

Protocole TCP

(Transmission Control Protocol)

M1 Info / 2020-2021 – Cours de Réseaux sans fil

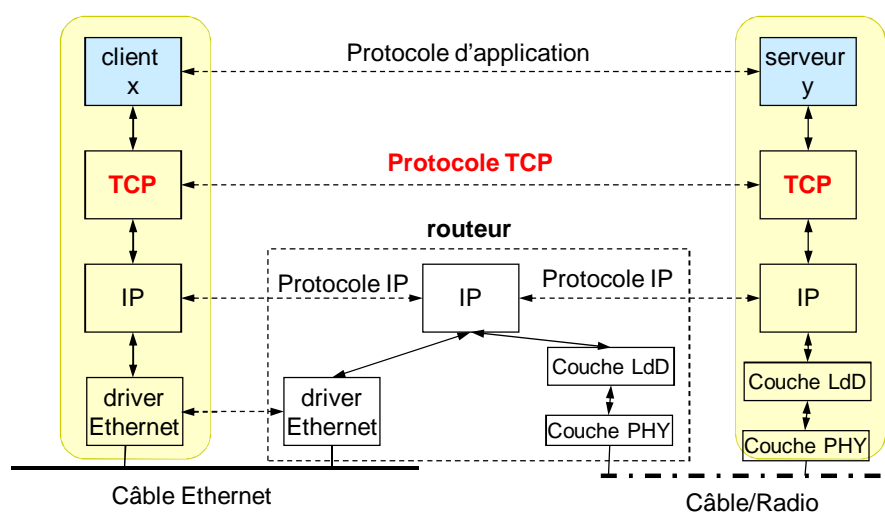
Z. Mammeri

Protocole TCP – M1 Info / 2020-2021 – Z. Mammeri - UPS

1

1. Généralités

Architecture TCP/IP

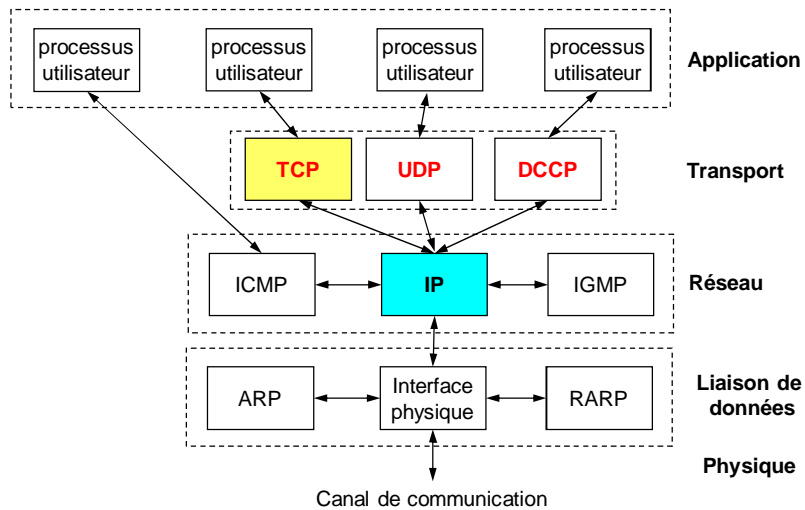


Protocole TCP – M1 Info / 2020-2021 – Z. Mammeri - UPS

2

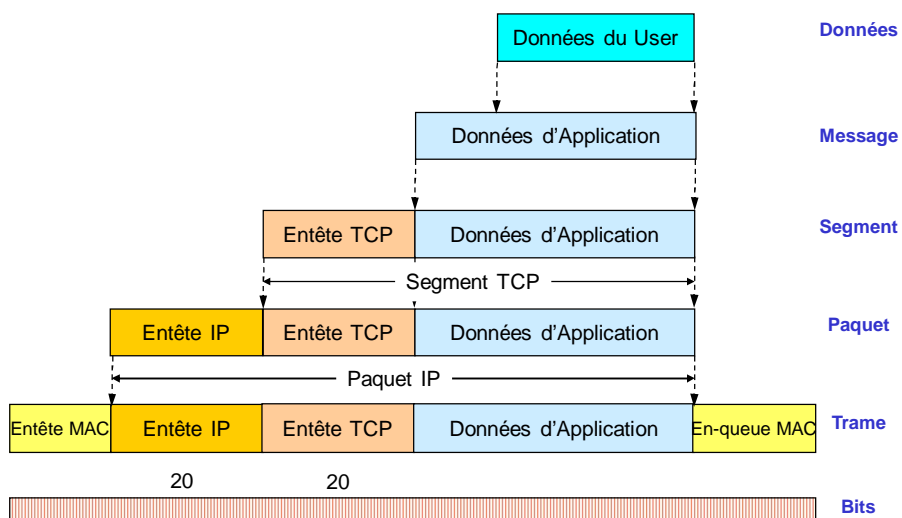
1. Généralités

Architecture TCP/IP



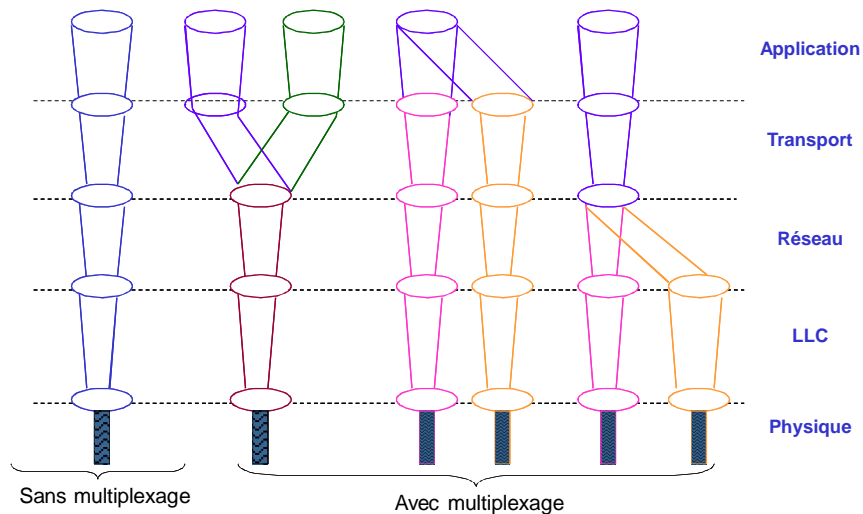
1. Généralités

Encapsulation



1. Généralités

Multiplexage (contexte général)



Protocole TCP – M1 Info / 2020-2021 – Z. Mammeri - UPS

5

1. Généralités

Fonctions relevant du niveau Transport

- **Transport de bout en bout** de messages
- **Fragmentation/réassemblage** de messages
- **Contrôle d'erreurs** (perte de segments, erreur d'entête TCP)
- **Contrôle de flux** (contrôle aux extrémités)
- **Contrôle congestion** (contrôle à l'intérieur du réseau)
- **Séquencement** de de paquets et livraison ordonnée de messages
- **Multiplexage** de connexion
- **Sécurité** (si les niveaux inférieurs ne sont pas sécurisés)

Protocole TCP – M1 Info / 2020-2021 – Z. Mammeri - UPS

6

1. Généralités

Protocoles de transport et QoS

● Paramètres de QoS

- Temps d'établissement de connexion
- Probabilité d'échec de connexion
- Débit de la connexion
- Temps de transit de bout en bout
- Taux d'erreurs
- Sécurisation de la connexion
- Priorité pour les données urgentes ou critiques
- Probabilité de résiliation de connexion par le provider

● QoS : non gérée actuellement par TCP, UDP...

1. Généralités

Protocoles de transport OSI

● OSI a défini 5 classes de TP (Transport Protocol)

- TP0 : Pas de multiplexage, Pas de reprise sur erreurs
- TP1 : Pas de multiplexage, Reprise sur erreurs signalées par le niveau Réseau
