Protocoles réseaux
TD n° 9 : Paquets en vol et diagrammes temporels

I) Lien avec fenêtre de taille 1

Dans cette section, on se place au niveau de la couche lien entre deux machines. Ce lien bidirectionnel peut *perdre* des paquets : un paquet envoyé soit n'arrive pas de tout (perte), soit arrive correctement (pas d'erreur, ni de création de paquet). On suppose ce lien suffisament court pour que ne pas plus d'un paquet en vol (fenêtre de 1 paquet) soit pertinent.

On suppose que la couche supérieure de la machine A doit transmettre à celle de la machine B un flux de données (suite d'octets sans fin, une éventuelle structuration de ce flux étant le travail de cette couche supérieure). Au niveau de la couche lien, on transmet des paquets de 1 ko maximum. Le lien est *synchrone* de la façon suivante : un paquet envoyé est soit reçu 1ms plus tard au maximum, soit perdu.

On dit que la transmission est *fiable* si pour toute donnée émise par A :
- B reçoit la donnée correctement,
- ou A reçoit une indication d'erreur

Exercice 1 : acquittement simple

On considère le protocole suivant. Les messages sont de deux types :
- Données. Entête constituée du caractère `'D'`, puis de la longueur du message (sur 2 octets) suivi des données elles-même (un segment d'au plus 1021 octets du flux)
- Acquittement. Message constitué d'un seul octet : `'A'`.

Le protocole est ainsi constitué :
- A envoie un paquet de données (au plus 1021 octets pris dans le flux) à B
- À chaque paquet de données reçu, B enfile son contenu dans son flux et envoie un acquittement
- A attend un des deux événements suivants
  - Soit un timeout a lieu : au bout de 2ms, A suspecte une perte et renvoie *le même* paquet de données que précédemment
  - Soit un acquittement est reçu : A pourra alors envoyer le segment *suivant* de données

Montrer que ce protocole n'assure pas une transmission fiable du flux de données.

Exercice 2 : bit alterné

On considère maintenant quatre types de messages : deux types de messages de données, `'D'` et `'d'`, et deux types d'acquittements `'A'` et `'a'`.

1. Spécifier un protocole utilisant judicieusement ces quatre types (A n'envoie que des données et B que des acquittements).
2. Montrer qu'il transmet de façon fiable le flux.
3. Pourquoi n'est-il pas utilisé en couche 4 par des protocoles tels que TCP ?

Exercice 3 : réseau asynchrone

On enlève l'hypothèse qu'un paquet non reçu au bout de 1ms est nécéssairement perdu.

1. Montrer que les protocoles des deux exercices précédents n'assurent pas une transmission fiable.
2. Rajouter une hypothèse pour que le protocole de l'exercice 2 devienne correct. Cette hypothèse ne doit pas être un délai maximum au-delà duquel on peut supposer quelque chose (le canal est *asynchrone*) ni une certitude que certains paquets arrivent (le canal est *avec pertes*).
3. Montrer que le protocole de transmission est fiable sous cette hypothèse.

II) TCP

Exercice 4 : avec ou sans TCP

400 Go de données doivent être téléchargés d'un serveur S vers une ordinateur C chaque nuit entre minuit et 7h du matin. Par ailleurs, la distance entre C et S est de 24km.

1. Quel est le débit nécessaire, en Mbps, afin que le téléchargement puisse se faire chaque nuit ?

On installe maintenant entre les deux machines une fible optique supportant un débit IP de 200 Mbps dans chaque sens. On utilise une application TCP qui fait des segments de 1ko de données chacun.

2. Combien de temps au minimum faut-il pour télécharger les données ?

3. Quelle est la latence de la ligne, au minimum ?

4. Quel est le temps mis à émettre un paquet de données ?

5. On suppose maintenant que C envoie un acquitement de 40 octets dès la reception d'un paquet, comme dans TCP, mais que, contrairement à TCP, S n'émet pas de nouveau paquet tant que le précédent n'a pas été acquité. Quelle est alors la latence ?

6. Quel est le débit ainsi obtenu ? Commentaire ?

7. Combien de bits sont-ils en vol à un instant t donné, de S vers C ?

Exercice 5 :

On rappelle que TCP utilise deux algorithmes de contrôle de congestion :

- l'algorithme **slow start**, où la taille de la fenêtre de congestion est augmentée de 1 MSS (maximum segment size) pour chaque acquittement reçu, et réduite une fois par RTT (round-trip time) si une perte de paquet a eu lieu ;
- l'algorithme **congestion avoidance**, où la fenêtre n'est augmentée que d'un MSS par RTT en l'absence de perte de paquets.

On considère deux nœuds A et B séparés par une route de 10ms. On suppose le débit de la route infini, c'est-à-dire que le débit est limité par la fenêtre de congestion et pas par le débit de la route. On suppose que le MSS est de 12000 bits.

1. A envoie des données à B au débit maximal autorisé par **slow start**. Combien de données A a-t-il envoyé au bout de 10ms ? 20 ms ? 30 ms ? 1s ?
2. Même question si A utilise l'algorithme **congestion avoidance**.
3. Pourquoi TCP change-t-il d'algorithme dès la première perte de paquet ?
4. Après un temps de silence (un temps pendant lequel il n'a rien eu à émettre) de plus d'un RTT, TCP réinitialise sa fenêtre à 1MSS et recommence à utiliser **slow start**. Pourquoi ?
5. Que pensez-vous de cet algorithme ?