

- UTC505/USRS4D -

Transport d'information

E. Gressier-Soudan

26/09/2021

E. Gressier-Soudan

1



Transport Internet

- Protocole de Transport TCP, Transmission Control Protocol

Relations Applicatifs/couche Application/couche Transport

- UDP :
 - Client/Serveur en LAN
 - Multimedia en LAN /WAN dont jeux multijoueurs en ligne
 - Multicast
 - TFTP, DNS, RTP, NFS, OSPF, RIP, SNMP, VoIP ...
- TCP :
 - Transfert de données fiable (fichiers, terminal virtuel ...
 - Client/Serveur en WAN
 - Unicast
 - DNS, Telnet, FTP, HTTP, SMTP, NNTP, NFS, BGP, LDAP ...
- La QoS du point de vue de la couche Transport, c'est un bon réglage des paramètres des connexions, et de l'implantation de la pile de protocole.

1. Transmission Control Protocol - TCP

- RFC 793: Transmission Control Protocol
- RFC 1180: A TCP/IP Tutorial
- RFC 2581: TCP Congestion Control
- RFC 2525: Known TCP Implementation Problems
- RFC 7323: TCP Extensions for High Performance
- ...
- Voir <https://tools.ietf.org/html/rfc7805> pour une cartographie des RFC relatives à TCP, date d'avril 2016, y figure la dernière RFC concernant une extension du protocole TCP, elle porte le numéro 7413...

Caractéristiques Générales de TCP

- Orienté Flot d'octets
 - ne préserve pas la notion d'enregistrement (du point de vue de l'utilisateur au niveau API socket),
 - séquençement des octets garanti.
- Mécanisme de connexion, en Full-Duplex (bidirectionnel)
 - Contrôle d'erreurs : Acquits Positifs avec Retransmissions en cas d'erreurs, avec le flot d'octets les ACK sont cumulatifs ("cumulative ACKs"), porté par les segments en sens inverse (ACK superposés, "piggybacking")
 - Contrôle de flux et fenêtre glissante
 - Pas de duplications des données possibles à la livraison,
 - Données urgentes,
 - Informé des ruptures de connexions (le moment dépend de la configuration choisie, c'est au moment où on envoie).
 - D'autres fonctionnalités plus fines qu'on découvrira dans le cours

Diamètre d'une connexion TCP en lien avec MSS

- Connaître le diamètre permet d'éviter la fragmentation en cours de route.
- Le diamètre de la connexion correspond au plus petit MTU rencontré sur le chemin qui supporte une connexion dans Internet.
- Quand une connexion TCP est ouverte, TCP utilise le paramètre MSS fourni par l'autre entité ou le MTU de l'interface de sortie.
- Les datagrammes sur cette connexion ont le bit DF à 1 (Don't fragment). Un routeur qui doit fragmenter, élimine le datagramme et génère un message ICMP "can't fragment". Suivant la version d'ICMP, la taille du MTU avec le prochain noeud peut être indiquée.
- Les prochains datagrammes envoyés sont plus petits. Toutefois, comme les routes sont multiples, et qu'elles changent, TCP tente périodiquement (recommandé toutes les 10 mn) d'augmenter la taille des segments.
- *En plomberie, on parlerait de la section d'une canalisation*

Entête d'un Segment TCP

TCP Header																																	
Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0		N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size																
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if <i>data offset</i> > 5. Padded at the end with "0" bytes if necessary.)																															
...																															

L'entête a toujours une longueur qui est multiple de 4 octets

https://en.wikipedia.org/wiki/Transmission_Control_Protocol

On peut remarquer que les @IP source et destination à la fin de l'entête IP précède les #port source et destination au début de l'entête TCP. Ce n'est pas un hasard car ils servent à identifier les flux, les connexions entre client et serveur....

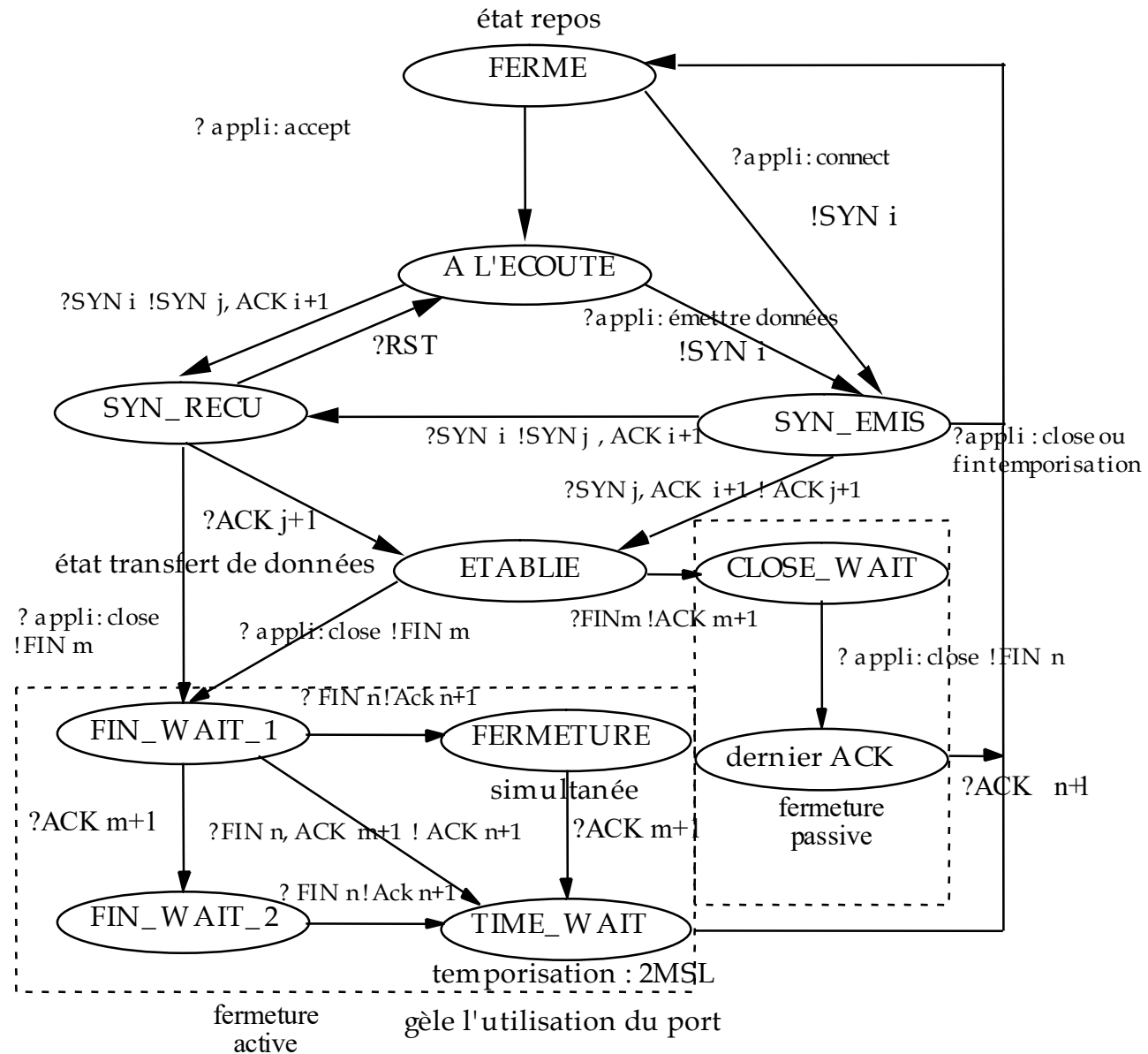
Champs significatifs d'un entête de segment TCP

