8/01/2010

M1 Info

Protocoles Réseaux
Devoir n°2

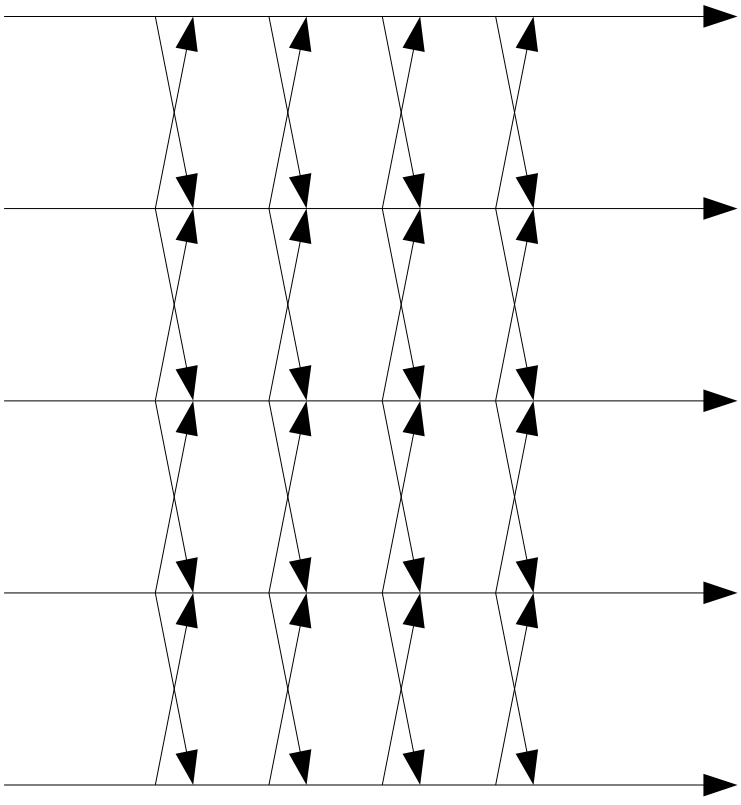
Question 1

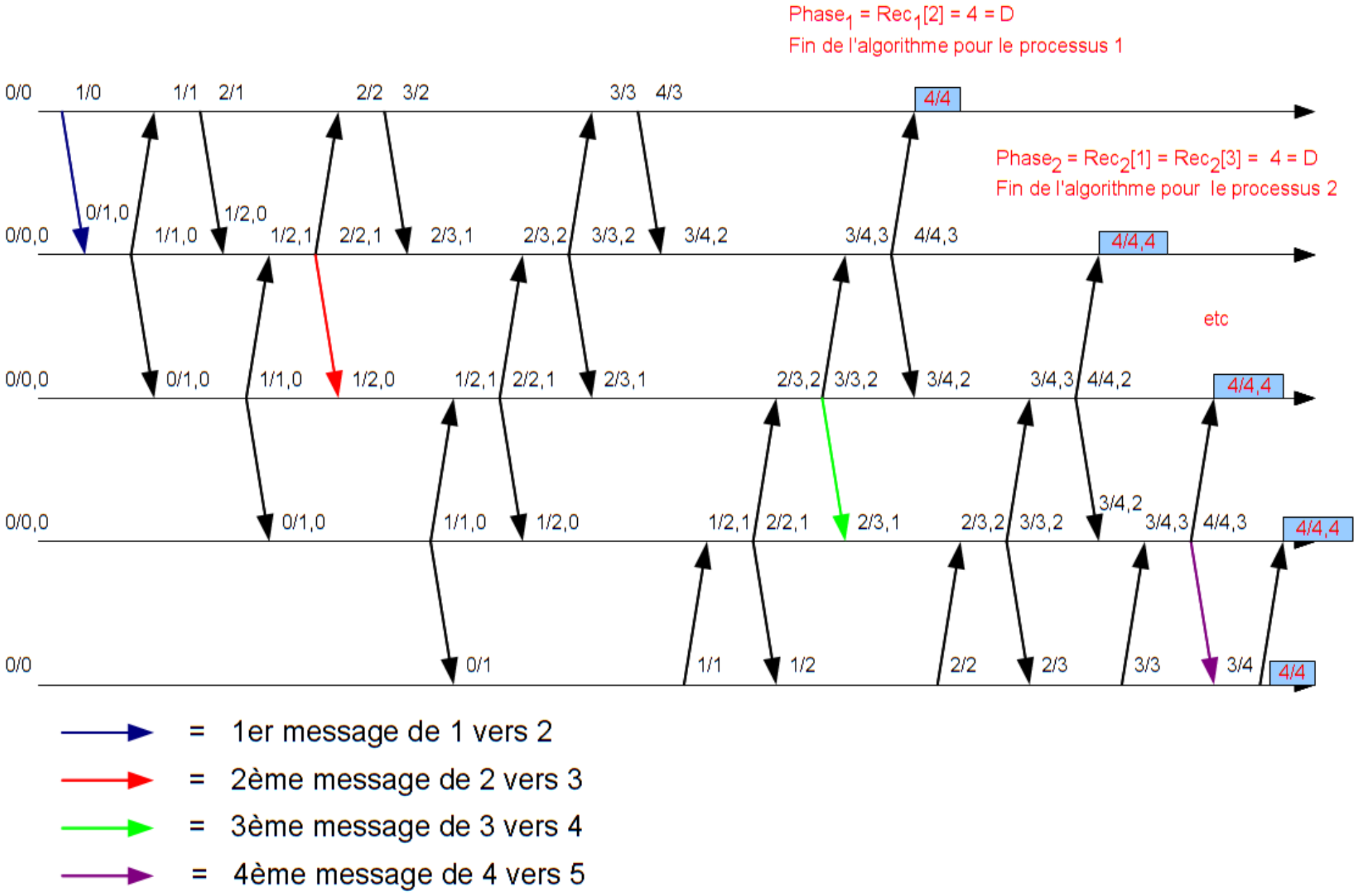
On peut donner l'exemple d'exécution suivant :

