Rapport Réseaux Serveur de chat

Kevin Lanvin

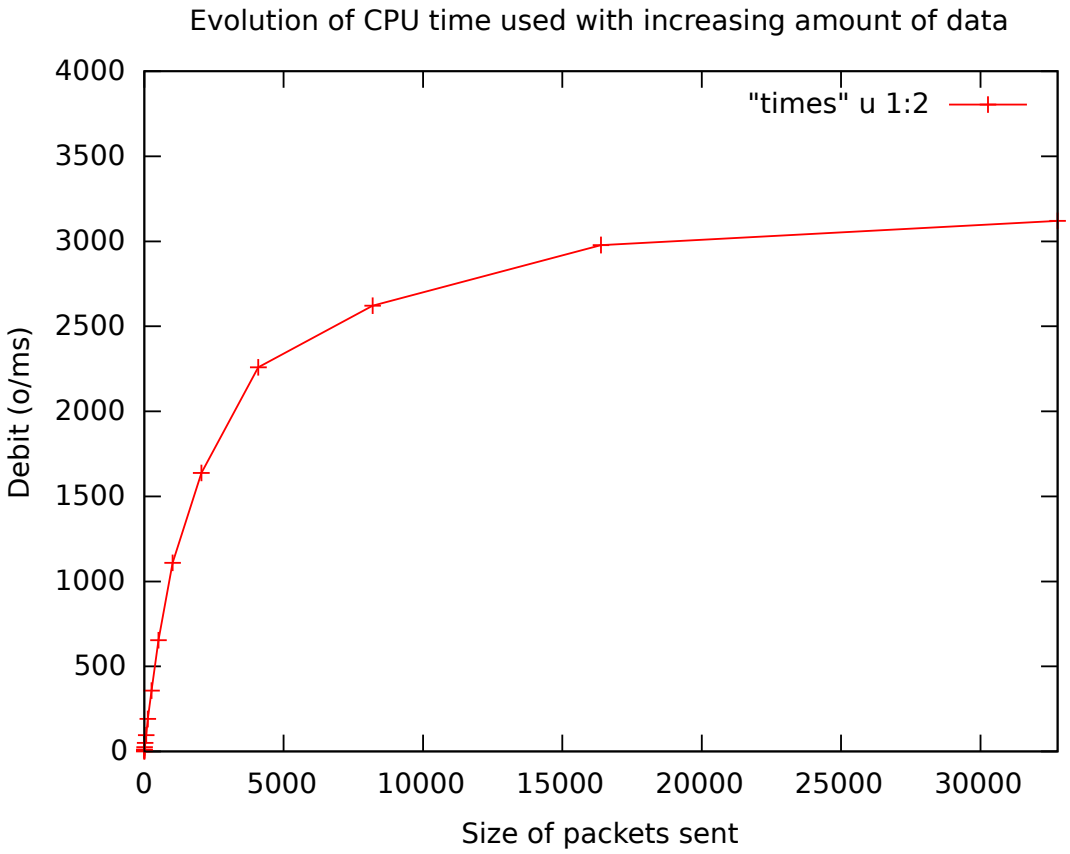
Nous allons ici nous intéresser au débit de données entre un client de chat et un serveur capable de répondre à une commande /echo qui renvoie les données qui lui sont envoyées jusqu'à une limite fixée.

Dans notre exemple, le serveur de chat a été conçu en utilisant les selectors, et lit les octets un à un. Aucune opération sur les String n'est effectuée côté serveur.

Côté client, pour obtenir des graphiques cohérents et plus représentatifs, chaque test est lancé un certain nombre de fois. Les tests pour une même taille de paquet ne sont pas effectués à la suite car une baisse de vitesse d'exécution de l'ordinateur temporaire affecterait alors tous les relevés pour la même taille de paquet, mais pas pour les autres. Une moyenne de tous les relevés pour la même taille de paquet est ensuite effectuée avant de dessiner la courbe.

Les tests ont été effectués en local, ce qui ne permet malheureusement pas de mettre en évidence la surcharge du réseau et donc l'impact exact d'une connexion en plus ou en moins.

Observation globale pour 65 536 octets envoyés :



Interprétation du graphique :

Cette courbe montre que l'impact de la taille des paquets envoyés est beaucoup plus important quand celle-ci est faible, et négligeable quand celle-ci est grande. Autrement dit : les connexions et déconnexions prennent beaucoup de temps. Multiplier le nombre de connexions multiplie également le temps de transmission total.

Avec wireshark, on remarque que la connexion et la déconnexion correspondent à plus de la moitié des paquets envoyés lors d'un échange entre le serveur et le client. Les données représentent à peine la moitié des paquets. Le trafic est le suivant pour chaque connexion :

