Algorithmique Distribuée : TD 2, bit alterné

Stéphane Devismes

11 septembre 2019

1 Le problème

Nous considérons maintenant des systèmes où les canaux de communications sont asynchrones et sujets à des pertes équitables de messages. Le but du protocole de *bit alterné* est de fournir des primitives d'envoi et de réception de messages fiables afin de rendre la gestion des pertes de messages transparentes à l'application. Ainsi, en remplaçant les primitives de communications bas niveaux par le bit alterné, les protocoles vus dans première partie du cours resteront fonctionnels dans des systèmes à communications non fiables, au prix d'un surcoût en messages.

Pour simplifier, nous nous restreignons à un réseau à deux nœuds liés où seul le processus e (l'émetteur) envoie des messages au processus r (le récepteur), via un lien de communication bidirectionnel.

2 Hypothèses

— Processus corrects et asynchrones.
— Liens FIFO asynchrones avec perte de message *équitable*. De plus, il n'y a ni création, ni duplication de message.
— Deux nœuds, notés e et r, liés par un lien de communication bidirectionnel.

3 Spécification

Les données à envoyer sont fournies par l'application de e via la file d'attente *Départ*. Chaque donnée dans *Départ* doit être *livrée* dans la file d'arrivée de r, *Arrivée*. Pour simplifier le raisonnement, nous considérerons que toutes les données de *Départ* sont virtuellement identifiées ce qui les rend distinguables. En d'autres termes, l'identifiant de la donnée sert uniquement dans le raisonnement, les processus n'y ont pas accès.

Ainsi le protocole du bit alterné doit vérifier la spécification suivante :

Définition 1 (Transmission FIFO fiable)**.**

1. *Toute donnée de la file Départ est insérée en temps fini dans la file Arrivée. (pas de perte)*
2. *Si une donnée d est insérée dans la file Arrivée, d a existé dans la file Départ. (pas de création)*
3. *Toute donnée d est insérée au plus une fois dans la file Arrivée. (pas de duplication)*
4. *Les données sont insérées dans Arrivée dans le même ordre que dans Départ. (FIFO)*

4 Principe

Le principe de l'algorithme est le suivant : pour transmettre une donnée d, e envoie régulièrement un message $DATA$ contenant d et une estampille (0 ou 1) à r jusqu'à ce que e reçoive un message d'accusé de réception Ack de r avec la même estampille. Pour cela, chaque processus dispose d'une variable booléenne « estampille » initialement égale à 0. Lorsque e démarre la transmission de la donnée d, il change son estampille, puis envoie un message contenant d et la valeur de son estampille. Il attend ensuite la réception d'un accusé de réception de la part de r. Si au bout d'un certain temps, e n'a pas reçu d'accusé de réception, il conclut que le message qu'il a envoyé ou son accusé de réception est perdu. Pour cela, e utilise un mécanisme d'attente (*minuteur*) et renvoie son message à l'expiration du délai d'attente. Ainsi, r peut recevoir plusieurs duplicatas

du même message. L'estampille permet alors d'éviter de livrer (c'est-à-dire insérer dans la file *Arrivée*) plusieurs fois le même message. Lorsque r reçoit la première copie du message, l'estampille du message est différente de son estampille locale, r livre le message, change la valeur de son estampille et envoie un accusé de réception à e. Toutes les autres copies du message reçues ne sont pas livrées car leur estampille sera égale à celle de r. Cependant, r accuse la réception de ces messages car la réception d'une copie d'un message de e peut signifier que l'accusé de réception précédent a été perdu. Lorsque e reçoit l'accusé de réception attendu, la transmission de d est terminée et on a la garantie que les valeurs d'estampille de e et r sont à nouveau égales. Ainsi, le système est dans un état similaire à l'état initial et e est prêt à transmettre une nouvelle donnée.

5 Minuteurs

Un *minuteur* (timer en anglais) est un dispositif qui permet de déclencher un évènement (une alarme) après qu'un certain temps se soit écoulé. Pour « démarrer un minuteur » vous utiliserez la primitive `ChargerMinuteur`(M,X) qui permet de déclencher un évènement **expiration de** M dans X unité de temps. L'**expiration de** M pourra être testée directement dans un « Si ».