voir page 2

Sur ce schéma on note :
Phase / *Rec*[voisin plus petit (en numéro)], Rec[voisin plus grand]

Remarque : Dans cette exemple on suppose que les processus ne "réagissent" pas tous aussi vite quand plusieurs peuvent envoyer pour clairsemer le dessin (ici, le processus 1 a le temps d'attente le plus court et le 5 a le temps d'attente le plus long), mais on pourrait aussi supposer qu'ils ont tous le même temps d'attente, ce qui donne un résultat plus court et plus compact :




Phase1 = Rec1[2] = 4 = D
Fin de l'algorithme pour le processus 1
Phase2 = Rec2[1] = Rec2[3] = 4 = D
Fin de l'algorithme pour le processus 2
etc
0/0
1/0
1/1
2/1
2/2
3/2
3/3
4/3
4/4
0/0,0
0/1,0
1/1,0
1/2,0
1/2,1
2/2,1
2/3,1
2/3,2
3/3,2
3/4,2
3/4,3
4/4,3
4/4,4
0/0,0
0/1,0
1/1,0
1/2,0
1/2,1
2/2,1
2/3,1
2/3,2
3/3,2
3/4,2
3/4,3
4/4,2
4/4,4
0/0,0
0/1,0
1/1,0
1/2,0
1/2,1
2/2,1
2/3,1
2/3,2
3/3,2
3/4,2
3/4,3
4/4,3
4/4,4
0/0
0/1
1/1
1/2
2/2
2/3
3/3
3/4
4/4
= 1er message de 1 vers 2
= 2ème message de 2 vers 3
= 3ème message de 3 vers 4
= 4ème message de 4 vers 5

Comme indiqué sur la figure, on trouve ces 4 événements dans le même ordre que listés sur le sujet.

Non, on ne peut pas trouver ces 4 évènements dans un ordre différent :

Pour la suite, notons que :

- D'après l'algorithme, la *Phase* de chaque processus *p* démarre à 0 et augmente de 1 à chaque émission d'un message à tous ses processus voisins (c'est ainsi qu'on appellera tous les *q* tels que *q* appartient à In_p), donc la $Phase_p$ représente le nombre de fois où un processus a "parlé" à ses voisins.
- Toujours d'après l'algorithme, le *Rec*[*q*] de chaque processus n'augmente que quand ce processus reçoit un message depuis *q*, donc $Rec_p[q]$ représente le nombre de fois où *p* a "entendu" *q*.

