

Exercices dirigés

UTC505/USRS4D

-Transport TCP-

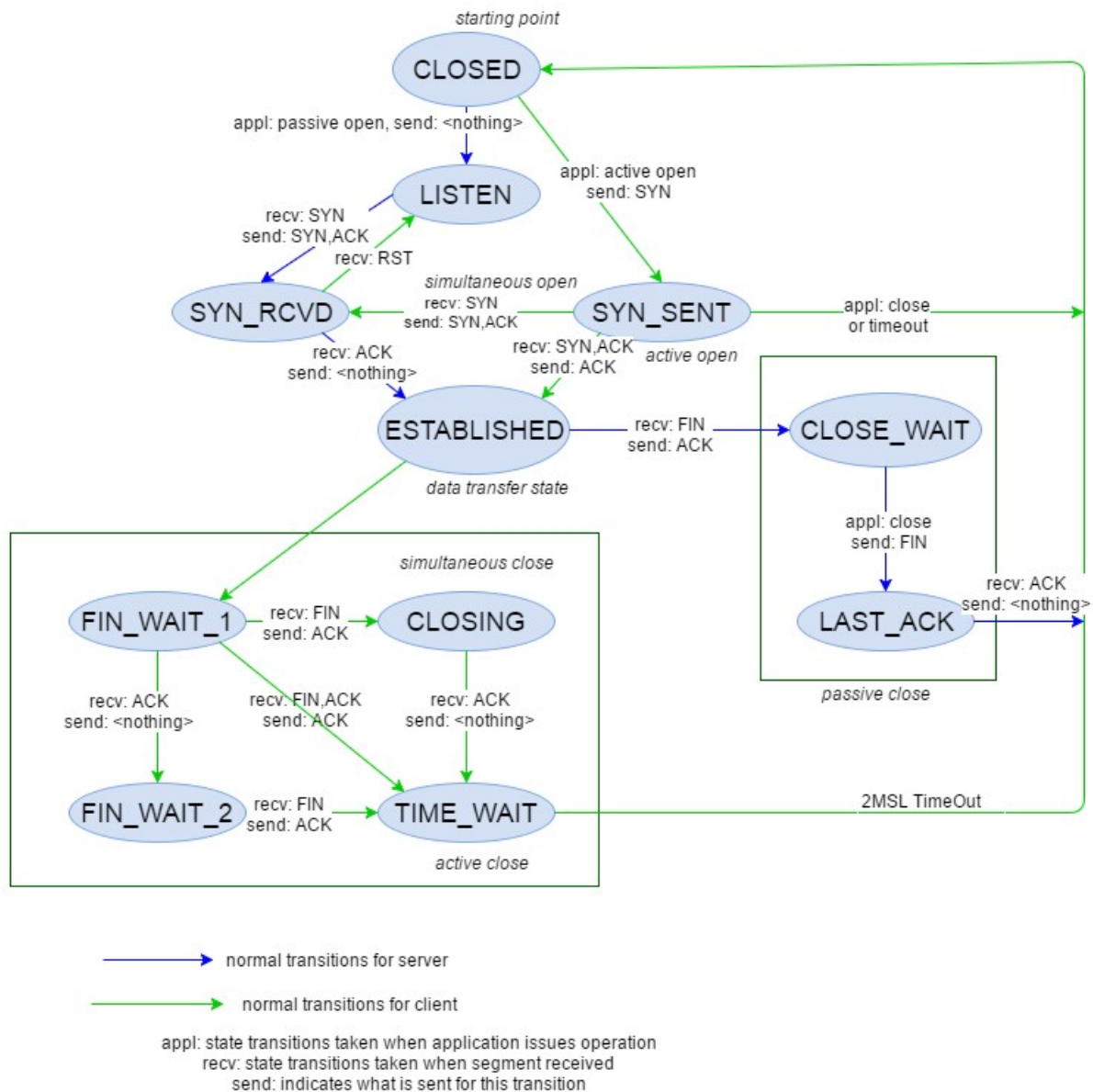
E. Gressier-Soudan

2021-2022

Ce polycopié a été élaboré par l'équipe enseignante "Réseaux et protocoles" à partir d'exercices rédigés par MM. Florin, Gressier-Soudan qu'ils en soient ici remerciés.

ED•TCP

Le diagramme d'état du protocole TCP (Transmission Control Protocol) va servir pendant tout l'ED. Il est issu de la RFC793.



