Systèmes et Réseaux (ASR 2) - Notes de cours
Cours 14

Anne Benoit

May 12, 2015

PARTIE 1: Systèmes

PARTIE 2: Réseaux

1 Architecture des réseaux de communication

2 La couche 2-liaison

3 La couche 3-réseau

4 Algorithmes de routage

5 La couche 4-transport

Couche qui assure un transport de bout en bout (de type logique), sans se soucier des routeurs intermédiaires (qui ne gèrent que les couches 1-2-3). Socket: interface de connection entre l'application et la couche transport (cf TDs).

5.1 Gestion des erreurs

Où gérer les erreurs? Toute couche de niveau supérieur à 2 doit offrir un service sans erreurs à la couche du dessus: les paquets sont livrés sans erreurs ou supprimés.

Exemple de la couche MAC: comment savoir si une trame Ethernet a des erreurs? Que faire d'une trame fautive? Et en Wifi?

Les erreurs sont donc (en général) transformées en pertes de paquets. Autres causes de pertes de paquets: débordement de buffers dans les ponts et routeurs, TTL qui expire (oscillation de routes des algorithmes de routage), ... Il faut réparer ces pertes de paquets.

Deux stratégies possibles:

- De bout en bout (*end-to-end*): A envoie ses paquets à B, et B redemande les paquets manquants. Permet de garder la simplicité: les routeurs ne voient pas forcément passer tous les paquets de A vers B (routes différentes), et c'est coûteux de conserver tous les paquets en mémoire au cas où il faudrait les retransmettre.

- A chaque saut (*hop-by-hop*): Chaque routeur fait ce travail (et demande les paquets manquants). Egalement des avantages: si le taux de pertes de paquets est constant, $p \in [0,1]$, alors un canal avec un débit R a une capacité réelle de $R(1-p)$ paquets/secondes.

  BER: *bit error rate*. Si les erreurs de bits sont indépendantes, alors le taux de pertes de paquets, PLR (*packet loss rate*) est $PLR = 1-(1-BER)^L$, où L est la longueur du paquet (en bits).

Avec k liens qui ont un taux de pertes de p, chaque saut à une capacité de $R(1-p)$, soit $C_{hh} = R(1-p)$.

Si l'on considère une correction de bout en bout, le taux d'erreurs sur les paquets est $p' = 1-(1-p)^k$ (un paquet arrive si et seulement si il ne s'est perdu sur aucun lien), d'où une capacité $C_{ee} = R(1-p)^k$ au lieu de $R(1-p)$.

Exemple:

k	PLR	C_{ee}	C_{hh}
10	0.05	$0.6R$	$0.95R$
10	0.0001	$0.9990R$	$0.9999R$

Ainsi, si le PLR est élevé, la correction de bout en bout n'est pas acceptable. Comment faire pour concilier les deux?

Faire à chaque saut sur les liens à fort taux d'erreur (Wifi mais pas Ethernet): récupérer des erreurs au niveau de la couche MAC, afin d'offrir un faible taux d'erreurs aux couches supérieures. Et en plus, récupération de bout en bout au niveau des hôtes.

5.2 Protocoles ARQ

Comment réparer les pertes de paquets? A-t-on déjà vu un tel protocole?

Protocole stop-and-go: rajouter éventuellement des numéros de séquence aux paquets, l'hôte destinataire envoie un ACK pour chaque paquet. Si pas d'ACK au bout d'un certain temps, l'hôte source retransmet le paquet.

Stop-and-go est un exemple de protocole de retransmission de paquets, c'est la méthode utilisée dans Internet pour réparer les pertes de paquets. On parle de protocoles ARQ (*Automatic Repeat reQuest*). TCP est un protocole ARQ: récupère les paquets perdus et transmet les paquets dans l'ordre.

Quelle est la limite de Stop-and-go? Si le produit bande-passante délai n'est pas très petit, alors le débit est faible, car on perd du temps en attendant un ACK. Exemple: $t = 8\mu sec/paquet$, temps d'aller-retour de 30 ms $\rightarrow$ 1 paquet toutes les 30.008 ms. Le taux d'utilisation du lien est $U = 0.008/30.008 = 0.00027$. Ainsi, sur un lien à 1 Gb/s, on n'utilise que 267 kb/s.