- Les bits indicateurs, s'ils sont positionnés, informent sur la nature du segment :
 - **"SYN"** initialisation d'une connexion, dans ce cas le numéro de séquence porté indique le numéro du premier octet du flot de données, un segment contenant un SYN consomme un octet dans le flot d'octets de données, le numéro i du premier numéro de séquence est déterminé aléatoirement
 - **"ACK"** acquittement des octets -> numéro envoyé – 1, acquit positif
 - **"RST"** réinitialisation de connexion
 - **"URG"** données urgentes contenues dans le segment
 - **"PSH"** délivrer les données au plus tôt au récepteur dès qu'elles sont correctement reçues
 - **"FIN"** plus aucune donnée ne sera envoyée par celui qui a fait FIN.
- L'entête seule fait $5 \times 4 = 20$ octets, Si le champ Data Offset est > 5 , il y a des options entre l'entête fixe et les données :

On a vu la notion d'Extrémité de connexion : @IP + n°port,

Quand on ajoute le numéro du premier octet associé à chaque extrémité d'une connexion, on définit une "instance de connexion". L'instance de connexion ne dure que jusqu'à la fermeture de la connexion.

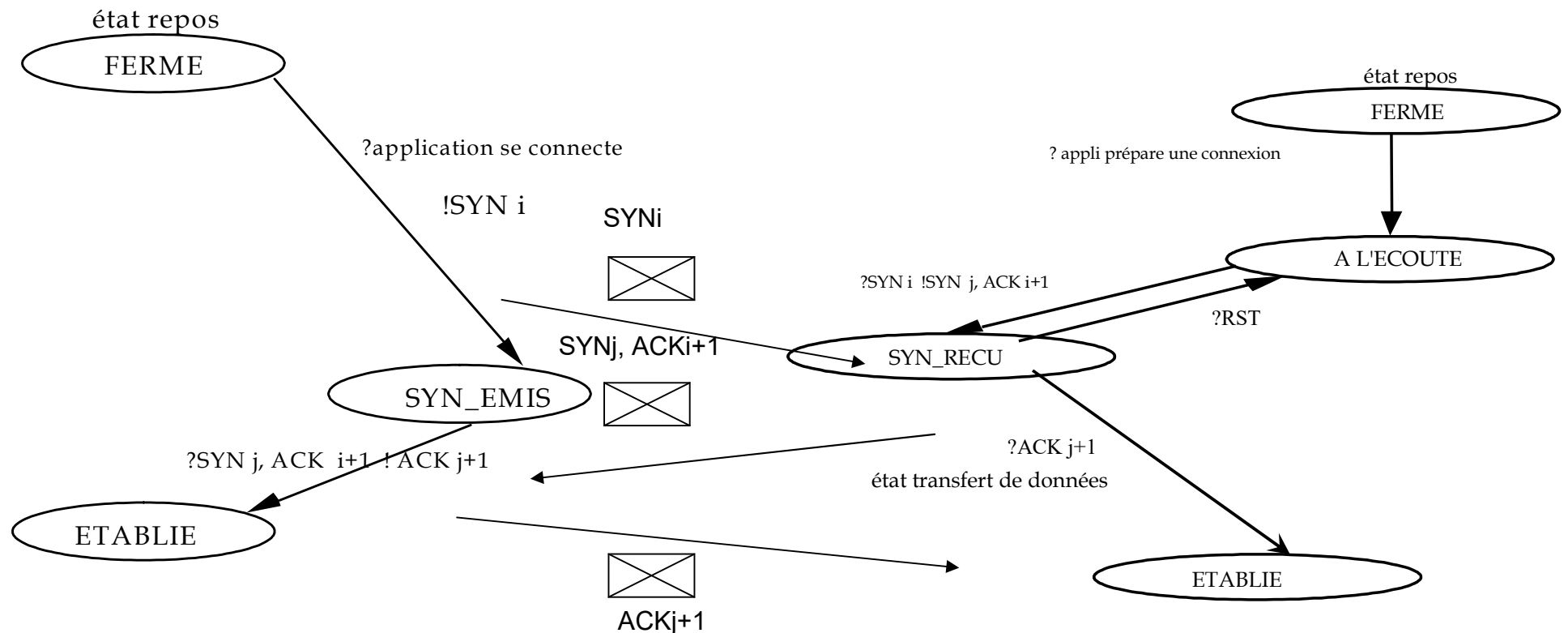
Automate Protocolaire TCP