TCP Protocol State Diagram

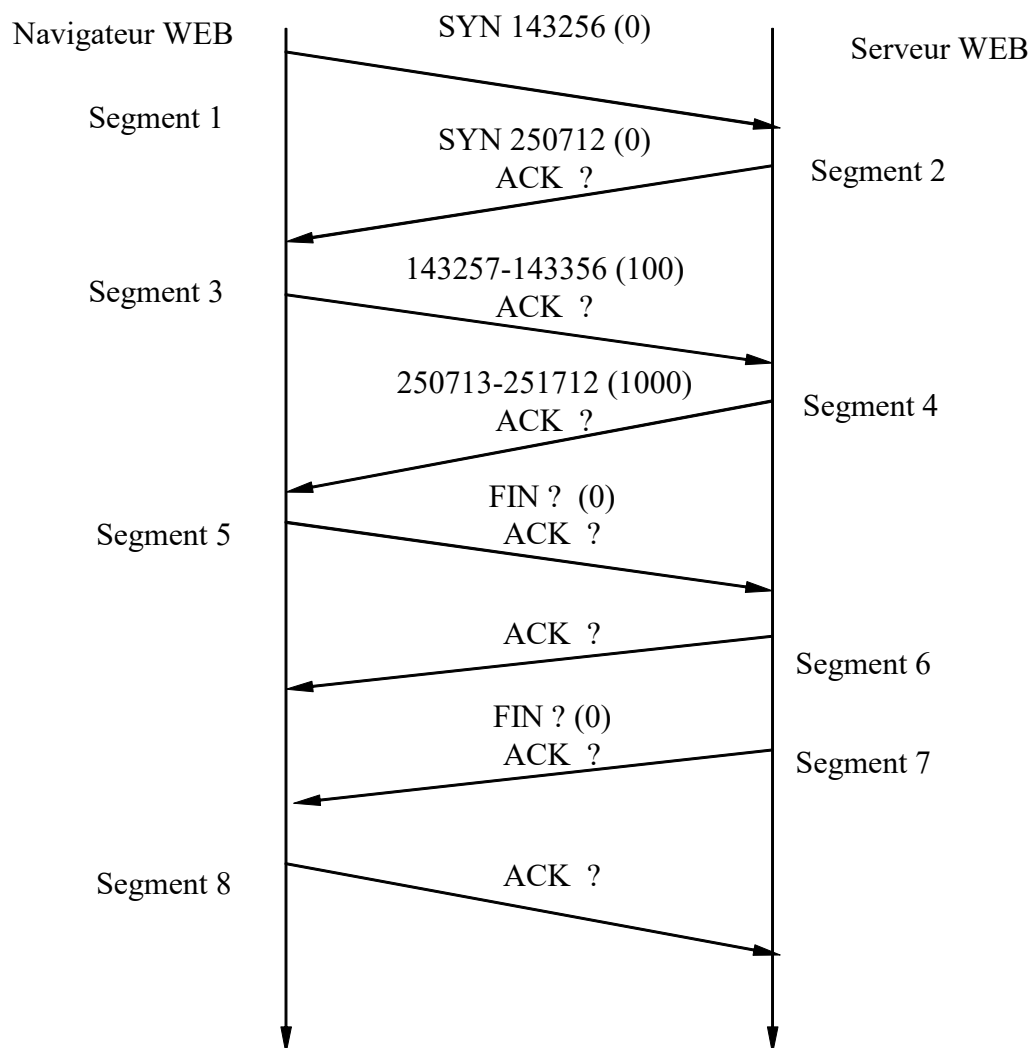
Source : <https://www.thedailyprogrammer.com/2016/09/tcp-state-transition-diagram.html> (consulté le 20/08/2020)