Solution: pipeliner les envois en permettant plusieurs envois avant de recevoir les ACKs. Nécessite un grand buffer au niveau du destinataire, mais on contrôle cette taille de buffer en fixant le nombre d'envois simultanés, on parle de *fenêtre coulissante* (qui a une taille fixée).

Voir exemples de protocoles ARQ:

1. Fenêtre coulissante. Paquets numérotés, taille de fenêtre W (jamais plus de W paquets sans ACK côté destinataire). Fenêtre offerte: paquets envoyés mais non ACK, ou qui peuvent être envoyés. Fenêtre utilisable: paquets que l'on peut envoyer

pour la première fois. La fenêtre se déplace vers la droite (coulisse) lorsque l'ACK pour le premier paquet de la fenêtre est reçu.

S'il n'y a pas de pertes, débit égal au débit du lien si la taille de la fenêtre est telle que $W \geq \beta/L$, où β est le produit bande-passante délai, et L la taille des paquets. Taille de fenêtre min (en termes de bits) égale au produit bande-passante délai.

2. Répétition sélective. ACK positifs qui indiquent les paquets reçus. Détection des pertes au niveau de la source: timeout si pas d'ACK reçu. Lors d'une perte, on retransmet le paquet qui était perdu.

3. Go-back-N. ACK positifs et cumulatifs: si ACK i, alors tous les paquets de numéro inférieur à i ont été correctement reçus. Détection des pertes au niveau de la source par timeout, et retransmission en partant du dernier paquet pour lequel on avait reçu un ACK. Moins efficace que la répétition sélective, mais plus simple, et le buffer destination n'a besoin que d'une taille 1.

4. Go-back-N avec ACKs négatifs. ACKs positifs cumulatifs, et ACKs négatifs: paquets jusqu'à i ok, mais une perte après cela. Détection de perte par timeout ou par réception d'un ACK négatif. Perte: Go-back-N (retransmettre depuis le denier paquet avec ACK).

Où sont utilisés les protocoles ARQ?

- A chaque saut au niveau de la couche MAC: Répétition sélective sur les modems, et stop-and-go en Wifi.

- De bout en bout: couche transport et TCP: variante de répétition sélective avec un peu de go-back-N; couche application: DNS utilise stop-and-go.

Alternatives à ARQ? On peut utiliser des codes pour réparer les paquets. Si la donnée est constituée de n paquets, rajouter k paquets redondants, de façon à ce que l'on puisse reconstruire la donnée à partir de n paquets parmi les $k + n$ paquets.

Avantages: pas de retransmission (intéressant si délai important), et c'est bien pour le multicast car des destinations différentes peuvent perdre des paquets différents. Par contre, moins bon débit car on ajoute de la redondance même si ce n'est pas nécessaire. On peut combiner FEC avec ARQ: codage si on pert un paquet.

5.3 Contrôle de flot

Différences de performance entre machines: si A envoie ses données trop rapidement à B, B ne peut pas les consommer assez rapidement. On peut alors avoir des pertes de paquets si le buffer de B déborde.

Données en provenance de la couche IP $\rightarrow$ buffer (place libre et données TCP) $\rightarrow$ appli

But: empêcher les débordements de buffer du récepteur. Différent du contrôle de congestion (au niveau IP) qui cherche à éviter les pertes de paquets dans le réseau.

Deux techniques principales (voir transparents):

1. Contrôle de flot avec pression retour: le récepteur envoie des messages STOP (pause) ou GO. Après réception d'un STOP, la source arrête de transmettre pendant x msec. C'est très facile à implémenter. Marche bien si le produit bande-passante délai est petit. Utilisé dans les protocoles X-ON / X-OFF et entre les ponts d'un LAN.

2. Utilisation de fenêtre coulissante: le récepteur n'envoie un ACK des paquets que lorsqu'ils ont été consommés par l'application, lorsqu'il a la place d'en recevoir des nouveaux. Permet d'éviter les débordements de buffers, mais par contre la source peut être amenée à renvoyer des paquets sans ACK alors qu'ils avaient déjà été reçus correctement.