26/09/2021



Ouverture de connexion Tcp



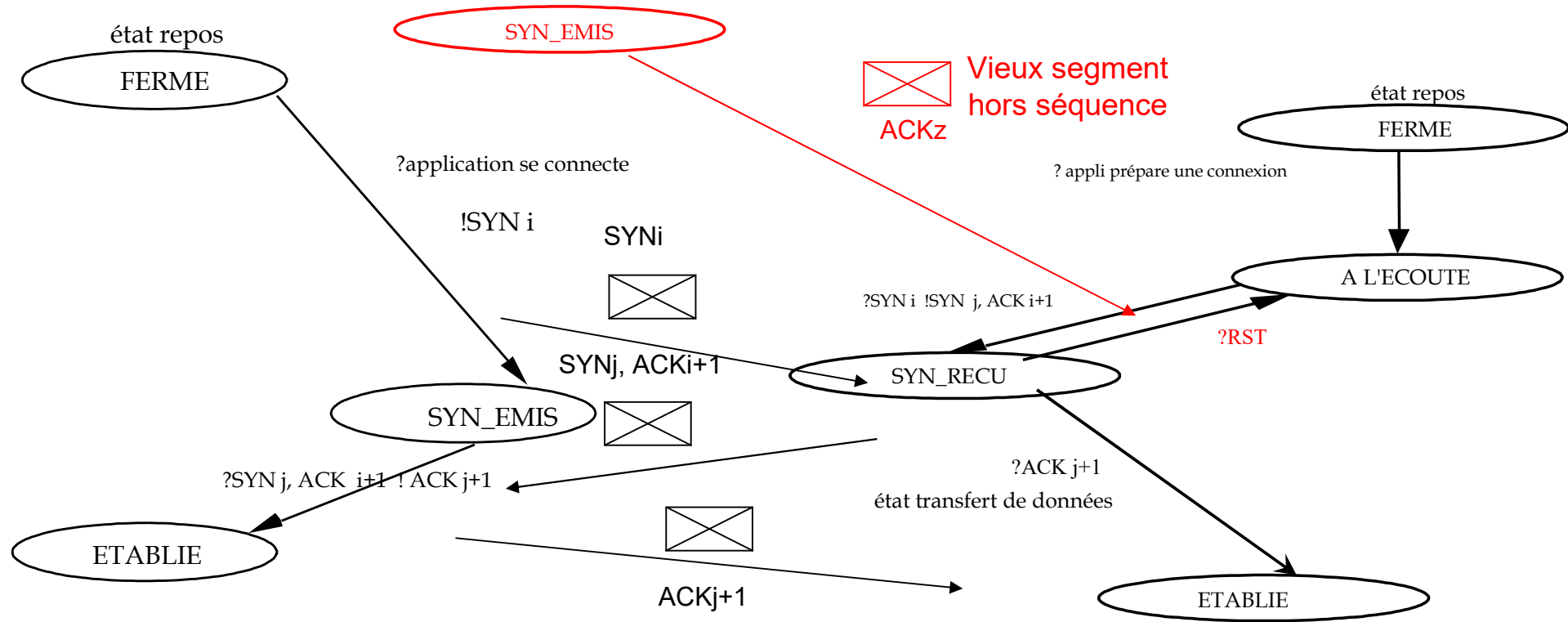
TCP numérote les octets :

- "i" est l'ISN, Initial Sequence Number, il est généré le plus aléatoirement possible
- "i+1" est le numéro du prochain octet attendu par le récepteur
- SYN porte un "phantom byte"

Gestion de l'échec d'une tentative de connexion

- Quand l'ouverture de connexion échoue (pas de réponse avant fin de temporisation), d'autres tentatives sont effectuées
- Un principe d'**exponential backoff** est mis en œuvre pour espacer les tentatives d'ouverture, le délai pour recommencer une nouvelle tentative double après chaque échec... le livre Fall et al donne un délai initial de 3s.
- Le nombre de tentatives est configurable au niveau système, 5 en général
- Ça marche pour le SYN et pour le SYN-ACK
- Sur Linux, c'est la variable `net.ipv4.tcp_syn_retries` qui représente ce paramètre

Ouverture de connexion perturbée



Questions :

- Est-ce que l'ouverture de connexion en cours va se faire quand même ? Que se passe-t-il ?
- Si c'est le client qui reçoit un segment non conforme au déroulement du protocole que se passe-t-il p/r à l'automate protocolaire ?