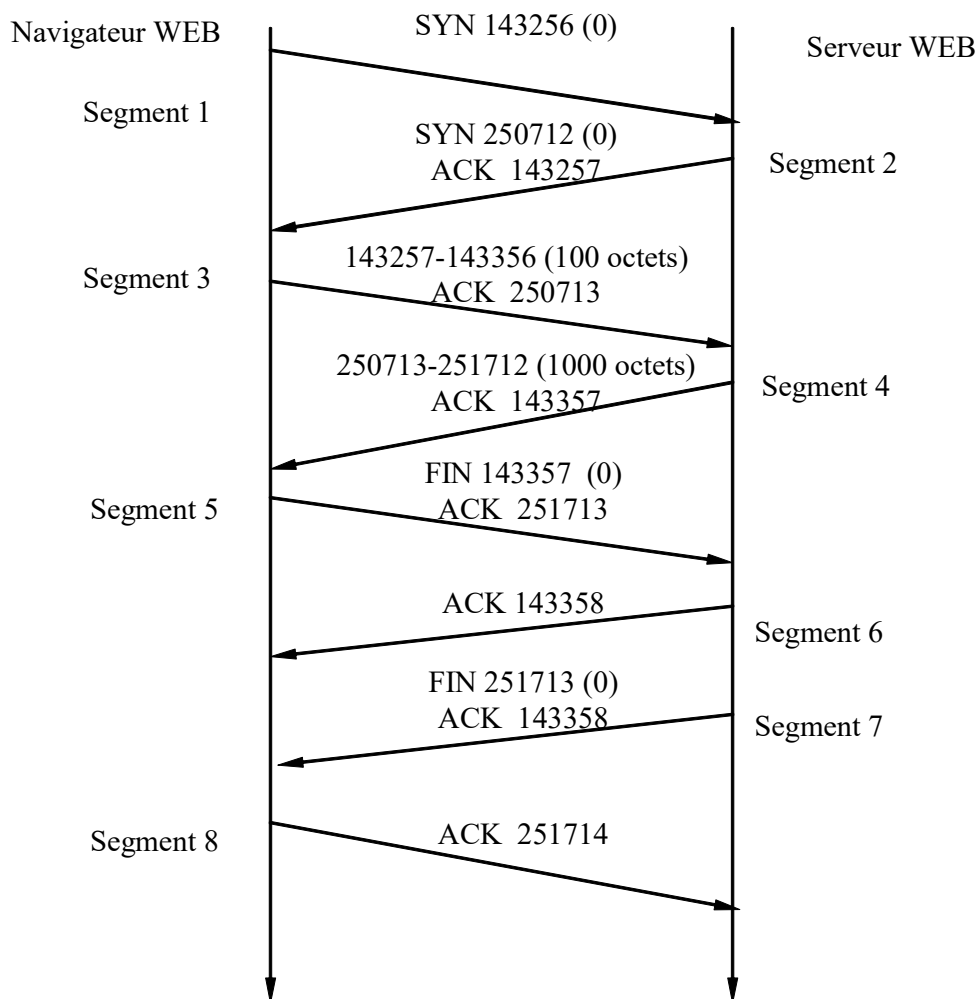
Exercice 1 : Numérotation des octets dans un échange TCP

L'échange TCP de la figure suivante correspond au transfert d'une page WEB entre un navigateur WEB et un serveur WEB. On fait l'hypothèse que la requête à la page WEB fait 100 octets et que la page WEB retournée fait 1000 octets. Il n'y a pas d'erreurs de transmission.

Pour chaque segment de données, différentes informations apparaissent. D'une part la présence d'un ou plusieurs des différents indicateurs comme SYN, FIN, ACK (contrôle). Par ailleurs sur la première ligne deux chiffres sont portés. Le premier chiffre correspond au numéro de séquence du premier octet du segment, le deuxième chiffre correspond au numéro du premier octet du prochain segment à envoyer. Le chiffre entre parenthèses correspond au nombre total d'octets transmis dans le segment. Si le segment est porteur d'un acquittement positif, l'indicateur ACK est mentionné et à côté de lui doit figurer la valeur du champ acquittement du segment TCP.

Complétez les numéros de séquence et les numéros d'acquittement qui manquent sur la figure (qui apparaissent sous forme de point d'interrogation). Indiquez à quoi correspondent les différents segments numérotés de 1 à 8.



Correction :

Au sens de TCP le client correspond au navigateur, et le serveur correspond au serveur web.

- L'ouverture de connexion porte sur les segments 1, 2 et 3, c'est un three way handshake classique dans TCP, avec le client (à gauche) qui est à l'initiative de l'ouverture de connexion, et le serveur (à droite).

Le segment 1 : c'est un SYN du client (ouverture de connexion active) vers le serveur (ouverture de connexion passive). Il compte pour un octet. Il contient le numéro de séquence du premier octet de la connexion du sens client vers serveur, soit 143256.