- TP2 : Multiplexage, Avec ou sans contrôle de congestion, Pas de reprise sur erreurs
- TP3 : Multiplexage, Avec ou sans contrôle de congestion, Reprise sur erreurs signalées
- TP4 : Multiplexage, Avec ou sans contrôle de congestion, Reprise sur erreurs signalées et non signalées

● TP classe 0 : adapté aux réseaux fiables

● TP classe 4 : adapté aux réseaux non fiables (ex. IP)

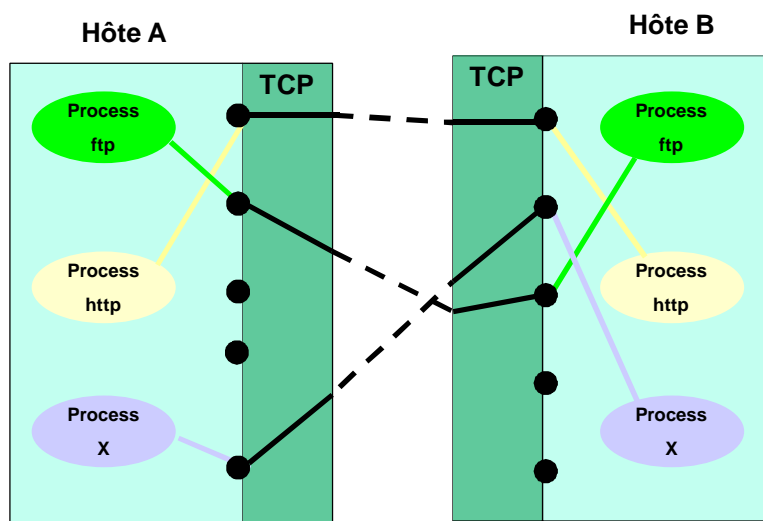
1. Généralités

Protocoles de transport TCP/IP

- **UDP** (*User Datagram Protocol*) ~ **TP classe 0**
 - Défini en 1980 (RFC 768)
 - Très simple : aucun contrôle d'erreurs ou de congestion
 - Adapté aux réseaux fiables ou bien si la fiabilité n'est pas requise
 - **Agressif pour le réseau (peu recommandé en général)**
- **TCP** (*Transmission Control Protocol*) ~ **TP classe 4**
 - Défini en 1981 (RFC 793)
 - Complexe : connexions fiables, contrôle de flux, contrôle de congestion
 - Adapté aux réseaux non fiables
 - **Très adaptatif (très recommandé en général)**
- **DCCP** (*Datagram Congestion Control Protocol*) ~ **TP classe 2**
 - Défini en 2006 (RFC 4336)
 - Complexité moyenne : connexions non fiables, avec contrôle de congestion
 - **Corrige le principal défaut de UDP (l'agressivité)**

1. Généralités

Concept de *Port* TCP



1. Généralités

Concept de *Port* TCP

- Port : « guichet » d'accès aux services applicatifs
- Quadruplet < #Port source, #Port destination, @IP source, @IP destination >
→ identification sans ambiguïté des flux
- (#Port, @IP) = Socket
- (Socket_source, Socket_Destination) = connexion TCP

- Numéros de port codés sur 16 bits → 64 k ports
- Ports réservés (*ftp, http,...*) : numéros inférieurs à 1024,
- Ports libres (attention à la sécurité sur ces ports)