Donc, quand 2 envoie un 2ème message à 3, c'est qu'il passe de phase 1 en phase 2.
Or, d'après l'algorithme, la *Phase* d'un processus *p* ne peut augmenter que si elle est inférieure ou égale à tous les *Rec* de *p*.
Donc pour que $Phase_2$ passe à 2 il faut que tous les Rec_2 soient au moins à 1, y compris $Rec_2[1]$, donc il faut que 2 ait déjà reçu un message depuis le processus 1 (fin de la flèche bleu), donc il faut que 1 ait déjà envoyé un message à 2 (début de la flèche bleu).

En conséquence, les 2 évènements « *émission du premier message envoyé par le processus 1 au processus 2* » et « *émission du 2ème message envoyé par le processus 2 au processus 3* » se trouveront toujours dans cet ordre là.

De même pour les messages de 2 à 3 et de 3 à 4, et pour ceux de 3 à 4 et de 4 à 5.

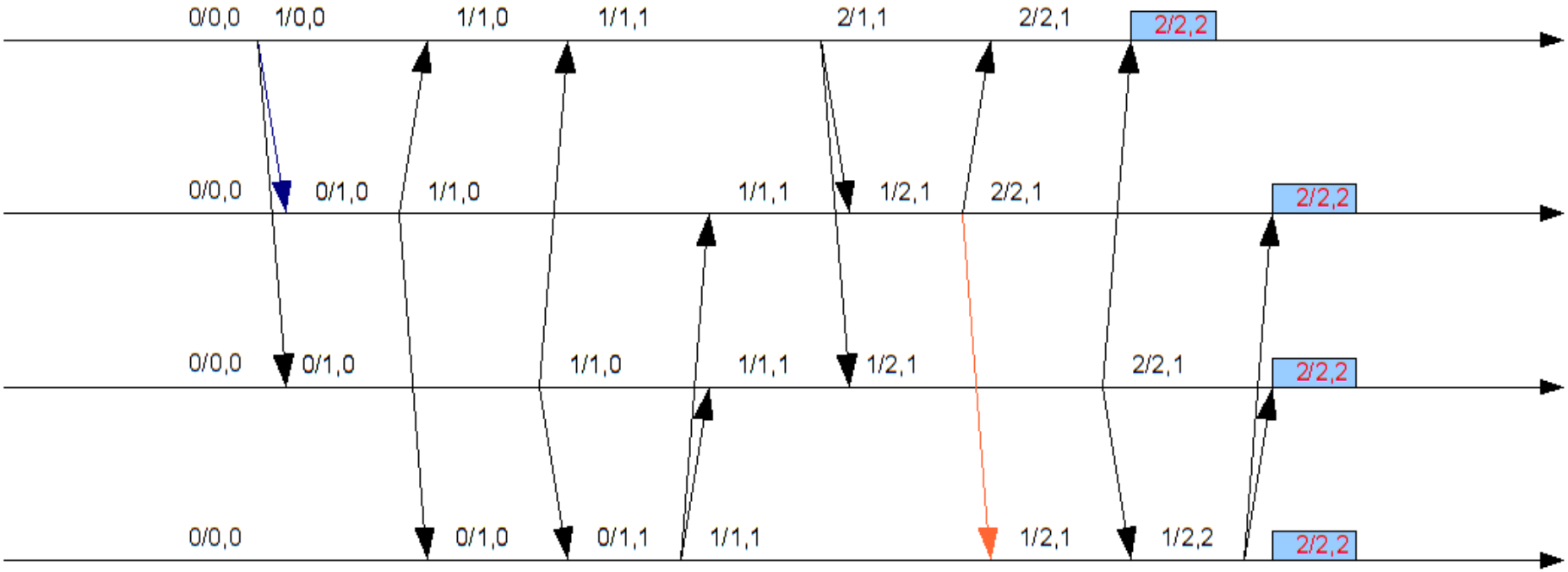
Ces 4 évènements seront donc toujours dans l'ordre trouvé dans l'exemple.

Question 2

Voici un exemple d'exécution (construit comme le précédent) :

voir page 4

Pour les mêmes raisons que dans la question 1, ces 2 évènements sont toujours dans le même ordre.



= 1er message de 1 vers 2

= 2ème message de 2 vers 4

Question 3

Montrons que l'algorithme de phases termine :

Pour que cet algorithme termine pour chaque processus, il faut que toutes les *Phase* aient atteint le diamètre, et que tous les *Rec* aient atteint les *Phase*.