Le segment 2 : c'est un SYN ACK du serveur vers le client qui montre que le serveur a accepté l'ouverture de connexion dans le sens client vers serveur en indiquant dans le ACK le numéro du prochain octet à recevoir par le serveur, soit 143257. Le serveur indique son numéro de premier octet dans le sens serveur vers client, soit 250712.

Les segments 1 et 2 ne transportent aucun octet de donnée.

Le segment 3 a un double rôle : fin du protocole d'ouverture de connexion, et début du transfert de données (ici une requête web). Dans son rôle de fin d'ouverture de connexion, il acquitte l'ouverture de connexion dans le sens serveur vers client, en indiquant dans le ACK le numéro de prochain octet à recevoir dans le sens serveur vers client, soit 250713.

- Le transfert de données porte sur les segments 3 et 4.

Le segment 3, envoyé par le client représenté par un navigateur web, contient donc aussi des données, plus exactement 100 octets de données qu'on peut imaginer être le

contenu d'une requête web. Les octets envoyés sont numérotés de 143257 à 143356. L'ACK superposé correspond à 250713 comme vu ci-dessus.

Le segment 4 est la réponse du serveur, serveur web, au client. Il contient 1000 octets de données qu'on peut imaginer être le contenu d'une page web. Les octets envoyés sont numérotés de 250713 à 251712. Il acquitte les données dans l'autre sens donc ACK porte le numéro 143357; ce qui veut dire qu'il acquitte tous les octets émis avant l'octet 143357 par le client.

- La fermeture de connexion porte sur les segments 5, 6, 7, 8.

Elle s'exécute en 4 messages, dans sa forme la plus "bavarde", c'est-à-dire par fermeture de chacun des sens de circulation de données de la connexion. Ici la fermeture est à l'initiative du Client. Le signal FIN compte pour un octet, c'est un octet fantôme. Les fermetures de connexion ne contiennent pas de données.

Le segment 5 ferme la connexion dans le sens client vers serveur avec un FIN qui porte le numéro de séquence 143357. Il acquitte les octets reçus dans le sens serveur vers client, et donc attend l'octet 251713.

Le segment 6 acquitte la demande de fermeture portée par le segment 5. Elle acquitte en portant le numéro du prochain octet attendu 143358 même si il n'y en aura plus d'envoyé suite à la demande de fermeture de ce sens de la connexion.

Le segment 7 ferme la connexion dans le sens serveur vers client avec un FIN qui porte le numéro de séquence 251713. Il acquitte les octets reçus dans le sens inverse avec l'ACK 143358 (en fait ce numéro n'a pas progressé mais il rappelle le numéro de prochain octet à recevoir quand même).

Segment 8, le client acquitte la demande de fermeture de connexion portée par le segment 7. L'ACK porte le numéro 251714.

Exercice 2 : Echange de données entre un client et un serveur FTP.

Question 1

Le protocole FTP, File Transfer Protocol appartient à quelle couche du modèle ISO-OSI ? Justifiez brièvement votre réponse.

Correction :

Le protocole FTP est un protocole de couche application. Il gère toutes les opérations liées à l'accès à des fichiers distants en vue de les transférer.

Attention, ce n'est pas une application. Il est de même niveau que HTTP.

Question 2

Expliquez pourquoi le protocole FTP nécessite l'utilisation de TCP, Transmission Control Protocol.

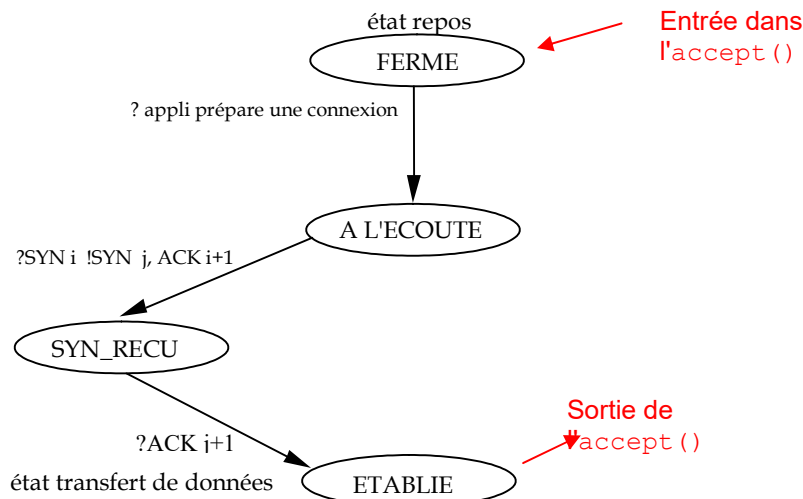
Correction :

Un transfert de fichiers nécessite que la totalité des informations échangées parviennent bien au destinataire. Un transfert d'opérations bancaires, l'échange d'un fichier personnel... nécessitent que la totalité des octets arrivent dans l'ordre et sans qu'il en manque un seul. TCP offre un mécanisme d'échange de données fiable et ordonné. C'est le protocole de transport nécessaire pour un transfert de fichiers.

