TD 5

CONTROLE DE FLUX ET ORDONNANCEMENT DES ROUTEURS

0. QUELQUES PETITES QUESTIONS

0.1 Paramètres de QoS

Listez trois paramètres de la qualité de service. Pour chacun, donnez un exemple et décrivez, en deux ou trois phrases, son effet sur un flux multimédia en temps réel.

0.2 Moindre effort

L'Internet est un réseau dit de « moindre effort ». Listez quatre effets sur les paquets d'un tel réseau.

0.3 Comment offrir la QoS ?

Il y a trois moyens de fournir une qualité de service : (1) surdimensionner le réseau, (2) introduire des protocoles qui garantissent de la qualité de service, (3) utiliser des applications adaptatives. Décrivez les avantages et les inconvénients de chacun, en une ou deux phrases chacun.

1. CONTROLE DE FLUX

1.1 Rappels

Un émetteur risque de déborder la mémoire de son récepteur et/ou la mémoire des routeurs dans le réseau quand il envoi des paquets trop rapidement. Un moyen d'éviter ce problème est d'imposer des limites sur l'émetteur. Quand il s'agit de limites qui visent la protection du récepteur, on les appel des mécanismes de **contrôle de flux**. Quand le but est d'éviter de la congestion dans le réseau, on parle des mécanismes de **contrôle de congestion** (dans ce contexte on peut également parler de contrôle de flux si on prend soin d'éviter de l'ambiguïté).

Dans votre cours de réseaux en licence vous avez vus comment TCP effectue le contrôle de flux. Le nombre d'octets envoyés dans le réseau est limité par une valeur envoyée par le récepteur. Ceci est un exemple de contrôle de flux en **boucle fermée**. Il n'est pas toujours possible ou souhaitable d'utiliser une boucle fermée pour le contrôle de flux. La boucle fermée peut être trop lente, peut demander trop de signalisation, ou peut demander des mécanismes trop complexes dans les routeurs. Dans ces cas, on a intérêt à s'en servir de contrôle de flux à **boucle ouverte**.

Dans le contrôle de flux à boucle ouverte, une source négocie préalablement le débit auquel elle va émettre. Le réseau et le récepteur acceptent ou non un flux proposé. S'il est accepté, le réseau et le

récepteur réservent les ressources nécessaires pour accepter le flux, et la source est censée rester dans les limites du débit négocié. C'est l'approche utilisée par IntServ et par DiffServ.

Pour négocier le débit, il faut le décrire. On pourrait tout simplement spécifier un débit par un nombre d'octets par unité de temps, par exemple, 64 kb/s. Mais une telle description n'est pas assez précise. Par exemple, si on émet zéro octets pendant dix secondes, est-ce qu'on a le droit d'émettre 1,28 Mo pendant les prochaines dix secondes ? Le moyen sur vingt secondes est 64 kb/s mais pendant dix secondes on a émis à 128 kb/s. On ne peut pas être sûr si on a respecté les limites ou pas.

On va étudier différentes manières de décrire les flux : le seau percé et le seau à jetons. Notre définition de ces mécanismes est basée sur la section 5.3.3 du livre de Andrew S. Tanenbaum [Andrew S. Tanenbaum. *Computer Networks*, 3rd edition. Prentice Hall (1996). A noter : la troisième édition existe également en traduction français. Une quatrième édition en anglais vient de sortir en 2002.]

2. SEAU PERCE

2.1 Rappels

Le **seau percé**, en anglais « leaky bucket », a été introduit par Jonathan S. Turner [Turner, J. S. « New Directions in Communications ». *IEEE Communications Magazine*, vol. 24, pp. 8 – 15. October 1986.] C'est un modèle très simple. On imagine que les paquets sont des gouttes d'eau. L'émetteur, au lieu d'envoyer les paquets directement, doit les placer dans un seau. Ce seau est percé dans une manière qui permet des gouttes de tomber à une vitesse r, exprimée en paquets par seconde. Si la taille des paquets est limitée, on limite ainsi le débit. Par exemple, si la taille maximale d'un paquet est de 512 o, et les gouttes tombent à une vitesse de $r = 1$ paquet/sec, le débit est limité à $512 \times 8 \approx 4$ kb/s.