Une *Phase* ne peut jamais dépasser *D*, puisque *Phase* doit être strictement inférieure à *D* pour pouvoir augmenter, et les *Rec* d'un processus p sont -au délai de réception près- les *Phase* des voisins de p, donc les *Rec* non plus ne peuvent pas dépasser *D*.
Donc l'algo termine quand toutes les *Phase* et tous les *Rec* valent *D*.

Comme les *Phase* ne peuvent qu'augmenter, et le font à chaque envoi, l'algo ne peut pas tourner en rond, ce qui nous limite à 2 possibilités :

- Soit les *Phase* et les *Rec* ont toutes atteint le diamètre et l'algo est terminé.
- Soit l'algo se "bloque" : plus personne n'envoie, mais les *Phase* et les *Rec* n'ont pas tous atteint *D*.

Montrons que cette 2ème situation est impossible :

Dans la suite nous supposerons que :

- La transmission est fiable : si $Phase_p$ vaut 3 et que *q* est voisin de *p*, alors $Rec_q[p]$ finira par valoir 3 (le message finira par arriver).
- Quand un processus peut envoyer, il finira par le faire (les temps d'attentes qu'on a mis dans les exemples ci-dessus ne peuvent pas être infinis).

L'algo de phases ne peut pas se bloquer, car tant que tout le monde n'a pas atteint le diamètre, on a toujours au moins un processus qui peut émettre : celui dont la phase est la plus petite (non strictement), car il respecte forcement les 2 conditions d'émission :

- Sa *Phase* est plus petite que *D* (si certaines *Phase* n'ont pas atteint *D*, alors la plus petite ne l'a pas atteint).
- Elle est aussi forcement (après réception des messages) inférieure ou égale à tous ses *Rec*, car si un des *Rec* de ce processus était plus petit, la *Phase* correspondante à ce voisin serait également plus petite (car on a reçu autant de messages de ce voisin qu'il en a émis, toujours après stabilisation), on aboutirait donc à une contradiction (le processus dont la *Phase* est la plus petite aurait un voisin dont la *Phase* serait encore plus petite), donc cette condition est toujours respectée aussi.

Donc l'algorithme de phases ne se bloque pas et ne tourne pas en rond.
Donc l'algorithme termine.

cqfd

Question 4

Notons la relation de causalité "≤".
Notons les v(p) les voisins d'un processus p (distance 1).
Notons q un processus qui termine l'algo.

Montrons que tous les autres ont alors déjà commencé :

Si q termine, alors il a :
D/D, D..., D (on reprend la notation des schémas)

Donc les v(q) ont tous, à un moment antérieur dans l'ordre causal, Phase = D (car si q à reçu D messages de leur part, ils en ont tous envoyé D messages à un moment antérieur dans l'ordre causal).
Donc tous leur *Rec* sont, à un moment antérieur dans l'ordre causal, d'au moins D-1 (d'après l'algorithme, la *Phase* d'un processus ne peut pas avoir 2 d'avance sur l'un de ses *Rec*, puisqu'elle doit leur être inférieure ou égale pour augmenter).
Ce qui donne au minimum :
D / D -1, D -1..., D -1

En suivant le même raisonnement, on a pour les v(v(q)) au minimum :
D -1 / D -2, D -2..., D -2

Pour les v(v(v(q))) on a au minimum :
D -2/D -3, D -3..., D -3

etc

On aboutit à v(v(...v(q)...)) avec au maximum D "v" (il ne peut pas y avoir de sommet à une distance de q plus grande que D) qui ont au minimum Phase = D – (D +1) = 1.

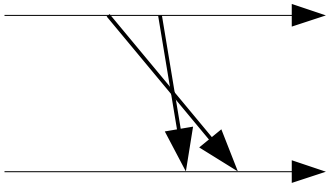
Donc, quand le processus q termine, tous les autres processus, même les plus éloignés, sont au moins en *Phase* 1, donc ils ont tous déjà commencé, et ces évènements sont comparables dans l'ordre causal, on a donc $e_p \leq f_q$ pour tous les processus p et q.

cqfd

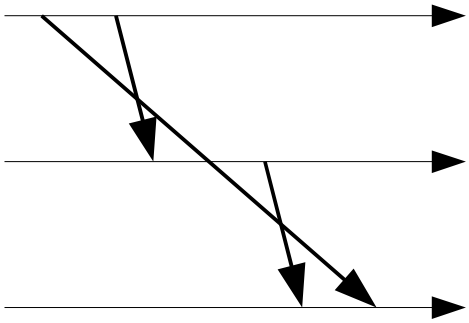
Question 5

Que se passe-t-il si la communication n'est plus FIFO ?
Vaste question !