Ouverture de connexion TCP

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	Inventec_9a:4d:aa	Broadcast	ARP	42	who has 163.173.228.17? Tell 163.173.231.107
2	0.00060600	dellEsge_f9:ad:5a	Inventec_9a:4d:aa	ARP	60	163.173.228.17 is at 00:0f:1f:f9:ad:5a
3	0.00062700	163.173.231.107	163.173.228.17	TCP	66	50074 > http [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
4	0.00186700	163.173.228.17	163.173.231.107	TCP	66	http > 50074 [SYN, ACK] Seq=0 Ack=1 win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=4
5	0.00191200	163.173.231.107	163.173.228.17	TCP	54	50074 > http [ACK] Seq=1 Ack=1 win=65700 Len=0
6	0.00241400	163.173.231.107	163.173.228.17	HTTP	544	GET / HTTP/1.1
7	0.00313900	163.173.228.17	163.173.231.107	TCP	60	http > 50074 [ACK] Seq=1 Ack=491 win=6912 Len=0
8	0.00360700	163.173.228.17	163.173.231.107	TCP	353	[TCP segment of a reassembled PDU]
9	0.00388500	163.173.228.17	163.173.231.107	TCP	1514	[TCP segment of a reassembled PDU]
10	0.00416400	163.173.231.107	163.173.228.17	TCP	54	50074 > http [ACK] Seq=491 Ack=1760 win=65700 Len=0
11	0.00500200	163.173.228.17	163.173.231.107	TCP	1514	[TCP segment of a reassembled PDU]
12	0.00503000	163.173.228.17	163.173.231.107	HTTP	74	HTTP/1.1 200 OK (text/html)
13	0.00528500	163.173.231.107	163.173.228.17	TCP	54	50074 > http [ACK] Seq=491 Ack=3240 win=65700 Len=0
14	0.03563900	163.173.231.107	163.173.228.17	HTTP	559	GET /logo_cnam.gif HTTP/1.1
15	0.03661300	163.173.228.17	163.173.231.107	TCP	353	[TCP segment of a reassembled PDU]
16	0.03679700	163.173.228.17	163.173.231.107	TCP	1514	[TCP segment of a reassembled PDU]
17	0.03691400	163.173.231.107	163.173.228.17	TCP	54	50074 > http [ACK] Seq=996 Ack=4999 win=65700 Len=0
18	0.03703200	163.173.228.17	163.173.231.107	TCP	1514	[TCP segment of a reassembled PDU]
19	0.03787100	163.173.228.17	163.173.231.107	TCP	1514	[TCP segment of a reassembled PDU]
20	0.03791500	163.173.231.107	163.173.228.17	TCP	54	50074 > http [ACK] Seq=996 Ack=7919 win=65700 Len=0
21	0.03792400	163.173.228.17	163.173.231.107	TCP	1514	[TCP segment of a reassembled PDU]
22	0.03797300	163.173.228.17	163.173.231.107	HTTP	1275	HTTP/1.1 200 OK (GIF87a)
23	0.03800800	163.173.231.107	163.173.228.17	TCP	54	50074 > http [ACK] Seq=996 Ack=10600 win=65700 Len=0
24	15.0372060	163.173.228.17	163.173.231.107	TCP	60	http > 50074 [FIN, ACK] Seq=10600 Ack=996 win=7984 Len=0
25	15.0372830	163.173.231.107	163.173.228.17	TCP	54	50074 > http [ACK] Seq=996 Ack=10601 win=65700 Len=0
26	15.0375270	163.173.231.107	163.173.228.17	TCP	54	50074 > http [FIN, ACK] Seq=996 Ack=10601 win=65700 Len=0
27	15.0379470	163.173.228.17	163.173.231.107	TCP	60	http > 50074 [ACK] Seq=10601 Ack=997 win=7984 Len=0

Trace Wireshark

26/09/2021

E. Gressier-Soudan

13



Numérotation relative

	Time	163.173.231.107	163.173.228.17	Comment	
<code>connect()</code>	0.000627000	(50074) →	(80) SYN	Seq = 0	<code>socket(), bind(), listen(), accept() ont été faits avant</code>
	0.001867000	(50074) ←	(80) SYN, ACK	Seq = 0 Ack = 1	
	0.001912000	(50074) →	(80) ACK	Seq = 1 Ack = 1	
<code>send(), receive(), read(), write()</code>	0.002414000	(50074) →	(80) PSH, ACK - Len: 490	Seq = 1 Ack = 1	<code>send(), receive(), read(), write()</code>
	0.003139000	(50074) ←	(80) ACK	Seq = 1 Ack = 491	
	0.003607000	(50074) →	(80) PSH, ACK - Len: 299	Seq = 1 Ack = 491	
	0.003885000	(50074) ←	(80) ACK - Len: 1460	Seq = 300 Ack = 491	
	0.004164000	(50074) →	(80) ACK	Seq = 491 Ack = 1760	
	0.005002000	(50074) ←	(80) ACK - Len: 1460	Seq = 1760 Ack = 491	
	0.005030000	(50074) →	(80) PSH, ACK - Len: 20	Seq = 3220 Ack = 491	
	0.005285000	(50074) ←	(80) ACK	Seq = 491 Ack = 3240	
	0.035639000	(50074) →	(80) PSH, ACK - Len: 505	Seq = 491 Ack = 3240	
	0.036613000	(50074) →	(80) PSH, ACK - Len: 299	Seq = 3240 Ack = 996	
	0.036797000	(50074) ←	(80) ACK - Len: 1460	Seq = 3539 Ack = 996	
	0.036914000	(50074) →	(80) ACK	Seq = 996 Ack = 4999	
	0.037032000	(50074) ←	(80) ACK - Len: 1460	Seq = 4999 Ack = 996	
	0.037871000	(50074) ←	(80) ACK - Len: 1460	Seq = 6459 Ack = 996	
	0.037915000	(50074) →	(80) ACK	Seq = 996 Ack = 7919	
	0.037924000	(50074) ←	(80) ACK - Len: 1460	Seq = 7919 Ack = 996	
	0.037973000	(50074) →	(80) PSH, ACK - Len: 1221	Seq = 9379 Ack = 996	
	0.038008000	(50074) →	(80) ACK	Seq = 996 Ack = 10600	
	15.037206000	(50074) ←	(80) FIN, ACK	Seq = 10600 Ack = 996	
	15.037283000	(50074) →	(80) ACK	Seq = 996 Ack = 10601	
<code>close() ou shutdown()</code>	15.037527000	(50074) →	(80) FIN, ACK	Seq = 996 Ack = 10601	<code>close() ou shutdown()</code>
	15.037947000	(50074) ←	(80) ACK	Seq = 10601 Ack = 997	