Il est à noter que les gouttes inutilisées ne sont pas reportées ou réutilisées par la suite.

2.2 Exercices

Donner deux arguments pour lesquels l'algorithme du seau percé peut être plus efficace qu'un algorithme permettant l'envoi de la même information en paquets plus courts en mêmes unités de temps.

3. SEAU A JETONS

3.1 Rappels

Le seau percé règle dans une manière très stricte l'envoi de paquets. Il ne permet pas, par exemple, l'envoi de rafales de paquets dans le réseau. Ca pourrait être bon (voir les exercices précédentes) mais ça pourrait être trop restrictive aussi. Pour permettre l'envoi de rafales de paquets, on peut employer une variante de l'algorithme du seau percé qui s'appelle le **seau à jetons** (« token bucket » en anglais).

Un émetteur place toujours ses paquets dans le seau. Mais le départ de paquets du seau n'est plus contrôlé par un simple flux périodique de gouttes. Il existe au bout du seau un mécanisme à jetons. Pour chaque jeton qu'il y a dans le mécanisme, on a le droit d'envoyer un paquet. Le débit est réglé en fonction de trois paramètres : la fréquence r d'arrivée de jetons dans le mécanisme (en jetons par

seconde), et la capacité maximale B du mécanisme (en jetons), et le débit maximale p de transmission (en paquets par seconde).

Avec le seau à jetons, les jetons pas utilisés restent dans le seau et peuvent être utilisés par la suite. Pourtant, les jetons qui arrivent dans un mécanisme déjà rempli à sa capacité maximale B doivent être jetés.

3.2 Exercices

3.2.1 Donnez des exemples d'applications qui pourraient bénéficier d'un contrôle de flux à seau à jetons au lieu d'un contrôle de flux à seau percé.

3.2.2 Un ordinateur connecté à un réseau est régulé par l'algorithme du seau à jetons, avec les paramètres r = 2K jetons/sec et B = 8K jetons. Le débit du lien sortant et de p = 4K paquets/sec. Au temps t_0, le seau est rempli de c_0 = 9K paquets et le mécanisme de b_0 = 8K jetons. Si l'émetteur est en train de placer des paquets dans le seau à une vitesse de v = 1K paquets/sec, pendant combien de temps l'ordinateur va t-il transmettre à la vitesse maximale du lien sortant ?

4. LIMITATION DE MEMOIRE

4.1 Rappels

La mémoire est une ressource limitée. Pour modéliser les limites de mémoire, on impose une taille maximale sur le seau. Soit C la capacité maximale du seau en paquets. S'il y a C paquets dans le seau, et un nouveau paquet arrive, ce paquet doit être jeté.

4.2 Exercices

Soit r = 10K jetons/sec, B = 1K jetons, p = 100K paquets/sec, et C = 2K paquets. Imaginez qu'une application est capable de placer des données dans le seau à un débit de v = 50K paquets/sec. Au temps t_0, le nombre de paquets dans le seau est $c_0 = 0$ et le nombre de jetons dans le mécanisme est $b_0 = 0$. Pendant combien de temps une rafale de données peut elle être mise en seau sans engendrer de perte de données ?

5. ORDONNANCEMENT DES ROUTEURS

5.1 Rappels

Le contrôle de congestion par les émetteurs n'est pas toujours suffisant pour éviter des problèmes dans le réseau. Par exemple, il peut y avoir des émetteurs mal paramétrés. Dans ce cas, nous aimerons que les routeurs protègent les flux d'autres émetteurs. Si on est prêt à introduire de la complexité dans les routeurs, il est possible d'avoir un ordonnancement qui assurera de l'équité entre des flux.