1. Généralités

Concept de *Port* TCP

- Pourquoi utiliser des ports ?
 - Classer les applications par catégorie
 - Allocation et gestion de ressources (CPU, mémoire, nombre de threads...)
 - Sécurité :
 - Contrôler ou interdire les accès sur certains ports
 - Reconnaître les flux (*http, ftp...*) et détecter les intrusions

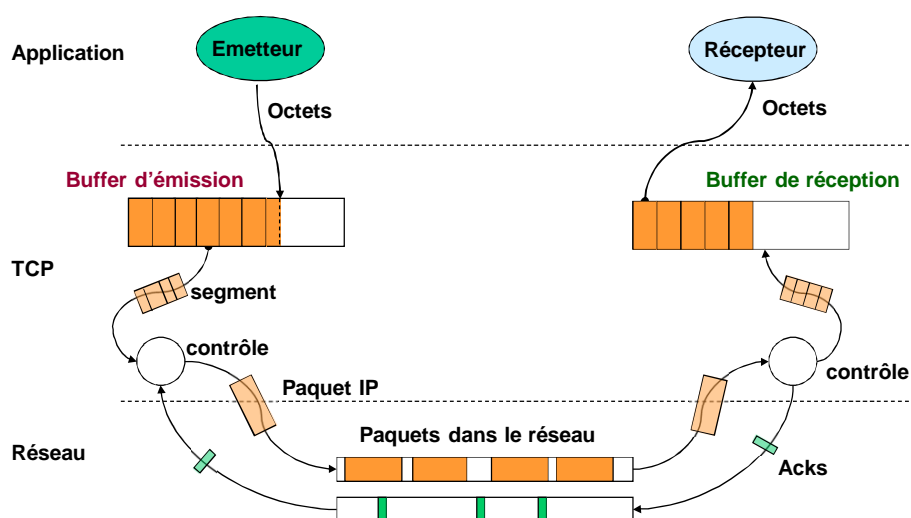
1. Généralités

Principe général de traitement des messages par TCP

- TCP découpe en segments les données applicatives à envoyer.
- A l'émission d'un segment, TCP-émetteur enclenche un temporisateur et attend que l'autre extrémité ait acquitté la réception du segment. Si l'acquiescement n'est pas reçu avant épuisement du temporisateur, le segment est retransmis.
- A la réception d'un segment, TCP-récepteur envoie ou non un acquiescement (on acquiesce au moins toutes les x unités de temps)
- TCP-récepteur remet dans l'ordre les données reçues (si nécessaire) avant de les passer à l'application
- TCP-récepteur rejette les segments dupliqués
- TCP réalise du contrôle de flux et du contrôle de congestion

1. Généralités

Buffers de TCP



1. Généralités

Flot vs Message

● Mode flot (*bit stream*)

- Mode de TCP
- Il n'y a pas de frontière entre les bits générés par l'application source.
- La source dépose en 'continu' ou non des flots de bits dans le buffer TCP
- TCP extrait un certain nombre de bits consécutifs pour former un segment et l'envoie

● Mode message

- Mode de UDP
- A chaque message de l'application correspond un segment UDP.
- La source dépose ses messages un par un dans le buffer UDP
- UDP extrait les segment un par un et les envoie.

1. Généralités

Interface TCP (interface sockets)

- **OPEN** (local port, socket distante, mode **actif/passif**,) → nom local de connexion
- **SEND** (nom local de connexion, adresse de buffer, nombre d'octets, indicateur d'urgence, timeout...)
- **RECEIVE** (nom local de connexion, adresse de buffer, nombre d'octets) → nombre d'octets, indicateur d'urgence
- **CLOSE** (nom local de connexion)
- **STATUS** (nom local de connexion) → Infos d'état
- **ABORT** (nom local de connexion)

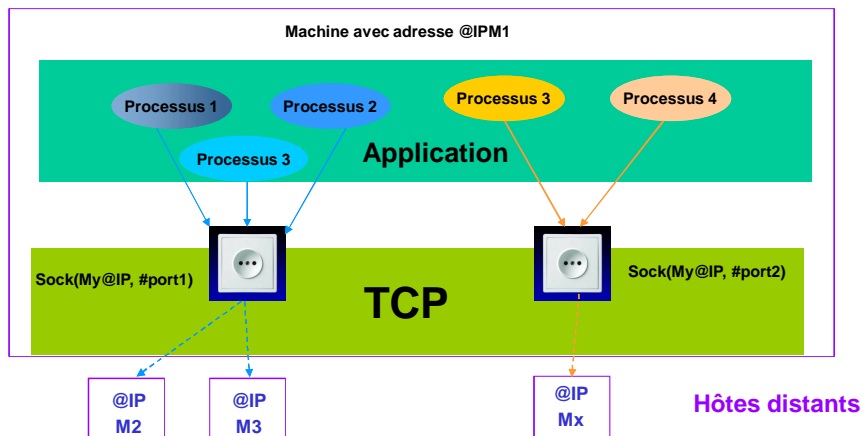
Toute la complexité de TCP est cachée aux applications. Elle apparaît uniquement quand on s'intéresse aux performances.

1. Généralités

Multiplexage selon TCP

● **Objectif :**

Permettre à plusieurs processus d'applications de partager une socket.



