine l'initiateur
- chaque arête est parcourue deux fois une de $father_p$ vers p (construction de l'arbre) et l'autre de p vers $father_p$: nombre de messages 2E (E nombre d'arêtes)

Exercices (suite)

on suppose que tous les processus ont des identités. On veut un algorithme distribué qui calcule sur un noeud (tous les noeuds) la liste de toutes les identités. (décision: cette liste est calculée)

(1) montrer qu'un tel algorithme est nécessairement une vague

(2) partant d'un algorithme de vague donner un algorithme permettant de calculer la liste des identités

(3) utiliser l'algorithme de l'arbre pour cela

(4) utiliser l'algorithme probe-echo

(5) Si on veut que ce soit un processus particulier p qui initie et obtienne le résultat comment peut-on procéder. Si on veut que tous les processus obtiennent le résultat comment faire?

Phases…

- tous les noeuds sont initiateurs
- (di)graphe quelconque (fortement connexe)
- D phases (D diamètre du graphe): dans une phase, chaque noeud envoie à tous ses voisins (sortants), attend d'avoir reçu de tous ses voisins (entrants) les messages de la phase pour passer à la phase suivante
  - décision après D « phases »

Phases…

In_p: liens entrants

Out_p: liens sortants

$Rcount_p[q]$: n de messages reçus de q par p

$Scout_p$: n de messages emis par canal de sortie

D: diamètre du graphe

$$S_p :< \forall q \in In(p) : Rcount_p[q] \geq Scout_p \wedge Scout_p < D >$$
$$\longrightarrow \forall r \in Out_p : send <> \text{ to } r; Scout_p := Scout_p + 1$$
$$R_p :< \text{ message de } q >$$
$$\longrightarrow receive <>: Rcount_p[q] := Rcount_p[q] + 1$$
$$D_p :< \forall q \in In_p : Rcount_p[q] \geq D >$$
$$\longrightarrow decide$$

Phases

Phases

Remarques:

- (di)Graphe quelconque (fortement connexe)
- Si $d(q,p) = l$, alors la phase la plus petite dans laquelle p entend parler de q est la phase l (et réciproquement si l est la phase la plus petite dans laquelle p entend parler de q alors $d(q,p) = l$).
  - On peut utiliser cette propriété pour calculer les distances entre processus et calculer des tables de routage minimal.
- ($d(q,p) = l$ est la longueur du plus court chemin de q vers p -attention les chemins sont « orientés »)

« gossip » (algorithme de Finn)

on « bavarde » jusqu'à avoir des informations de tous

- HO_p ceux dont a entendu parler
- OK_p ceux dont les voisins entrants ont entendu parler.
- initialement: $HO_p = \{p\}$, $OK_p = \emptyset$,
- les messages contiennent HO_p, OK_p: quand p reçoit

  $< h, o >$: $HO_p := h \cup HO_p$ et $o \cup OK_p$
- quand p a reçu de tous ses voisins entrants:

  $OK_p = OK_p \cup \{p\}$
- quand $OK_p = OH_p$ décision

« gossip »

$$
\begin{aligned}
&S_p : \{true\} \longrightarrow send < HO_p, OK_p > \ \text{ to } q \in Out_p \\
&R_p : \{\text{ message de } q\} \longrightarrow receive < HO, OK > \\
&\qquad\qquad\qquad Rec_p[q] := True; \\
&\qquad\qquad\qquad HO_p := HO_p \cup HO \\
&\qquad\qquad\qquad OK_p := OK_p \cup OK \\
&D_p : \{OK_p = HO_p\} \longrightarrow decide \\
&A_p : \{\forall q \in In_q : Rec_p[q]\} \longrightarrow OK_p := OK_p \cup \{p\}
\end{aligned}
$$

On ne précise pas quand on envoie des messages: S_p est activé quand on veut -(on peut aussi envoyer un message uniquement quand il y a un changement)

Gossip

- Quand $HO_p = OK_p$, tous les processus dont p a reçu un message sont eux-même connus de leurs voisins entrants. Le graphe étant fortement connexe, OK_p contient tout le monde (si $(\forall p : p \in L \Rightarrow In_p \subseteq L)$ et $L \neq \emptyset$ alors L contient tous les sommets du graphe)
- (p peut n'envoyer des messages que quand OK_p ou HO_p est modifié: dans ce cas $2n$ valeurs possibles pour OK_p et HO_p, on échange donc au plus $2ne$ messages (e est le nombre d'arcs)
- On a besoin de l'identité des processus
- (Di)graphe quelconque : fortement connexe
- En cas de changement de la topologie (In_p/Out_p) l'algorithme re-converge

Gossip

(0) on a toujours $OK_p \subseteq HO_p$; si HO^a est la valeur de HO au moment de l'évènement a: $p \in HO^a \Rightarrow e_p \preceq a$

(1) si $OK_p = HO_p = L$ alors $\forall x : x \in L \Rightarrow In_x \subseteq L$

soit q, il existe un chemin (connectivité du graphe) de q à p:

$q = p_0 \rightarrow \cdots p_i \rightarrow \cdots \rightarrow p_{n-1} = p$

Alors $p_{n-1} = p \in L$ et $p_i \in L \Rightarrow p_{i-1} \in L$ et ainsi $q \in L$, et donc quand $OH^a = OK^a$: $\boxed{\forall q \in \Pi : e_q \preceq a \preceq d_p}$ (e_q premier évènement de la vague sur q)

(2) $OK_p = HO_p$ sera bien réalisé un jour: (Si $q \in HO_p - OK_p$, un jour $q \in OK_q$ et l'algorithme assure qu'un jour $q \in OK_p$)

digression…

Autre gossiping…

Autre Gossiping

Algorithmes de gossiping diffusion épidémique: (pull-push)

```
loop
    wait(Δ)
    p ← random peer
    if push and in state I then
        send update to p
    end if
    if pull then
        send update-request to p
    end if
end loop
```

```
procedure onUpdate(m)
    store m.update        ▷ means switching to state I
end procedure

procedure onUpdateRequest(m)
    if in state I then
        send update to m.sender
    end if
end procedure
```

État I = état infecté

coupon problème

compléter son album de n vignettes en tirant à chaque fois une vignette:

t_i le temps pour obtenir la i-ème vignette

- $\dfrac{n-i+1}{n}$ est la probabilité d'en obtenir une nouvelles si on en a i-1, et donc

$$E(t_i) = \frac{n}{(n-i+1)}$$

T_n v.a. du temps pour n vignettes:

$$E(T_n) = \sum_{1 \le i \le n} T_i = \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} = n\,.H_n$$

et $O(n\log(n))$