Le mécanisme d'ordonnancement le mieux connu pour assurer de l'équité s'appelle la mise en file d'attente équitable et pondérée, ou « weighted fair queueing » (**WFQ**) en anglais. Tanenbaum parle de WFQ dans section 5.3.6 de son livre, et nous nous basons là-dessus.

6. WFQ

6.1 Rappels

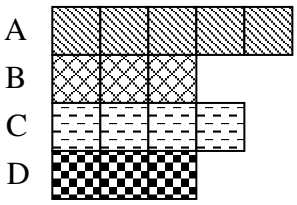
Si les paquets sont tous de la même taille, on peut facilement partager le lien de sortie d'un routeur entre plusieurs flux dans une manière équitable. Il suffit d'appliquer une politique de répartition par permutation circulaire. On envoi un paquet par flux, chacun à son tour, en cycle permanent. S'il y a toujours des paquets de chaque flux en file d'attente, le routeur donnera à chaque flux une partie égale de son débit.

Si on veut que le routeur partage son débit dans une manière pondérée, ça reste simple si les paquets sont toujours tous de la même taille. Par exemple, si on veut partager le débit entre flux A, B, C, et D, avec un poids de 1 pour les flux A, C, et D, et un poids de 2 pour le flux B, on peut appliquer la politique de répartition par permutation circulaire, en prenant deux paquets quand il est le tour de flux B, et un paquet pour chacun des autres flux.

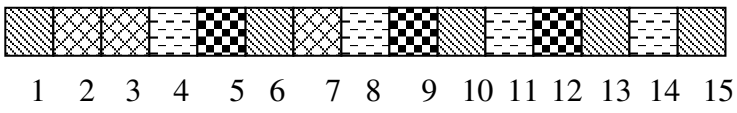
Un problème survient quand les paquets ne sont plus tous de la même taille. Dans l'exemple qu'on vient de citer, le flux A pourrait obtenir un débit supérieur du débit de B en envoyant des paquets d'une taille au moins deux fois plus grande que ceux de B. Le WFQ évite ce problème.

Le WFQ permet de simuler ce qui se passerait si on pourrait découper des paquets en petits morceaux tous de la même taille. Cette simulation permet de trouver l'ordre dans lequel les paquets seraient reconstitués. Au lieu de découper et de reconstituer les paquets, WFQ utilise la simulation afin de décider sur l'ordre dans lequel il doit envoyer des paquets.

Voici un exemple. Il y a des flux A, B, C, et D. Il y a un paquet de chaque flux en file d'attente. Ils sont des longueurs suivantes : 5, 3, 4, 3.

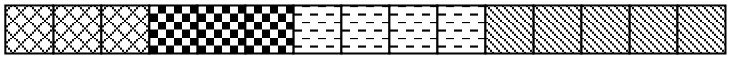


Nous voulons donner un poids de 2 au flux B. Avec WFQ, nous imaginons que les paquets peuvent être découpés en morceaux de taille 1. Le routeur simule un traitement de chaque morceau à son tour, avec un double priorité pour les morceaux de flux 2 ainsi : A B B C D A B C D A C D A C A.



Cette simulation nous donne l'ordre de reconstitution de paquets. Au temps 7, le paquet de flux B serait reconstitué. Au temps 12, le paquet de flux D. Au temps 14, le paquet de flux C. Et, finalement, au temps 15, le paquet de flux A.