1. Généralités

Exemple de multiplexage avec TCP

● **Sans multiplexage de connexion**

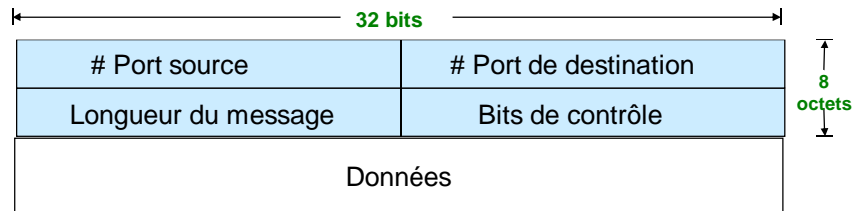
1. Lancer le web browser
2. Taper l'URL d'une page
3. Le browser ouvre une connexion TCP pour télécharger la page
4. Le browser ouvre une 2ème connexion pour ramener une image de la page
5. Le browser ouvre une 3ème connexion pour gérer le son associé

● **Avec multiplexage de connexion**

1. Lancer le web browser
2. Taper l'URL d'une page
3. Le browser ouvre une connexion TCP pour télécharger la page
4. Le browser utilise la même connexion pour la page, l'image et le son

2. Format de segment TCP

Segment UDP



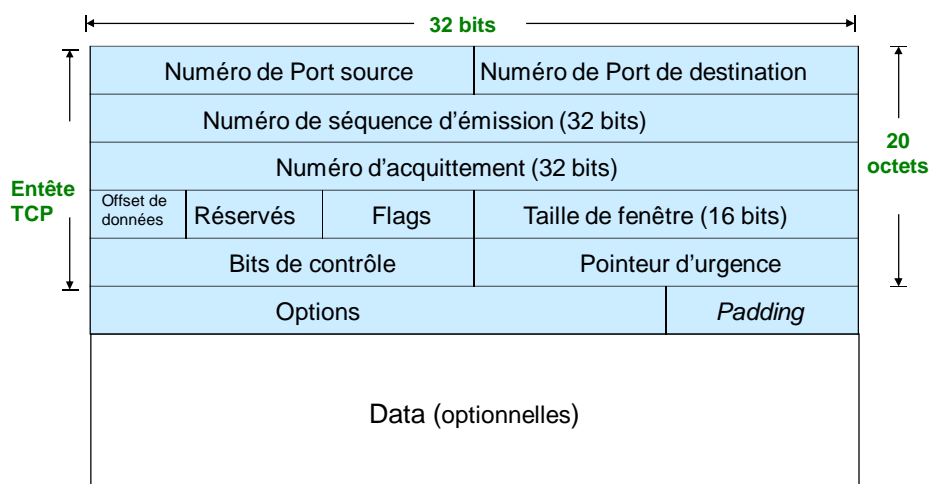
$$RUdp = \frac{TIU}{EtM + EIP + EUDP + TIU + EqM} = \frac{TIU}{EtM + 20 + 8 + TIU + EqM}$$

RUdp : rendement de UDP TIU: Taille information utile EUDP: Entête UDP EIP: Entête IP
EtM : Entête MAC EqM : Enqueue MAC

$$RUdp (Ethernet) = \frac{TIU}{Preamble + EtM + 20 + 8 + TIU + EqM} = \frac{TIU}{8 + 14 + 20 + 8 + TIU + 4} = \frac{TIU}{54 + TIU}$$

2. Format de segment TCP

Segment TCP



2. Format de segment TCP

Segment TCP

- **Port source** et **Port destination** : indiquent les ports utilisés par les applications.
- **Numéro de séquence d'émission** : **numéro du premier octet** de données dans le segment (sauf pour un segment avec SYN=1). Si le bit SYN est à 1 (c.-à-d. si le segment est une demande de connexion), le numéro de séquence signale au destinataire que le prochain segment de données qui sera émis commencera à partir de l'octet *Numéro de séquence d'émission* + 1.
- **Numéro d'acquittement** : indique le **numéro du prochain** octet attendu par le destinataire. Si dans le segment reçu FIN=1 et #Seq= x, alors le #Ack renvoyé est x+1 (x est interprété comme étant le numéro de l'octet FIN et que cet octet est acquitté).
- **Offset de données** : des options (de taille variable) peuvent être intégrées à l'entête et des données de bourrage peuvent être rajoutées pour rendre la longueur de l'entête multiple de 4 octets. L'Offset indique la position relative où commencent les données.
- **Taille de fenêtre** : indique le nombre d'octets que le destinataire peut recevoir. **si Fenêtre = F et que le segment contient un Numéro d'acquittement = A, alors le récepteur accepte de recevoir les octets numérotés de A à A + F -1.**
- **Bits de contrôle** : séquence de contrôle (CRC) portant sur l'entête de segment.