26/09/2021

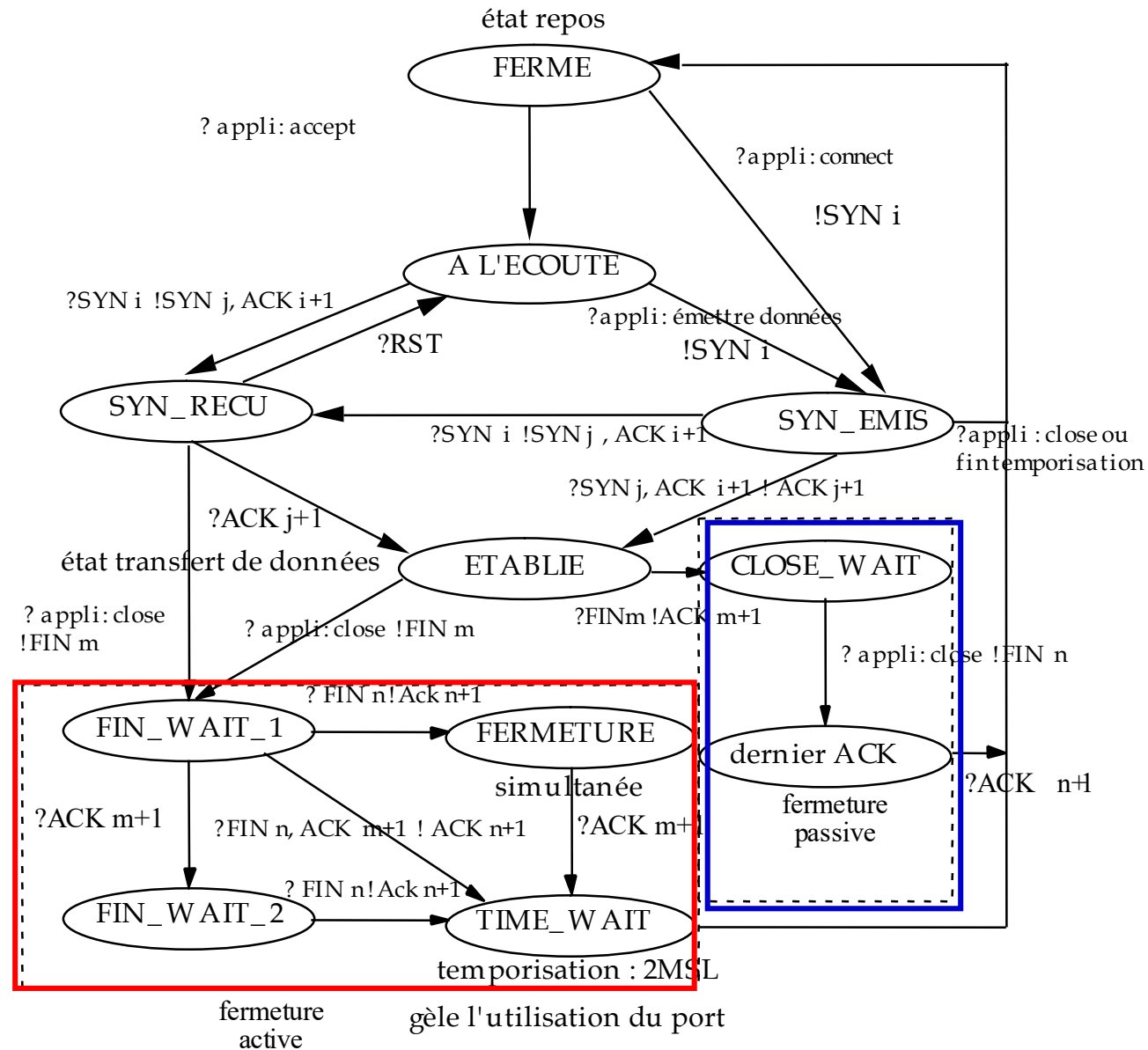
E. Gressier-Soudan

14

Ack

- Le récepteur n'est pas obligé d'envoyer un ACK pour chaque segment reçu. On a un système d'ACK retardés, mais on ne peut pas retarder plus de 500ms, dans les mises en œuvre c'est plutôt 200ms