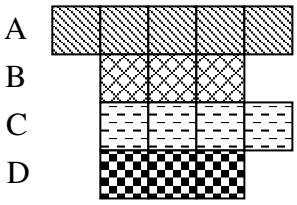
En réalité, il n'est pas pratique de découper des paquets en petits morceaux. La surcharge de fragmentation et reconstitution serait trop élevé. WFQ utilise la simulation simplement pour établir l'ordre dans lequel il doit envoyer les paquets. Le résultat de cette simulation est qu'il doit les envoyer dans l'ordre B D C A :



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

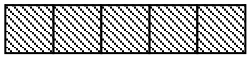
Nous venons de voir ce qui se passe si les paquets sont tous ordonnancés simultanément. Mais en pratique, WFQ réordonnance les paquets chaque fois qu'un paquet arrive dans la file d'attente, et chaque fois qu'un paquet est envoyé. En plus, il n'arrête pas d'envoyer un paquet déjà en cours d'être envoyé.

Par exemple, si le paquet du flux A arrivait au temps 1 et les autres au temps 2, nous aurions le suivant pour les arrivés :



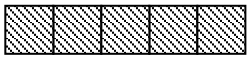
1 2 3 4 5

Temps 1, simulation : WFQ aura simulé A A A A A, avec le temps de finir pour le paquet du flux A au temps 5.



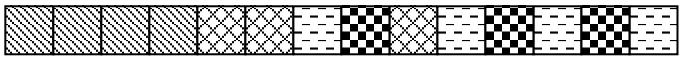
1 2 3 4 5

Temps 1, ordonnancement : L'ordonnancement est pour un seul paquet du flux A, qu'il commence à envoyer.



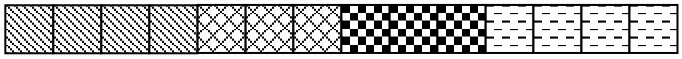
1 2 3 4 5

Temps 2, simulation : WFQ a déjà envoyé un unité du paquet de flux A. Quand les autres paquets arrivent, on ne va pas interrompre le paquet du flux A. Alors WFQ aurait la simulation suivante : A A A A B B C D B C D C D C.



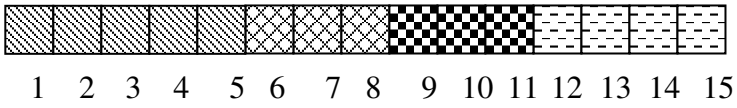
2 3 4 5 6 7 8 9 10 11 12 13 14 15

Temps 2, ordonnancement : Le temps simulé pour la terminaison d'envoi de chaque paquet est 5 (A), 10 (B), 14 (D), 15 (C). Alors le routeur va envoyer les paquets dans l'ordre : A B D C.



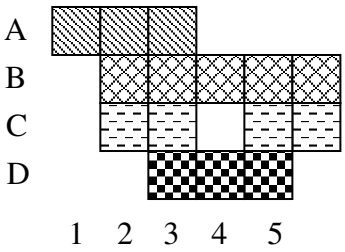
2 3 4 5 6 7 8 9 10 11 12 13 14 15

Résultat : Voici le résultat de WFQ :

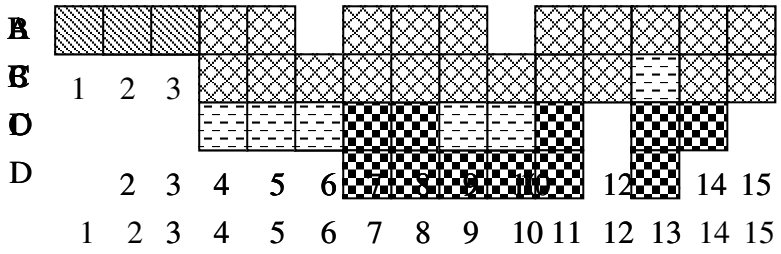


6.2 Exercices

Supposons que les paquets arrivent dans l'ordre suivant, et que le routeur donne un poids de deux au flux B.



A noter : il y a deux paquets de taille 2 chacun qui arrivent dans le flux C. Montrez l'ordre dans lequel les paquets sont envoyés par le routeur.


A
B
C
D
1 2 3
2 3 4 5 6 12 14 15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15