UJF - UFR-IMA - MST ESI2
ISTG - RICM2
Examen d'algorithmique distribuée
Durée 1h

Tous documents autorisés – Le barême est indicatif

12 septembre 2001

Soit un réseau fiable de type graphe bidirectionnel avec cycle. On souhaite donner à tout noeud la possibilité de diffuser sans restriction une suite de messages à tous les autres noeuds. Une simplification de la vague pour graphe bidirectionnel quelconque permet de garantir qu'un message émis par un noeud atteint tous les autres.

1) diffusion (7 pts) Compléter le schéma simplifié permettant à tous les noeuds de diffuser des messages vers tous les autres noeuds. Tout doublon de message doit être éliminé. Tout noeud peut diffuser une suite de message sans que l'émission d'un message nécessite d'attendre la fin de la diffusion du message précédemment émis par ce noeud. `transférer_message_vers_applications()` et `décision_de_diffusion_de()` ne sont pas à définir. **La vague la plus générale[1] pour graphe quelconque non fifo pourrait être utiliée directement. Cependant la diffusion ne s'intéressequ'au flux. On pourrait éliminer la gestion du reflux. Une première solution qui ne garde que la dernière vague par émetteur est la suivante.**

```
// L ensemble de liaisons
// nodeId() fonction donnant le numéro du noeud l'exécutant
int lastH[n]=0 ; // numero de dernière vague par noeud
int HL =0; // horloge logique
loopALT
 //traitement d'une requête de diffusion
décision_de_diffusion_de( msg ) ==>
         lastH[nodeId()]=HL++
         pourtout l de L l!<DIFF,msg,nodeId(),lastH[nodeId()]>
 //réception d'un message
pourtout lin de L lin?<DIFF,msg,I,h> ==>
         if (lastH[I]<h) // première occurence :acceptation du message
        lastH[I]=h
                pourtout l de L-{lin} l!<DIFF,msg,I,h>
                transférer_message_vers_applications(msg)
         else // cycle RIEN . On oublie
end loopALT
```

Mais le mieux est d'utiliser l'algorithme de diffusion sur IP : RPF Reverse Path Forwarding. On ne propage que les messages arrivant sur le chemin de retour vers l'émetteur.

[1] plusieurs émetteurs, plusieurs vagues par émetteur.

```
// L ensemble de liaisons
// nodeId() fonction donnant le numéro du noeud l'exécutant
int up[n]=NIL ; // numero de chemin de retour
loopALT
 //traitement d'une requête de diffusion
décision_de_diffusion_de( msg ) ==>
         pourtout l de L l!<DIFF,msg,nodeId()]>
 //réception d'un message
pourtout lin de L lin?<DIFF,msg,I> ==>
         if (I!=ICI()  // non iniiateur
      && (up[I]==NIL  // chemin retour inconnu
           || up[I]==lin // chemin retour connu
             ))
 // première occurence du message
        up[I]=lin
                pourtout l de L-{lin} l!<DIFF,msg,I,h>
                transférer_message_vers_applications(msg)
         else // cycle RIEN . On oublie
end loopALT
```

2) Cohérence faible (3 pts) Cependant, on peut chercher à vérifier des propriétés particulières sur la diffusion c'est à dire sur la façon dont les diffusions sont perçues par les destinataires. La cohérence faible consiste simplement à garantir que les messages issues d'un même émetteur sont reçues dans l'ordre d'émission par les récepteurs.

1. la propriété FIFO des liaisons suffit-elle à garantir la cohérence faible ? Pourquoi? **La propriété FIFO garantit la cohérence faible dès que le routage est déterministe. Dans une vague, seul les refluxs de vague peuvent remonter par des chemins up différents dès qu'il y a plusieurs routes possibles entre paires de noeuds. Les 2 solutions précédentes offrent la cohérence faible si le réseau est FIFO. La première solution est non seulement sans cohérence mais aissi non fiable si le réseau est NON FIFO fiable. En effet elle "oublie" les vagues qui arrive en retard. Par contre RPF est fiable si le réseau est NON FIFO fiable. C'est aussi le cas de la vague la plus générale pour graphe quelconque non FIFO**
2. Dans le cas où la propriété de cohérence faible n'est pas implicitement assurée, rappelez une façon de la garantir.**Il faut retarder la délivrance à l'application des messages hors séquence. Retarder suffit car les liaisons sont fiables.**