6 L'algorithme

Écrivez le code de l'algorithme. Vous utiliserez les variables suivantes. Pour l'émetteur :
- *Départ* : file de données (variable d'entrée, donc non-initialisée)
- $Tag \in \{0, 1\}$: estampille Booléenne
- $Buffer$: $\perp$ ou donnée
- M : minuteur (vous utiliserez une périodicité de 10 unité de temps)

Pour le récepteur :
- *Arrivée* : file de données (variable de sortie qu'il faut initialiser à $\emptyset$)
- $Tag \in \{0, 1\}$: estampille Booléenne

Il y aura deux types de messages.
- Les messages de type $DATA$: $\langle DATA,D,T\rangle$ où D est une donnée et T est une estampille Booléenne.
- Les messages de type Ack : $\langle Ack,T\rangle$ où T est une estampille Booléenne.

Algorithme 1 Algorithme du bit alterné, code pour l'émetteur e, variable d'entrée *Départ* : file de données

Algorithme 2 Algorithme du bit alterné, code pour le récepteur r, variable de sortie *Arrivée* : file de données

7 Correction de l'algorithme

Question 1. Démontrez le lemme suivant :

Lemme 1 (pas de création)**.** *Si une donnée d est insérée dans la file Arrivée, d a existé dans la file Départ.*

Question 2. Nous considérons deux types d'évènements internes à l'émetteur e :

- Le *début de transmission de d*, noté $DEBUT\_TRANS(d)$, qui correspond à l'affectation de $Buffer$ à d lors de l'exécution de la ligne 8.
- La *fin de transmission de d*, notée $FIN\_TRANS(d)$, lorsque $Buffer$ passe de la valeur d à la valeur $\perp$, ligne 17.

Dans la suite, nous appellerons *transmission de d* la période entre $DEBUT\_TRANS(d)$ (inclus) et $FIN\_TRANS(d)$ (inclus).

Remarque 1. *Soit d la $i^{ème}$ $(i > 0)$ donnée transmise par e, tous les messages envoyés par e entre le début et la fin de la transmission de d sont estampillés avec la valeur $i \bmod 2$.*

Démontrez le lemme suivant :

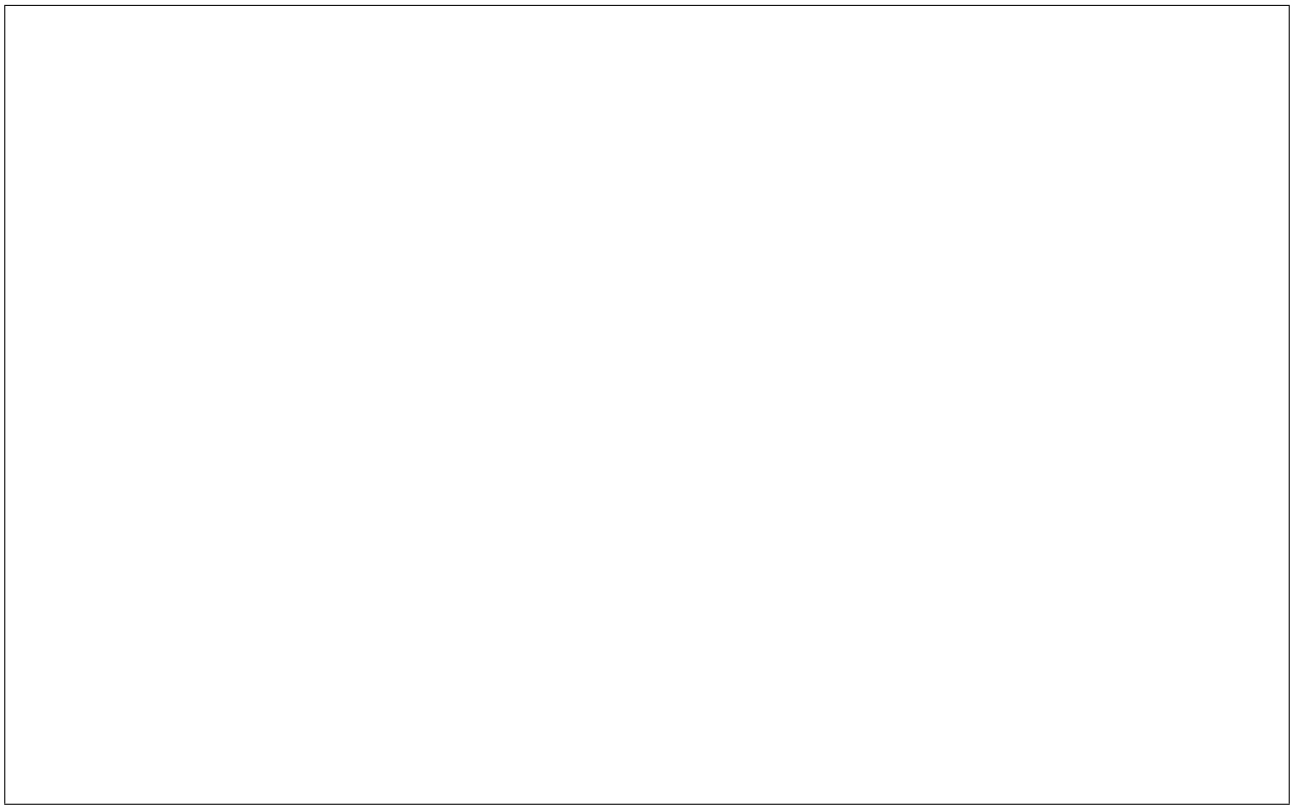
Lemme 2 (Pas de duplication)**.** *Toute donnée d est insérée au plus une fois dans Arrivée.*