2. Format de segment TCP

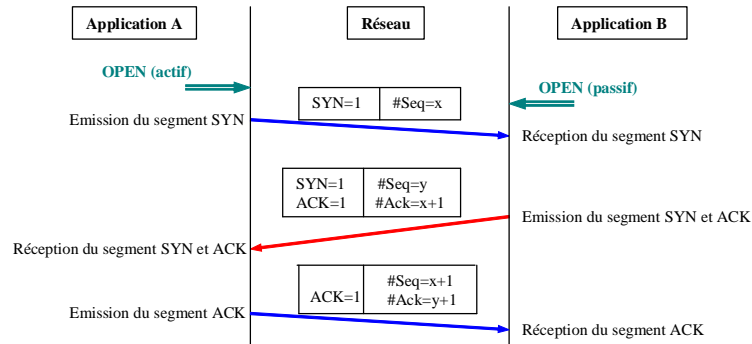
Segment TCP

- **Flags (URG, ACK, PSH, RST, SYN, FIN)**
 - **URG** = 1 si le segment contient des données urgentes et = 0 sinon.
 - **ACK** = 1 indique que le numéro d'acquittement est valide et il peut être pris en compte par le récepteur. ACK = 0 si l'accusé de réception est non valide.
 - **PSH** = 1 indique que les données doivent être remises à l'application dès leur arrivée et de ne pas les stocker dans une file d'attente.
 - **RST** = 1 pour demander la réinitialisation d'une connexion.
 - **SYN** = 1 et **ACK** = 0 servent à demander l'établissement de connexion.
 - **SYN** = 1 et **ACK** = 1 servent à accepter une demande de connexion.
 - **FIN** = 1 indique que l'émetteur n'a plus de données à émettre et demande de rompre la connexion **de son côté**.
- **Pointeur d'urgence** : indique l'emplacement (numéro d'octet) des données urgentes dans un segment. Il est valable uniquement si le bit URG=1.
- **Options** (taille variable) : options nécessitant des traitements particuliers.

3. Ouverture et fermeture de connexion TCP

Ouverture de connexion TCP

- Poignée de mains à 3 phases

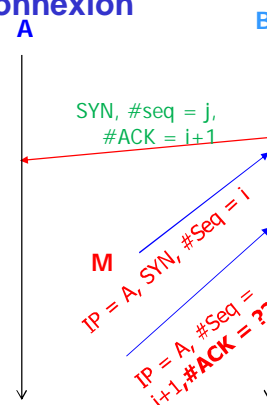


- Robustesse aux pertes de paquets et paquets tardifs durant la phase d'établissement de connexion.

3. Ouverture et fermeture de connexion TCP

Pourquoi choisir des numéros de séquences aléatoires à l'établissement de connexion

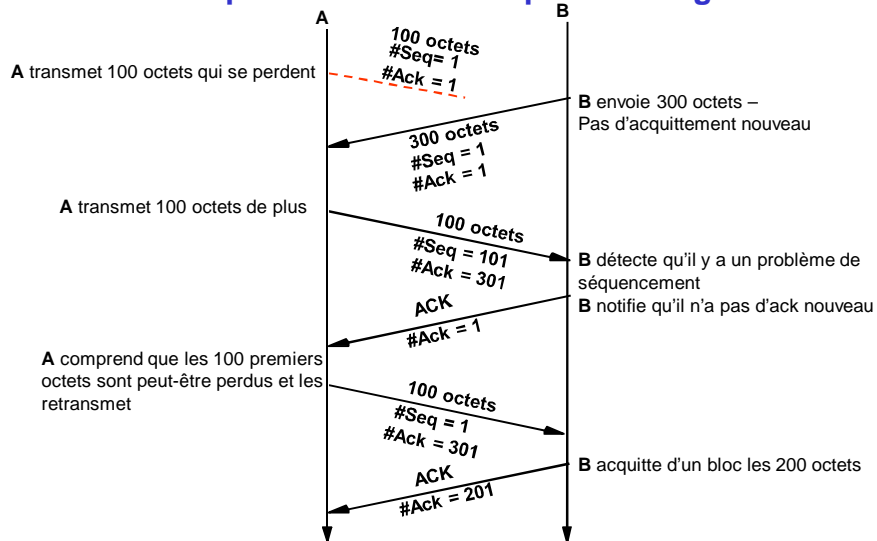
- Supposons que B a confiance en A à partir de l'@IP de A
 - e.g., accepter les demandes de création de compte issues de A
- Un attaquant M veut se faire passer pour A
 - M ne souhaite pas recevoir des données issues de B mais seulement lui envoyer des données pour créer un compte sur B
- M peut-il établir une connexion avec B en se faisant passer pour A?



Si M ne peut pas intercepter les messages entre A et B (pour connaître les numéros de séquence), il ne pourra pas se faire passer pour A une fois la connexion entre A et B établie, à condition que A et B choisissent des numéros de seq aléatoires.