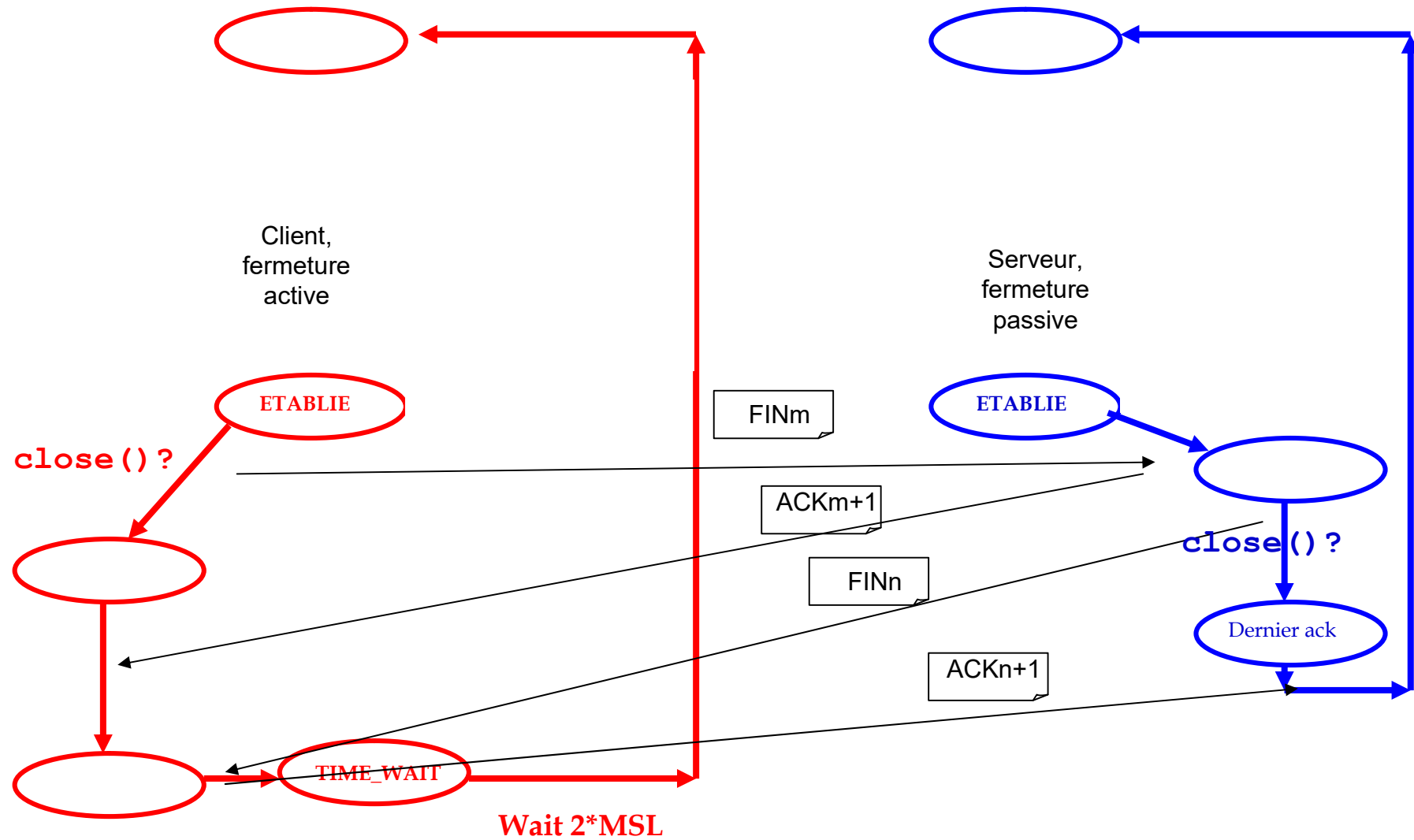
Automate Protocolaire TCP- partie fermeture de Connexion



26/09/2021



Fermeture de connexion



Espace de numérotation et vieux paquet

Vieux paquet : c'est un paquet d'une ancienne connexion qui a été retardé, peut-être égaré un certain temps et qui revient dans la connexion et qui peut être interprété comme un paquet valide. C'est un problème plus spécifique en Transport qu'en Liaison. Il peut revenir à l'ouverture de connexion ou pendant que la connexion est établie. Il correspond à une connexion identifiée mais à une instance ancienne.

Quelle valeur de MSL (Maximum Segment Lifetime) faut-il pour qu'un vieux paquet ne soit pas accepté comme un paquet correct du flot d'une connexion, 30s, 1mn, 2mn ?

Combien de temps pour épuiser l'espace des n° de séquence TCP (espace de numérotation des octets de 4 Go)

Le Syndrome du vieux paquet est aussi pris en compte à la fermeture de connexion par le **gel de référence** de la connexion, au niveau du client en général, mais pas avec HTTP.

Vieux Paquets et Epuisement de l'espace de n°

Type de Réseau	nominal b/s	nominal o/s	délai d'épuisement
Ethernet	10Mb/s	1,25Mo/s	53mn
FastEthernet	100Mb/s	12,5Mo/s	5mn20s
GigabitEthernet	1Gb/s	125Mo/s	32s effrayant...
10GigabitEthernet	10Gb/s	1250Mo/s	3,2s terrorisant...
<i>Multiplex T1 continental</i>	<i>1,544Mb/s</i>	<i>0,193Mo/s</i>	<i>46mn</i>
<i>Multiplex T3 continental</i>	<i>44,736Mb/s</i>	<i>5,592Mo/s</i>	<i>12mn</i>

C'est un peu moins pire en réalité... sauf dans les data centers avec des débits de 40Gb/s voire 100Gb/s, des tailles de fenêtre de près d'1Go et des optimisations de code et de recopies mémoire diminuent la latence du système dans la communication de bout en bout. On est presque au niveau NIC...

MSL

- MSL : Maximum Segment Lifetime (30s, 1mn, 2 mn), le client fait la fermeture active, et le serveur la fermeture passive, la connexion peut rester bloquée longtemps (max 4mn), c'est le client qui gèle l'instance de connexion car son port n'est pas réutilisable pendant 2MSL
- État TIME_WAIT de l'automate TCP -> pb lors de réutilisation du port si un client enchaîne fermeture/ouverture immédiate de connexion sur ce port (le gel de connexion n'opère pas). Option pour court-circuiter le gel de connexion : SO_REUSEADDR

TCP Option TIMESTAMP

- Une option relativement récente qui permet de détecter les vieux paquets, PAWS pour Protection About Wrapped Sequence Numbers)
- Initialement RFC1323, mise à jour dans la RFC7323 **"TCP Extensions for High Performance"**
- A l'origine pour mesurer le délai A/R, RTT (Round Trip Time) :
 - L'émetteur met une date logique plus que temporelle dans le champ TSval,
 - Le récepteur envoie dans un segment de retour cette date dans le champ TSecr
 - A l'arrivée du segment en retour, on peut mesurer le RTT sous certaines conditions
 - C'est aussi une date qui permet de situer un segment dans le temps et de savoir si il est ou non dans la fenêtre

Exemple issu de la RFC 7323

TCP entité 1

TCP entité 2

```

    <A,TSval=1,TSecr=120> ----->
<----- <ACK(A),TSval=127,TSecr=1>
    <B,TSval=5,TSecr=127> ----->
<----- <ACK(B),TSval=131,TSecr=5>

```

```
. . . . . pause (60 ticks)
```

```

<C, TSval=65, TSecr=131> ----->
<----- <ACK(C), TSval=191, TSecr=65>

```

Le segment C ne peut servir au calcul de RTT pour l'entité 1, l'écho TSecr de valeur 131 à TSval indique aucun échange.