3. Méthode des crédits. Le récepteur indique combien de données il est prêt à recevoir. Les crédits (ou fenêtre annoncée) sont envoyés en même temps que les ACKs, qui sont cumulatifs. Ils font bouger la frontière droite de la fenêtre (les ACKs bougent la frontière gauche): ACK n et crédit k: le récepteur a assez de place pour les paquets $n+1$ à $n+k$. A priori, $n+k$ ne doit pas décroître (des paquets ont pu être envoyés avant que le crédit soit reçu). La source est bloquée si le crédit est 0 (ou égal au nombre de paquets sans ACKs).

   Les crédits sont directement liés au nombre de places disponibles dans le buffer. En cas de perte, le crédit est toujours égal au nombre de places dans le buffer moins le nombre de paquets reçus dans l'ordre.

5.4 La couche transport: UDP vs TCP

Couches physique + liaison + réseau: transportent les paquets de bout en bout. La couche transport, uniquement au niveau des hôtes, rend les services réseaux disponibles aux applications. Deux protocoles de transport: UDP (User Datagram Protocol) qui est uniquement une interface de programmation pour accéder aux services réseau, et TCP (Transmission Control Protocol), qui gère les erreurs et le contrôle de flot.

5.4.1 Le protocole UDP

Utile pour les applications qui n'ont pas besoin de retransmission. Exemples:

- Applications de téléphonie: c'est le délai qui est important, et la retransmission de paquets perdus fait augmenter le délai!

- Application qui n'envoie qu'un seul message (DNS par exemple pour la résolution de noms): plutôt que de payer le surcoût de TCP (plusieurs paquets avant d'envoyer les données), utilisation de stop-and-go au niveau couche application.

- Multicast, non supporté par TCP.

UDP utilise des numéros de port: l'entête contient les numéros de port source et destination, les adresses IP, la longueur du paquet, et un checksum. Envoi du message entier dans un paquet UDP, jusqu'à 8K. Pas de garantie de séquence sur la réception des

messages, et des messages peuvent se perdre. MTU: Maximum Transmission Unit. Si un message est trop gros, il est fragmenté au niveau de la couche IP.

Bibliothèque de sockets, liées aux numéros de ports. Pas de connexion: une fois la socket créée, le serveur peut recevoir des messages sur son port, et le client envoie des messages.

5.4.2 Le protocole TCP

Garantit que les données sont délivrées dans l'ordre, sans pertes, sauf si la connection est coupée. Marche pour toute application qui envoie des données, mais pas avec le multicast. En plus, contrôle de flot et contrôle de congestion (pas détaillé ici).

Protocole avec connection: les 2 parties doivent se synchroniser (synchroniser les numéros de séquence) avant de commencer à échanger des données. ARQ pour les pertes, et crédits pour le contrôle de flot. La connection est fermée lorsque la communication est terminée. Bibliothèque de sockets avec connection.

Utilisation de numéros de ports comme UDP. Entête TCP + Données TCP = segment TCP (= données IP), auquel la couche IP rajoute un entête. Les octets à envoyer sont stockés dans un buffer en attendant que TCP décide de créer un segment. MSS: maximum segment size, 536 octets par défaut (paquets IP de taille 576). Numéros de séquence basés sur le numéro d'octet, et non sur le numéro du paquet. Les segments ne sont jamais fragmentés au niveau de la source.

Protocole ARQ: hybride entre go-back-N et répétition sélective: fenêtre coulissante, ACKs cumulatifs, détection de pertes par timeout au niveau de la source. En plus, retransmission rapide et ACKs sélectifs.

- Opération de base (voir transparent): communication bi-directionnelle. Notation:

  firstByte:lastByte+1(SegmentDataLength) ack AckNb+1 win OfferedWindowSize

  Ligne 8: expiration du timer, retransmission. Le récepteur a gardé les paquets ultérieurs, et peut donc faire à réception un ACK de 10001. Numéro de séquence initial choisi aléatoirement entre 0 et $2^{32} - 1$. Les timers sont réinitialisés après timeout (implémentation TCP SACK).