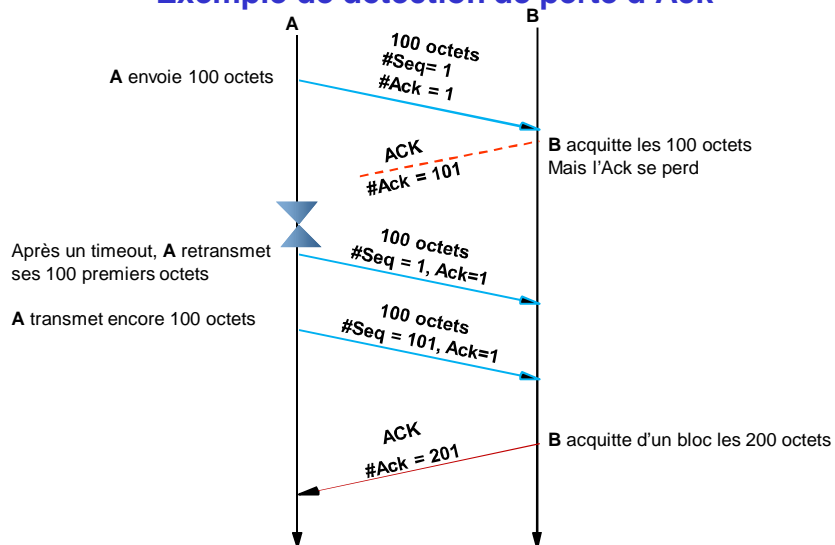
4. Contrôle d'erreurs

Exemple de détection de perte de segment



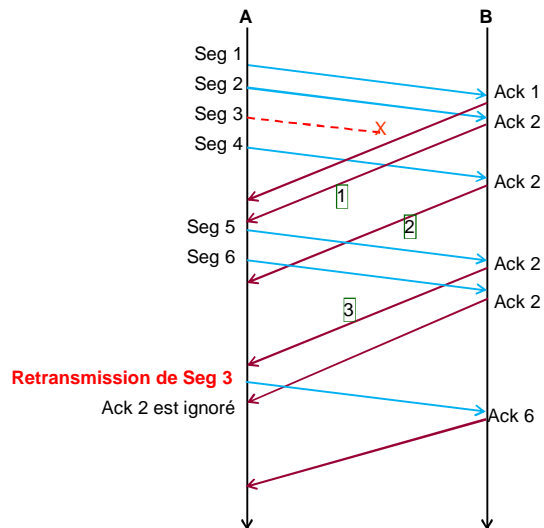
4. Contrôle d'erreurs

Exemple de détection de perte d'Ack



4. Contrôle d'erreurs

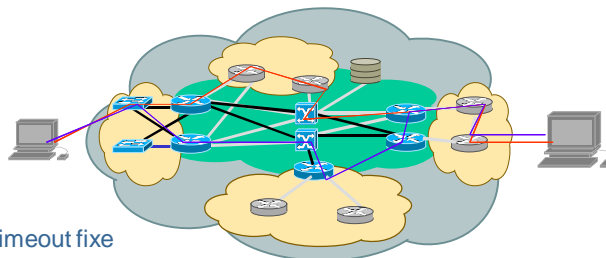
Exemple de retransmission rapide



4. Contrôle d'erreurs

Valeur d'armement du temporisateur

- Le temps de traversée du réseau est variable d'un segment à l'autre



- Valeur de timeout fixe
 - Valeur trop basse : détection de fausses pertes et retransmissions inutiles
 - Valeur trop élevée : beaucoup d'attente avant de décider de retransmettre en cas de perte effective
- TCP utilise un algorithme adaptatif de calcul de la valeur du time-out (être à l'écoute du réseau et fixer la valeur du timer en conséquence)
- Problème : compromis entre la précision de l'état du réseau et le coût de l'algorithme

4. Contrôle d'erreurs

Algorithme de Jacobson

- On part avec une valeur de *RTTestimé* (RTT: round trip time) par défaut
- A chaque réception d'Ack, on détermine le temps qu'a mis l'Ack pour être reçu, *NouveauRTT*
- A chaque réception de nouveau Ack, ré-estimer la valeur du RTT par :

$$RTTestimé = (\alpha * RTTestimé) + (1 - \alpha) NouveauRTT$$

- La valeur d'armement du timer, dite RTO (retransmission TimeOut) est calculée par la formule suivante qui évite des valeurs trop élevées ou trop basses suite aux estimations :

$$RTO = \text{Min} \{ \text{BorneSupRTO}, \text{Max} \{ \text{BorneInfRTO}, \beta * RTTestimé \} \}$$

- *BorneInfRTO*, *BorneSupRTO*, α et β sont fixées par configuration. $\alpha < 1$ et $\beta > 1$.
- Beaucoup d'implantations de TCP utilisent $\alpha = 0,875$ et $\beta = 2$

4. Contrôle d'erreurs

Problème avec l'algorithme de Jacobson

- Idéal pour le calcul du RTO : on reçoit un Ack pour CHAQUE segment
- Inconvénient : on ne peut plus bénéficier des Ack groupés
- TCP réel
 - un Ack peut être associé à 1 segment, plusieurs segments, un morceau de segment ou à des morceaux contigus de segments.
 - Cas de retransmission :
 - ◆ Quand TCP retransmet un segment et puis il reçoit un acquittement : est-ce que cet Ack correspond au segment initial ou bien au segment retransmis ? TCP n'a aucun moyen de distinguer les deux cas.
 - ◆ Associer l'Ack au segment le plus ancien peut conduire à une surestimation du *NouveauRTT* si plusieurs tentatives de retransmissions ont été effectuées.
 - ◆ Associer l'Ack au dernier segment émis peut conduire à une sous-estimation du *NouveauRTT*.

4. Contrôle d'erreurs

Algorithme de Karn

- Une des solutions utilisées dans la pratique pour remédier au problème avec l'algorithme de Jacobson est connue sous le nom d'**algorithme de Karn** :
 - Ne pas mettre à jour la valeur de *RTTestimé* en cas de retransmission
 - A chaque de retransmission : calcul d'une valeur dite *RTOaugmenté* par la formule suivante (λ vaut généralement 2 et RTO est la dernière valeur de RTO fournie par l'algorithme de Jacobson) :
$$RTO_{augmenté} = RTO * \lambda$$
 - Armer le timer avec la valeur *RTOaugmenté*.
 - Dans la pratique, après n retransmissions consécutives, *RTOaugmenté* vaut $RTO * 2^n$ (augmentation exponentielle)

5. Contrôle de congestion

Principe de base

- Les applications doivent s'adapter aux conditions du réseau et non le contraire
 - Quand le réseau est sous-chargé, TCP émetteur augmente le débit
 - Quand le réseau est jugé surchargé, TCP émetteur réduit le débit
- Problème avec les retransmissions :
 - Un utilisateur non averti (ou malicieux) pourrait se dire « je transmets avec un débit élevé et en cas de perte, je retransmets ».
 - Sans précautions, les retransmissions aggravent la congestion. En effet, la congestion de certains routeurs conduit à la perte de segments (car ils sont rejetés par des routeurs saturés). Ensuite, les nœuds d'extrémité qui ont perdu leurs segments retransmettent ce qui augmente la charge du réseau et donc à plus de pertes et ainsi de suite jusqu'à ce que le réseau se bloque complètement.
- L'application émettrice doit aussi tenir compte de l'application réceptrice (crédit)
- Conséquence : le débit de l'émetteur dépend de la charge du réseau et du crédit que lui accorde le récepteur