Le segment ne peut être pris en compte que s'il fait avancer la frontière gauche de la fenêtre.

En fait, c'est plus complexe... (1)

Les segments arrivent en sequence, mais
l'acquittement est retardé (cumulative ACK)

```
TS.Recent
<A, TSval=1> -----> 1
<B, TSval=2> -----> 1
<C, TSval=3> -----> 1
      <----- <ACK(C), TSecr=1>
```

Naturellement c'est le segment (A) qui compte, le timestamp du plus vieux segment non acquitté est utilisé pour l'écho.

En fait, c'est plus complexe... (2)

Les segments arrivent dans le désordre et sont tous acquittés.

	TS.Recent
<A, TSval=1> ----->	1
<----- <ACK(A), TSecr=1>	1
<C, TSval=3> ----->	1
<----- <ACK(A), TSecr=1>	1
<B, TSval=2> ----->	2
<----- <ACK(C), TSecr=2>	2
<E, TSval=5> ----->	2
<----- <ACK(C), TSecr=2>	2
<D, TSval=4> ----->	4
<----- <ACK(E), TSecr=4>	4

Avoir en tête qu'on envoie celui qui fait avancer la limite gauche de la fenêtre.

Les questions à traiter avec TCP

- SACK et fenêtrage en réception (vu en ED)
- Timeout et Calcul du délai de retransmission
- Contrôle de Flux et Fenêtrage glissant en émission, option Window Scale
- Contrôle de Congestion
- Les évolutions de TCP

Temporisations : Fast Recovery and Fast Retransmit

TCP offre un transfert fiable au-dessus d'un réseau à datagrammes donc potentiellement non fiable et pouvant subir des erreurs de transmission et des congestions (plus fréquent). Deux techniques sont utilisées pour les retransmissions :

- **Go Back N**
 - A chaque segment émis est associée une temporisation, si elle arrive à échéance, on retransmet tous les segments émis depuis le segment dont la temporisation a expiré. L'évaluation de la valeur de cette temporisation est importante, elle est fondée sur le délai de propagation A/R (RTT)
- **Fast Retransmit and Fast Recovery**
 - Dès qu'un récepteur reçoit un segment hors séquence, il envoie un ACK avec le no du prochain octet à recevoir, et sauve le segment reçu en attendant de combler le trou dans le flot de données avec le segment manquant... si l'option SACK est permise et négociée à l'ouverture de connexion il informe sur les blocs bien reçus
 - Un émetteur qui reçoit plusieurs fois le même ACK (duplicated ACK) suspecte qu'il y a eu une perte de segment. Au bout de 3 ACKs identiques, il ré-émet le segment manquant. Ce phénomène est aussi un indicateur de congestion.

Calcul du Délai de Propagation A/R et de la temporisation de retransmission

- Le délai de propagation A/R est la base du paramétrage du délai de temporisation pour retransmission. Il est estimé en permanence.
- La difficulté tient à ce que le RTT évolue au plus près de la charge réelle du réseau traversé : une temporisation trop courte déclenche une retransmission sans raison, ce qui charge le réseau puis déclenche le slow-start de façon prématurée et sous-dimensionne le seuil de congestion.
- Mesure du RTT : temps séparant l'émission d'un segment de la réception de son acquit par le récepteur (segment non retransmit).
 - Si une mesure de RTT est en cours pour un segment, il n'y a pas de nouvelle mesure prise pour un nouveau segment émis.
 - Le RTT est mesuré en ticks d'horloge TCP, 1 tick vaut généralement 500ms dans une implantation standard... mais c'est à reconsidérer avec le Cloud et des liaisons intra-datacenter à 10 ou 40Gb/s

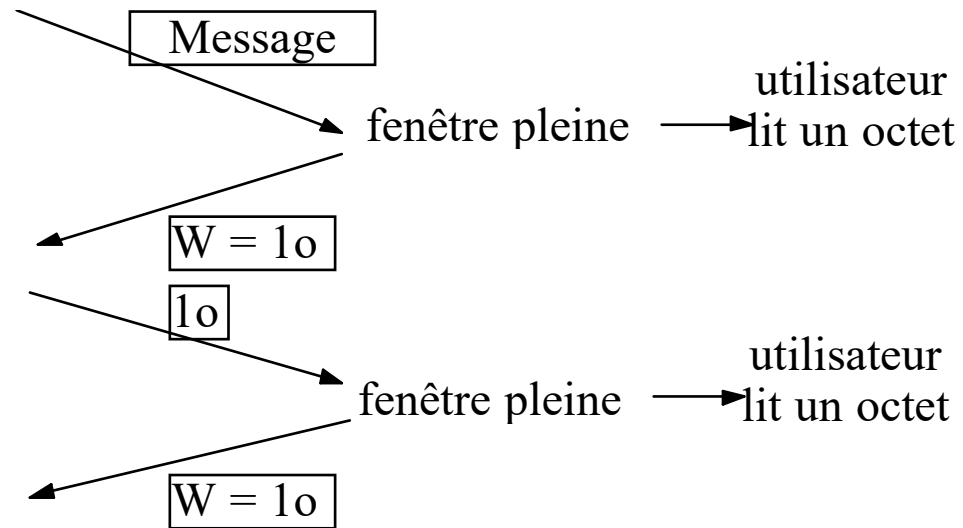
RTT – méthodes de calcul

- **1ère proposition : (RFC0793)**
 $R = a \cdot R + (1-a) \cdot M$, "Retransmit TimeOut" $RTO = R \cdot b$
R ancienne estimation du RTT, M dernier temps mesuré pour un RTT
a vaut généralement 0,9 et b vaut 2
Ce résultat ne prend pas en compte les grandes variations de RTT
- **2ème proposition : (Van Jacobson 1988)**
 $Err = M - A$
 $A = A + g \cdot Err$, A est une estimation du RTT moyen où $g = 1/8$
 $D = D + h (|Err| - D)$, D est une estimation de la variation moyenne où $h = 0,25$
 $RTO = A + 4D$
Ce résultat est meilleur. C'est la méthode standard.
- Une fois le RTO calculé, il augmente de façon exponentielle (en doublant) à chaque réémission... la transmission arrête au bout de 15 tentatives (peut aller jusqu'à une quinzaine de minutes)