- Retransmission rapide (voir transparent): mécanisme pour découvrir les pertes avant le timeout (souvent trop grand): si 3 duplicatas d'un ACK sont reçus avant le timeout, alors retransmettre. Quel ACK est envoyé sur la figure?

- ACKs sélectifs (SACKs): mécanisme plus adapté en cas de multiples pertes (retransmission rapide bien pour pertes isolées), qui envoie des ACKs précisant exactement les octets reçus (ACKs positifs d'un interval d'octets). La source peut détecter un trou et retransmettre les octets manquants.

La source peut envoyer un octet même avec un crédit de 0, et tester ainsi la fenêtre du récepteur, afin d'éviter les interblocages en cas de perte de crédit.

Mécanismes additionnels.

- Quand envoyer un ACK? Peut être envoyé immédiatement, ou attendre d'avoir des données à envoyer ou de recevoir plus de segments (ACKs cumulatifs). Retarder l'envoi des ACKs permet de diminuer les surcoûts et le nombre de paquets IP, mais retarde les temps de réaction.

  Algorithme: retarder d'au plus 0.5 sec l'envoi d'un ACK. Segments complets: au moins un ACK par segment. Envoi d'un ACK même si le segment reçu n'est pas dans l'ordre (ACK du dernier octet reçu dans l'ordre +1).

- Algorithme de Nagle: grouper des petites données en segments, afin d'éviter de transmettre plein de petits segments. Par contre, rajoute du délai! Cet algorithme décide quand créer un segment et le transmettre via la couche IP. Principe: accepte un seul segment plus petit que MSS sans ACKs.

  Algorithme: lors de nouvelle donnée qui arrive de la couche supérieure ou à réception d'un ACK, si un segment plein est prêt, l'envoyer. Sinon, s'il n'y a pas de données sans ACK, alors envoyer un segment. Sinon, démarrer un timer et attendre; à expiration du timer, créer un segment et l'envoyer.

  L'algorithme peut être désactivé par l'application.

  Exemple (voir transparent) avec des données qui arrivent une par une...

- Silly Window Syndrome avoidance. Ce syndrome SWS apparaît quand le récepteur est lent ou occupé: si la source a beaucoup de données à envoyer, elle se retrouve à l'envoyer en plein de petits paquets si les fenêtres annoncées sont petites, ce qui est une perte de ressources (exemple à gauche).

  Solution: empêcher le récepteur d'envoyer des ACK avec des petites fenêtres.

  ReceiveBuffer: contient octets entre PlusGrandLu et ProchainAttendu, puis la fenêtre offerte, puis une réserve. Une nouvelle annonce est faite uniquement quand la réserve est de taille supérieure à min(MSS, 1/2 ReceiveBuffer). Voir exemple sur les transparents, où l'on voit aussi un envoi suite à une expiration du timer pour vérifier que le récepteur ne veut pas de données.

- SWS avoidance au niveau de la source = Nagle: Nagle + SWS avoidance veulent tous les deux éviter l'envoi de petits paquets (côté source ou côté récepteur).

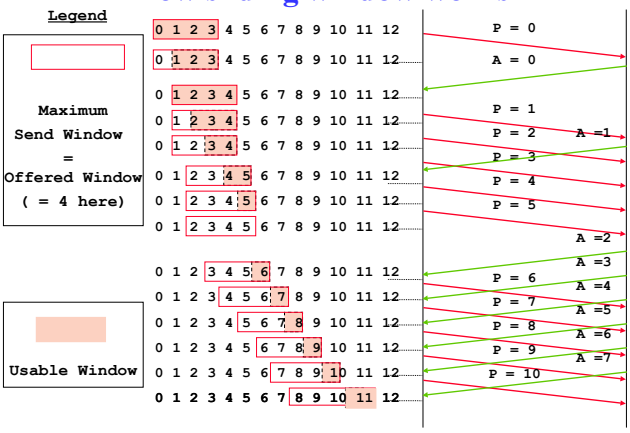
5.5 Conclusion

TCP: fournit un service fiable au programmeur. Complexe et complexe à utiliser, mais puissant: marche bien avec tout type d'applications. Implémente aussi le contrôle de congestion, mais nous n'aurons pas le temps d'en parler cette année!