3) Cohérence forte (5 + 5 pts) Dans la cohérence forte, on veut garantir que l'ensemble des messages issues de plusieurs émetteurs est reçu exactement dans le même ordre par tous les récepteurs. Cette propriété est plus difficile à garantir car elle revient à construire un ordre total sur les messages quels que soient les émetteurs. Différentes solutions sont possibles à partir des différents schémas d'algorithme assurant une sérialisation du fonctionnement des noeuds.

1. Une solution est de n'utiliser le réseau que comme un anneau FIFO monodirectionnel. Le droit de diffuser est donner par un jeton circulant sur l'anneau. Proposer une diffusion pour anneau monodirectionnel FIFO où le jeton de diffusion est immédiatement relaché par un noeud après émission du message à diffuser sur l'anneau. A quel moment, le message peut il être transférer à l'application ? **Le principe pour un émetteur est d'attendre le jeton, émettre ses données puis reémettre le jeton. Lorsque ses données lui reviennent apès un tour, il les retirent de l'anneau. Les données sont transmises à l'application dès leurs réceptions.**

   ```
   loopALT
   ```

```
 //réception d'un message
 lin?<MSG, I,msg> ==>
         if(I!=nodeId())
             transférer_message_vers_applications(msg)
     lout!<MSG, I,msg>
         else // RIEN
//réception jeton
 lin?<TOK> && décision_de_diffusion_de( msg )==>
         lout!<MSG,nodeId(),msg>
 lout!<TOK>
 lin?<TOK> && !décision_de_diffusion_de( msg )==>
 lout!<TOK>
end loopALT
```

2. Une autre solution est de se ramener à un algorithme assurant la sérialisation du fonctionnement des noeuds. L'exclusion mutuelle est un tel algorithme. Modifier l'algorithme d'exclusion mutuelle sur graphe bidirectionnel FIFO quelconque vu en cours pour assurer une diffusion à cohérence forte ? Le LOCK doit garantir qu'un seul noeud a acquis le droit d'émettre. Le UNLOCK peut être utilisé pour émettre le message et relacher l'exclusion.

On effectue un LOCK puis une diffusion puis un UNLOCK. La diffusion pourrait une forme simplifié de la première question car l'exclusion mutuelle garantit qu'il y a 1 seule diffusion à la fois

```
Initialement WAITED==L DEFER=={},
int HL=0 ; // horloge diffusion
int D=0; // date dernière requete diffusion
int lastH[n]=0 ; // numero de dernière vague par noeud
loopalt
// lancer demande d'exclusion mutuelle
décision_de_diffusion_de( msg)  && état== dehors  ==>
    état = requete_en_cours;
    D=HL;HL++;
    WAITED==L;DEFER=={};
    pourtout l de WAITED l!<REQ,D,nodeId()>

exist l de L  l?<REQ,d,n>  ==> HL=max(HL,d+1);
          if(etat==dedans || (etat==requete_en_cours && GAGNE(D,d,n))
             DEFER=DEFER+{l} /* autorisation différée */
         else l!<ACK>

exist l de L  l?<ACK>  ==>
          WAITED=WAITED-{l}

// verrou acquis - lancer diffusion
état = requete_en_cours && WAITED=={}   ==>  état = dedans
     lastH[nodeId()]=D;
     pourtout l de L l!<DIFF,msg,nodeId(),lastH[nodeId()]> // diffuser
     pourtout l de DEFER l!<ACK>  // débloquer
     WAITED=L  DEFER={};etat= dehors ;

 //réception d'un message
pourtout l de L l?<DIFF,msg,I,h> ==>
```

```
         if (lastH[]<h) // première occurence
        lastH[I]=h
                pourtout l de L-{l} l!<DIFF,msg,I,h>
                transférer_message_vers_applications(msg)
         else // cycle :  RIEN

endloopalt

avec GAGNE(x,y ,s) ((x<y) || ((x==y) && (ICI<s)))
```