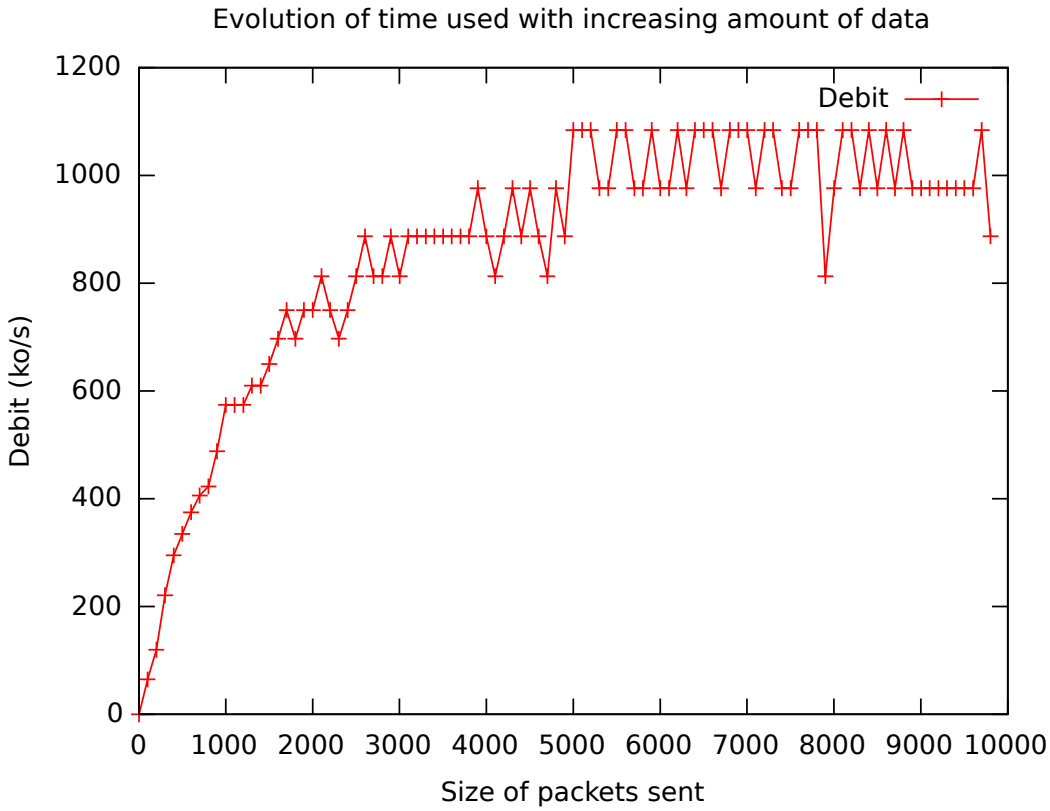
1. → Demande de connexion SYNC
2. ← ACK de connexion , SYNC
3. → ACK SYNC
4. → Envoi du /echo
5. ← ACK réception de la commande
6. ← Réception du « echo : » (je n'ai pas pu supprimer cette partie verbeuse)
7. → ACK echo :
8. → Envoi d'un octet
9. ← ACK réception de l'octet + OK echo
10. → ACK OK echo + FIN connexion
11. ← ACK FIN
12. → ACK ACK FIN

On a donc 12 transmissions. Quand on multiplie les connexions, on envoie donc beaucoup plus de données que prévu. La mise en place de la connexion en elle-même requiert 6 paquets, soit la moitié de la conversation avec le serveur.

De même , pour des paquets de très petite taille, on envoie un paquet avec une entête remplie, pour n'envoyer qu'un ou deux octets de données. Ces informations sont également des données qui prennent de la bande passante mais qui ne sont pas des données utiles.

L'impact de la taille des paquets, et donc du nombre de connexions effectuées pour envoyer nos données n'est donc pas à négliger. Il faut privilégier des paquets de grande taille, afin d'augmenter le rapport : données utiles/données totales .

Observations pour 10 000 octets envoyés, avec un échantillonnage plus précis :



Nombre total d'occtets envoyés : 10 000.
Pas entre chaque taille de paquet testé : 100.
Nombre de tests faits pour chaque valeur taille de paquet : 10.

Après de nombreux tests, il semblerait que les fluctuations que l'on peut voir sur cette courbe soient dues à l'environnement et qu'elles n'aient pas de rapport avec l'application testée. En effet, si on renouvelle le test, on n'obtient pas de pic ni de creux aux mêmes endroits.

Analyse du programme :

Cependant, je m'attendais à ce que la courbe soit en escalier. En effet, lorsqu'on atteint un diviseur du nombre total d'octets envoyés, (ici 10 000), alors les octets sont équitablement répartis entre tous les paquets envoyés.

Mais si on prend « size » (taille des paquets) tel que : *size=10000/k – 1* , où k est un entier naturel inférieur ou égal à 10000, alors cela voudra dire qu'on aura une connexion qui n'enverra que peu d'octets.

Cela ne pose pas de problème pour une petite valeur de size. En effet, si size = 3 (k=2500), alors on a 3333 connexions qui enverront 3 octets de données et une qui enverra un octet. Cela est négligeable car elle ne représente que 1/3334eme de toutes les connexions.

Mais si on prend size = 4999 (k=2), alors 2 connexions enverront 4999 octets de données, et la dernière en enverra seulement 2. Ici cela pose problème car 1/3 des connexions utilisées ne sert presque à rien. On augmente le débit de moitié quand on passe de 4999 octets par paquets à 5000.

A l'inverse, quand on prend *size = 10 000/k +1* , cela ne change pas le débit pour des grandes valeurs de size, car on va juste remplir de moins en moins la dernière connexion. Autant de données réelles seront donc envoyées sur le réseau. Cela va continuer jusqu'à ce qu'on puisse de nouveau retirer une connexion et ainsi réaugmenter le débit.

Voilà pourquoi la courbe devrait être en escaliers, et non linéaire.