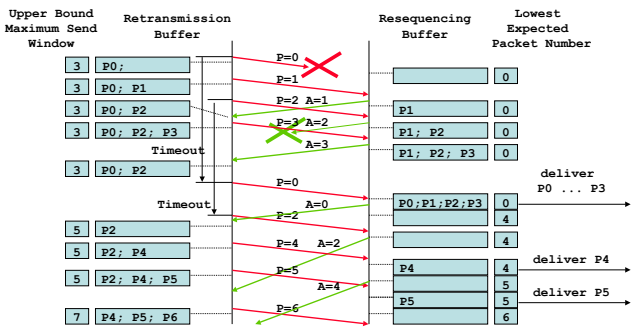
Couche transport (TCP)

- Protocoles ARQ
- Contrôle de flot
- TCP

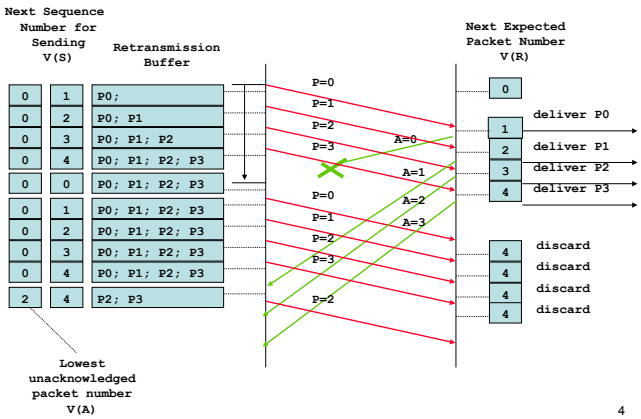
How sliding window works



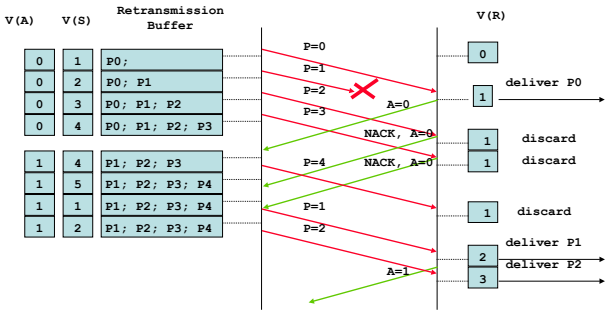
An example of ARQ protocol with selective repeat



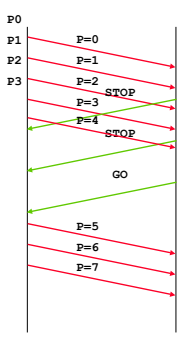
An example of ARQ protocol with Go Back N



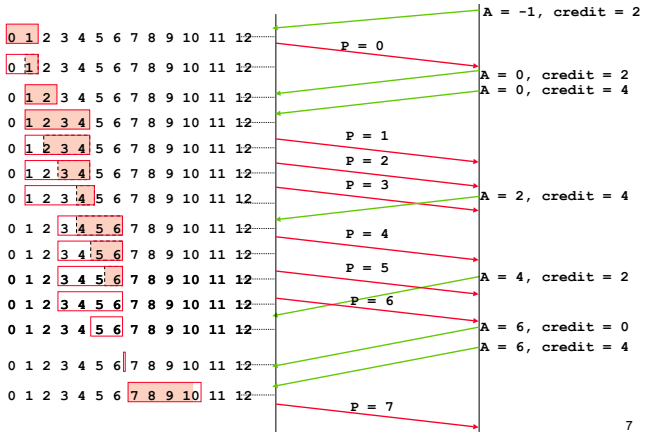
An example of ARQ protocol with Go Back N and negative acks