Il se passe qu'on pourrait avoir, par définition, des évènements de ce genre ci :



Ou encore de ce genre là :



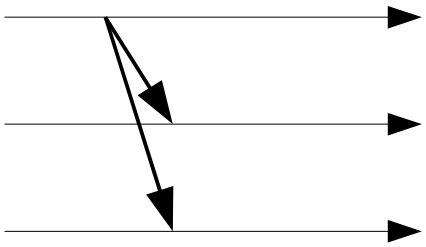
Si la question signifie :
« Si la communication n'est plus FIFO, les réponses aux questions précédentes s'en trouvent-elles changées ? »

Alors la réponse est :
« Non »

Les raisonnements utilisés ci-dessus ne s'appuyaient pas sur la propriété FIFO de nos réseaux.

Question 6

Il a 2 façon de comprendre la question :

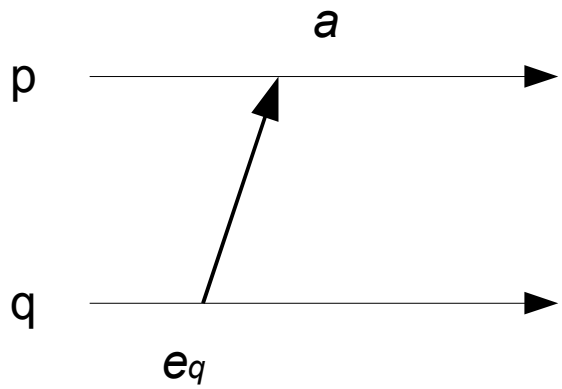


1) Si on comprend que sur ce dessin, 1 message est échangé (nombre de fois où un processus envoie à tous ses voisins), alors chaque processus fait D émissions (leurs *Phase* comptent ces émissions et s'arrêtent toutes à D comme montré précédemment), on a donc $D * |\Pi|$ messages échangés, où $|\Pi|$ est le nombre de processus.

2) Si on comprend que sur ce dessin, 2 messages sont échangés (on compte le nombre de flèches), alors chaque processus envoie autant de messages qu'il a d'arête à chaque phase, et comme chaque arête est partagée par 2 processus, et qu'on a D phases, on a alors $2 * D * |V|$ messages échangés, où $|V|$ représente le nombre d'arêtes.
   On peut aussi calculer ça de la façon suivante : $D * (\sum_{\text{pour tout p appartenant à } \Pi} |In_p|)$ (D fois le nombre de voisins de chaque processus).

Question 7

Il y a peut-être une erreur dans l'énoncé, ou alors je la saisi mal la proposition (1) :



Dans cet exemple, on a bien $e_q \leq a$, alors que p est à une *Phase* de 0, inférieure à $d(p,q) = 1$.

Question 8

Si on suppose que le diamètre est inconnu des processus, mais qu'ils connaissent le nombre |Π| de processus, et on peut alors remplacer *D* par |Π|-1, qui est le diamètre maximal du réseau.

On peut aussi supposer que les processus n'envoient plus des Ø mais des messages contenant la liste des processus dont eux-mêmes ont entendu parler.
Un processus pourrait alors probablement s'arrêter lorsque les derniers messages de chacun de ses voisins ne lui parle pas de processus que lui même ne connaisse déjà (par exemple, dans le graphe de la question 1, lors de sa 1ère réception, le processus 1 entend parler de 2 pour la première fois, lors de sa 2ème réception il entend parler de 3 pour la première fois, etc).
Mais cette solution ne marche peut-être pas toujours, il est possible qu'un processus s'arrête parfois trop tôt.

THE END

(Hors rendu)

Remerciements à Raphaël

source : http://info.paris7.free.fr
Questions, commentaires, critiques, remarques, etc : phosphore85@gmail.com