5. Contrôle de congestion

Principes de base

Structures de données utilisées par TCP

- Fenêtres glissantes (stocker les segments avant leur transmission, avant de les passer à l'application et pour contrôler les Ack)
- Fenêtre de crédit (pour ne pas saturer le récepteur)
- Fenêtre de congestion de l'émetteur (notée Cwd : congestion window) : c'est la taille maximum (en octets) que TCP-émetteur peut émettre avant d'être obligé d'attendre un Ack. Cwd n'a pas de valeur fixe (comme pour la couche liaison de données) mais une valeur dynamique.

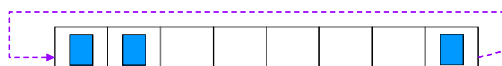
Hypothèse de TCP : si la source ne reçoit pas d'Ack, la cause la plus probable est une perte de segment due à une congestion d'un des routeurs traversés

Hypothèse non vraie dans le cas des réseaux sans fils

5. Contrôle de congestion

Rappel sur les buffers circulaires

	0	1	2	3	4	5	6	7	Producteur	Consommateur
Etat initial									CP = 8 IxP = 0	CC = 0 IxC = 0
4 produits	■	■	■	■					NP = 4 IxP = 4	NC = 4 IxC = 0
2 consommés			■	■					NP = 6 IxP = 4	NC = 2 IxC = 2
5 produits	■		■	■	■	■	■	■	NP = 1 IxP = 1	NC = 7 IxC = 2
1 produit 2 consommés	■	■			■	■	■	■	NP = 2 IxP = 2	NC = 6 IxC = 4

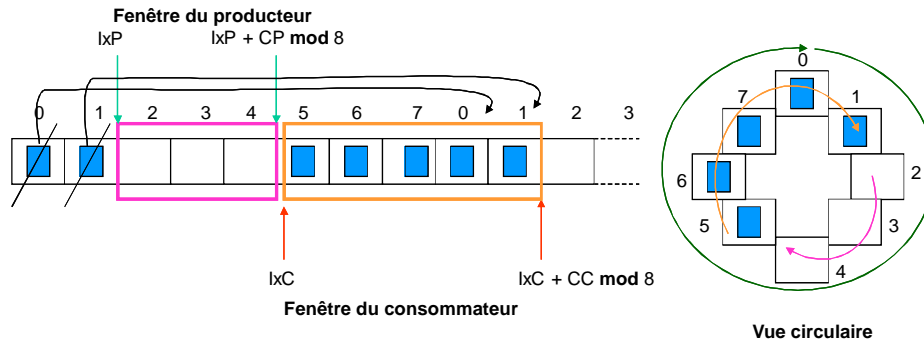


Gestion circulaire

CP : capacité de production CC : capacité de consommation IxP : indice de la case de la case livre
IxC : indice de la case de début de consommation

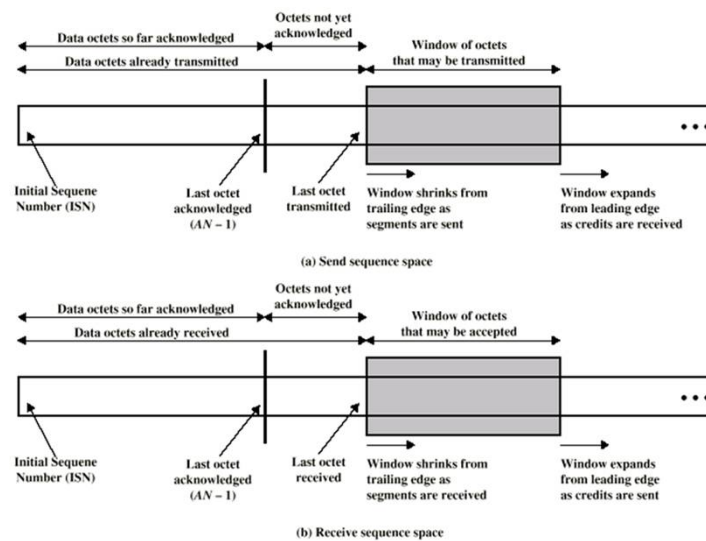
5. Contrôle de congestion

Rappel sur les buffers circulaires



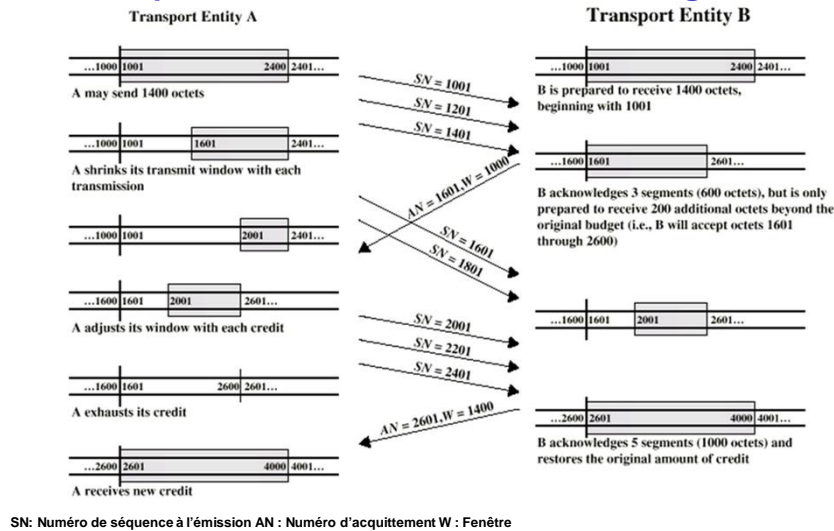
5. Contrôle de congestion

Fenêtres glissantes



5. Contrôle de congestion

Exemple de fonctionnement de fenêtres glissantes



5. Contrôle de congestion

Stratégie « Additive increase – Multiplicative decrease »