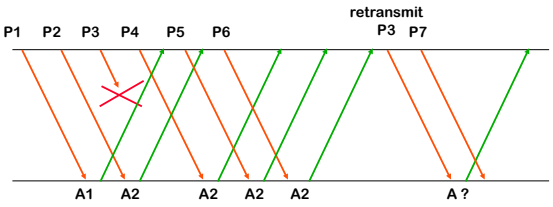
Backpressure flow control



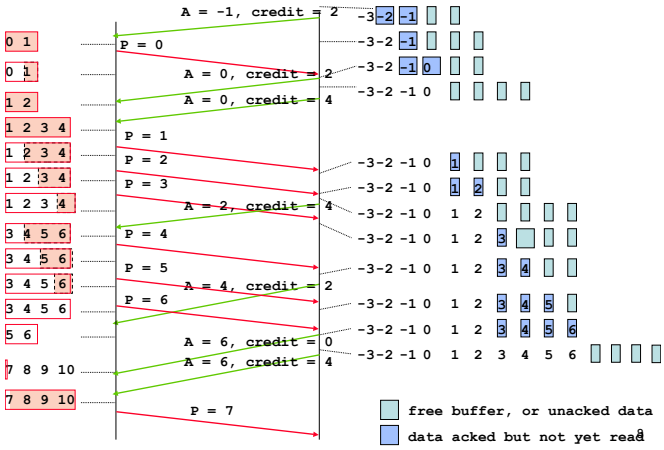
Credit flow control



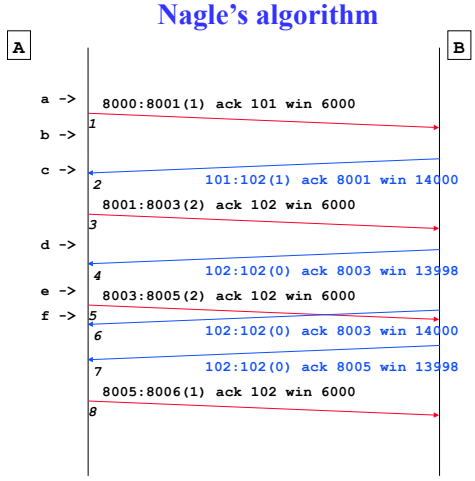
TCP: "Fast Retransmit"



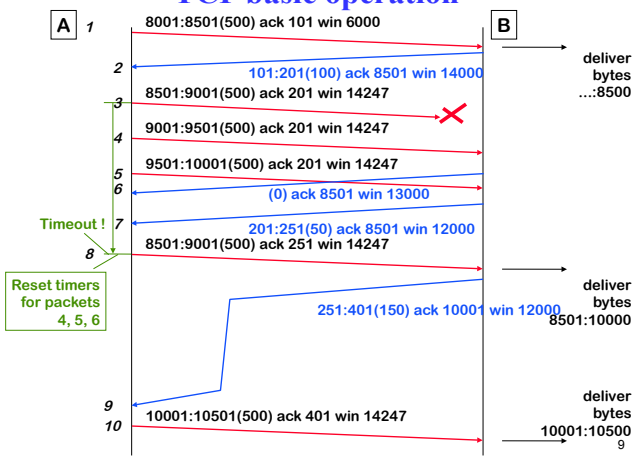
Credits are modified as receive buffer space varies



Nagle's algorithm



TCP basic operation



Silly Window Syndrome avoidance

```
ack 0, win 2000 <-----
     0:1000  -----> bufferSize= 2000B,
                    freebuf= 1000B
  1000:2000  -----> freebuf= 0B
ack 2000, win 0 <-----
   application reads 1 Byte: freeBuf = 1
ack 2000, win 1 <-----
   2000:2001  ----->  freeBuf = 0
   application reads 1 Byte: freeBuf = 1
ack 2001, win 1 <-----
   2001:2002  ----->  freeBuf = 0
   application reads 1 Byte: freeBuf = 1
ack 2002, win 1 <-----
   2002:2003  ----->  freeBuf = 0
```

```
ack 0 win 2000  <-----
     0:1000  ----->   bufferSize= 2000B, freebuf = 1000B,
                      reserve = 0B
  1000:2000  ----->   freebuf= 0B, reserve = 0B
ack 2000, win 0 <-----
                application reads 1B: freeBuf=reserve=1B,
               ....
                application has read 500B: reserve = 500B
persistTimer expires
window probe packet sent

   2000:2001  ----->
                data is not accepted (out of window)
 ack 2000, win 0 <-----
                ....
                 application has read 1000B: reserve = 1000B
ack 2000, win 1000 <-----
     2000:3000 ----->
```