Question 3. Nous disons que l'estampille x ($\in \{0, 1\}$) est *présente* dans le système si et seulement si au moins un de ces cas est vérifié :

- $Tag_e = x$,
- $Tag_r = x$, ou
- Il existe un message ($DATA$ ou Ack) estampillé avec la valeur x.

Démontrez par récurrence sur l'ordre de transmission des données le lemme suivant :

Lemme 3. *Lorsque e débute la transmission de sa $i^{ème}$ donnée $(i > 0)$, l'estampille $i \bmod 2$ n'est pas présente dans le système.*

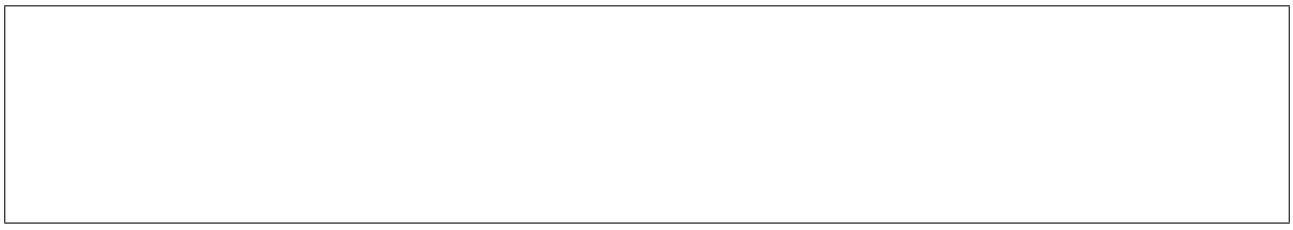


La transmission de la $i^{\text{ème}}$ donnée, d, termine lorsque e reçoit un Ack estampillé avec la valeur $i \bmod 2$. D'après le lemme précédent, cela est nécessairement dû à la réception par r d'un message $DATA$ contenant d. Toujours d'après le lemme précédent, lorsque r reçoit ce message pour la première fois, d est inséré dans *Arrivée*. D'où :

Corollaire 1. *Lorsque e finit la transmission de sa $i^{ème}$ donnée ($i > 0$), cette donnée a été insérée dans Arrivée.*

Question 4. Démontrez par contradiction le lemme suivant :

Lemme 4. *Toute transmission de donnée termine.*



D'après les lemme et corollaire précédents, ainsi que le fait que e teste régulièrement si des données a envoyé sont présentes dans *Départ* lorsque $Buffer = \perp$, nous avons :

Corollaire 2 (Pas de perte)**.** *Toute donnée de la file Départ est insérée en temps fini dans la file Arrivée.*

Lemme 5 (FIFO)**.** *Les données sont insérées dans Arrivée dans le même ordre que dans Départ.*

Preuve. Les données de *Départ* sont transmises dans l'ordre de leur insertion car *Départ* est une file. D'après les lemmes précédents, chaque donnée est insérées (exactement une fois) dans *Arrivée* durant sa transmission. Par suite, l'ordre est préservé. □

La propriété 1 est assurée par le corollaire 2. La propriété 2 est assurée par le lemme 1. La propriété 3 est assurée par le lemme 2. Enfin, la propriété 4 est assurée par le lemme 5. Ainsi :

Théorème 1. *Le protocole donné dans les algorithmes 1 et 2 assure la transmission FIFO fiable.*