« Démarrage lent – Diminution dichotomique » ou « Augmentation lente – Réduction drastique »

- Le nombre d'octets que l'émetteur peut transmettre à un instant donné est limité à une quantité dite *Fenêtre Autorisée* :

$\text{Fenêtre autorisée} = \text{Min} \{ \text{Fenêtre de contrôle de flux}, \text{Fenêtre de congestion} \}$

$\text{Fenêtre de contrôle de flux} = \text{Crédit} - \text{nombre d'octets non encore acquittés}$

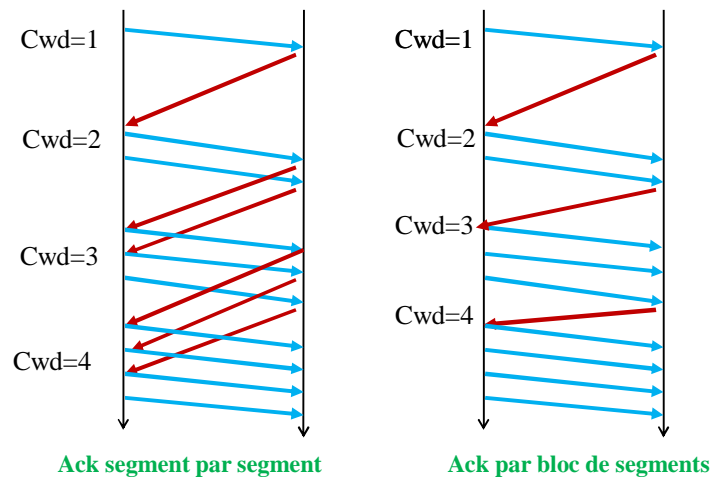
- Démarrer la transmission avec une valeur de Cwd égale à Cwd_{min} (qui est un paramètre de configuration de réseau et qui correspond à un segment)
- Si l'Ack revient avant la fin du timer, augmenter la fenêtre de congestion
- Si l'Ack ne revient pas, réduire de moitié la fenêtre de congestion

$$Cwd = Cwd + \text{TailleSegment}$$

$$Cwd = Cwd/2$$

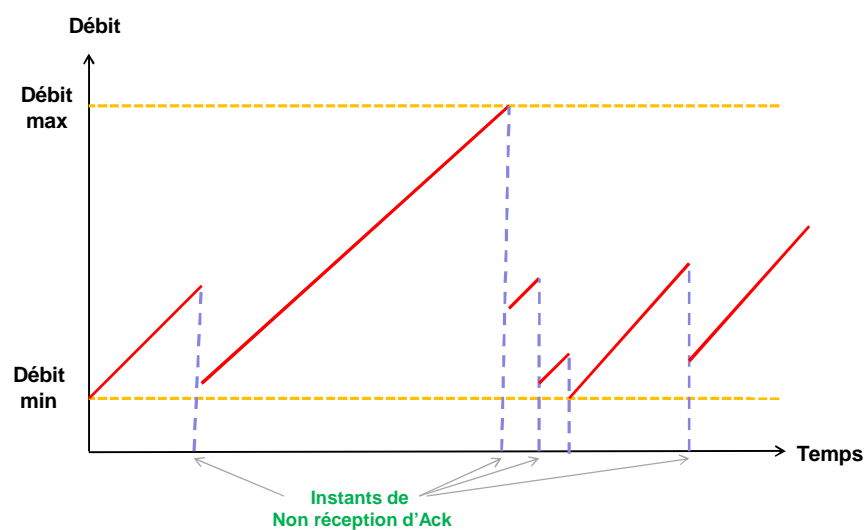
5. Contrôle de congestion

Exemple de « Additive increase »



5. Contrôle de congestion

Exemple de changement de débit



6. Autres sur TCP

Syndrome de la fenêtre stupide

- Situation qui arrive lorsque les données sont passées à TCP source :
 - Soit en très petite quantité (ex. octet par octet) – Dans ce cas, l'application Source est très lente. TCP émetteur envoie des segments de données contenant très peu de données (ex. 1 octet de données par segment).
 - Soit en gros blocs, mais l'application côté récepteur lit les données octet par octet (ou en petite quantité) – Dans ce cas, l'application Destinataire est lente. TCP récepteur envoie un Ack pour chaque octet délivré à l'application Destinataire.
- Dans les deux cas : gaspillage de bande passante
- Solution de Nagle: Retarder la transmission des segments de données pour envoyer des segments avec une taille minimale
- Solution de Clark : Retarder la transmission des Acks et acquitter plusieurs octets en même temps

Protocole TCP – M1 Info / 2020-2021 – Z. Mammeri - UPS

6. Autres sur TCP

TCP avancés

- TCP : Reno, New Reno, Vegas, Tahoe...
 - Algorithmes plus élaborés pour l'estimation du RTT et calcul du RTO
 - Algorithmes plus élaborés pour le contrôle de congestion
- TCP pour les réseaux sans fils
- TCP « light » (pour les réseaux de capteurs...)

Protocole TCP – M1 Info / 2020-2021 – Z. Mammeri - UPS

44