Contrôle de flux TCP

- Mécanisme de fenêtre d'octets : chaque extrémité indique une taille de fenêtre, champ sur 16 bits (soit 65535 o max). Sa taille peut être étendue à l'aide de l'option `Window scale` à l'ouverture de connexion
 - « *Sur Linux systems, la valeur du facteur multiplicatif de la fenêtre est indiqué dans la valeur*
`/proc/sys/net/ipv4/tcp_window_scaling` »
- Le récepteur envoie son crédit en fonction des octets de données que retire son utilisateur.
- La valeur de la taille de la fenêtre active est indiquée dans chaque segment (Principe de crédit plutôt que de fenêtre. C'est pour mieux résister aux pertes de segments, en effet, si un émetteur ne reçoit pas son crédit, il ne reste bloqué que jusqu'au message suivant). Il existe un timer, "persist timer" qui oblige l'émetteur à demander le crédit périodiquement s'il n'y a pas d'échange.

Silly Window Syndrome



Silly Window Syndrome

(baisse de perf dramatique pour certains types d'applis)

Problème qui se produit quand on transmet de grands blocs côté émetteur, et que le récepteur ne lit ses données que par très petits bouts.

Solution :

- **Le récepteur** n'indique que des tailles de fenêtre dont la valeur est au moins un MSS, sinon supérieure à la moitié de la taille maximum de la fenêtre.
- **L'émetteur** respecte au moins une des conditions suivantes pour envoyer ses segments :
 - a) Envoi de segments de données de longueur MSS pas inférieur
 - b) On peut envoyer des données > moitié de la taille du crédit annoncé par le récepteur
 - c) S'il n'y a aucune donnée non acquittée il peut envoyer ce qu'il veut

Je n'ai pas abordé l'algorithme de Nagle... qui semble plutôt utilisé pour les applications interactives (frappe de caractères), il se résume dans https://fr.wikipedia.org/wiki/Algorithme_de_Nagle ainsi:

1. "Le premier octet reçu par TCP est envoyé immédiatement ;
2. Tant que l'accusé de réception n'a pas été reçu, les octets à envoyer sont stockés dans un tampon. Après l'[acquiescement](#) du premier octet, les données du tampon sont envoyées en un seul segment (mais si la taille du tampon atteint la taille maximale d'un segment, l'envoi des données a lieu) ;
3. On stocke de nouveau les données dans le tampon en attendant l'acquiescement.

Généralement, les implémentations TCP permettent la désactivation de l'algorithme de Nagle (correspondant typiquement à l'option TCP_NODELAY)"

Connexion vue comme un conteneur

- Pb encore plus important avec les réseaux à haut débit.

capacité d'une connexion (*bit* ou en *octet* si on / par 8)

= débit nominal (bit/s) * délai de propagation [\[1\]](#) A/R

encore appelé Delay Bandwith Product

[\[1\]](#) au niveau transport, ne pas confondre avec le niveau physique, ce délai de propagation A/R est spécifique à chaque connexion de Transport, il fait l'objet d'une évaluation périodique par la couche Transport

Type de Réseau	nominal b/s	nominal o/s	RTT (ms)/capacité (o)
Ethernet	10Mb/s	1,25Mo/s	3ms/3,750ko
FastEthernet	100Mb/S	12,5Mo/s	3ms/37,5ko
GigabitEthernet	1Gb/s	125Mo/s	3ms/375ko
Multiplex T1 cont	1,544Mb/s	0,193Mo/s	60ms/11,58ko
Multiplex T3 cont	44,736Mb/s	5,592Mo/s	60ms/335,52ko
Multiplex T1 sat	1,544Mb/s	0,193Mo/s	500ms/95,5ko

Dimensionnement de la fenêtre

- La capacité d'une connexion est à comparer avec la taille de la fenêtre.
- La taille de la fenêtre (65,5Ko) est trop petite pour certains réseaux. La taille de la fenêtre peut être augmentée avec l'option "window scale" qui permet de définir une taille de fenêtre plus grande (spec d'option : type = 3, lg = 3, valeur).
- Le champ option contient un facteur multiplicatif (valeur). Une valeur de 2 dans le champ option "window scale" donne une valeur de 65535×2^2 , soit 256 Ko. Cette option s'utilise à l'ouverture de cnx, par les deux entités (TCP est full-duplex, les entités sont toutes les deux émettrices).
- Valeur max du champ window scale : 14, facteur 2^{14} ce qui donne une fenêtre de taille max de l'ordre d'1 Go

Débit d'une cnx TCP – modèle analytique

- D'après Madhavi et Floyd (1997), le débit effectif d'une connexion TCP est:

$$B = 1,22 * MTU / (RTT * \sqrt{\text{taux de perte}})$$

- On suppose que le MTU est celui de la connexion et que les datagrammes sont de même longueur et de taille MTU.
- Ce résultat provient d'une analyse du mécanisme d'évitement de congestion en régime permanent pour une connexion.



Merci pour votre attention !!!