Question 3

Soit la partie d'automate TCP suivante :





On rappelle que "?" représente une condition, et "!" une action, ici ce sont des envois de messages.

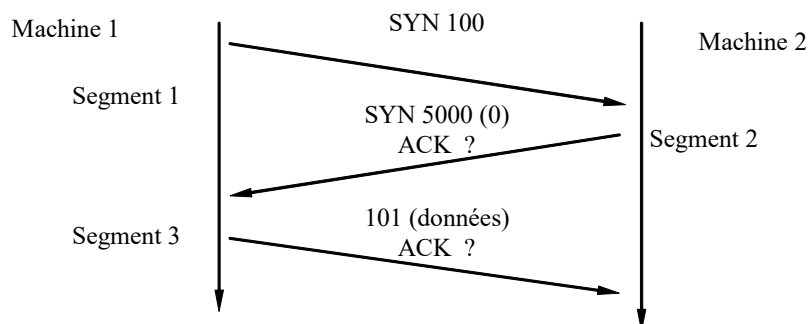
Est-ce que cette partie de l'automate protocolaire correspond à l'ouverture de connexion active ou à l'ouverture de connexion passive ? Justifiez votre réponse.

Correction :

Cette partie de l'automate protocolaire TCP correspond à une ouverture de connexion passive : état "A L'ECOUTE" avec attente d'un SYN. C'est la partie exécutée par le serveur généralement.

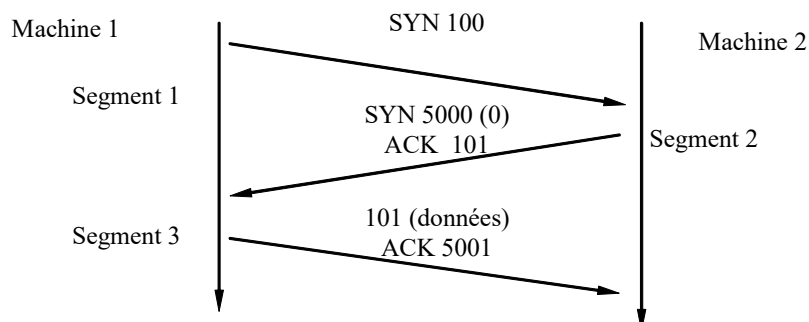
Question 4

Avec un espion de ligne, on constate l'échange suivant lors d'une ouverture de connexion TCP pour un transfert FTP.



Complétez le schéma ci-dessus en donnant les numéros portés par les acquittements des segments échangés lors de l'ouverture de connexion TCP.

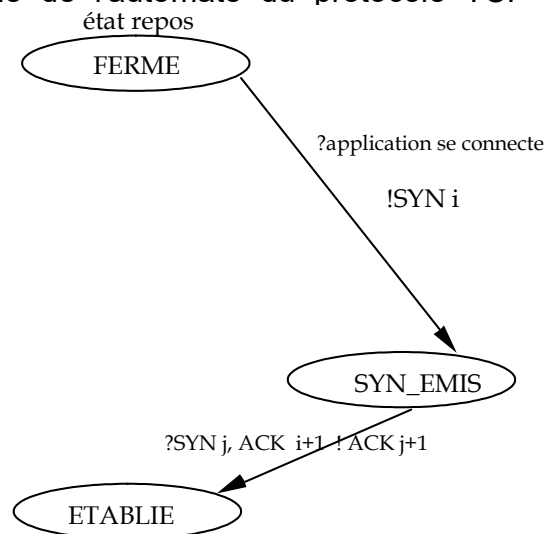
Correction :



Le numéro de séquence d'octet porté par un segment est acquitté dans le sens inverse à l'aide du mécanisme de piggybacking. Le numéro d'octet pour l'acquittement indique le numéro du prochain octet à recevoir.

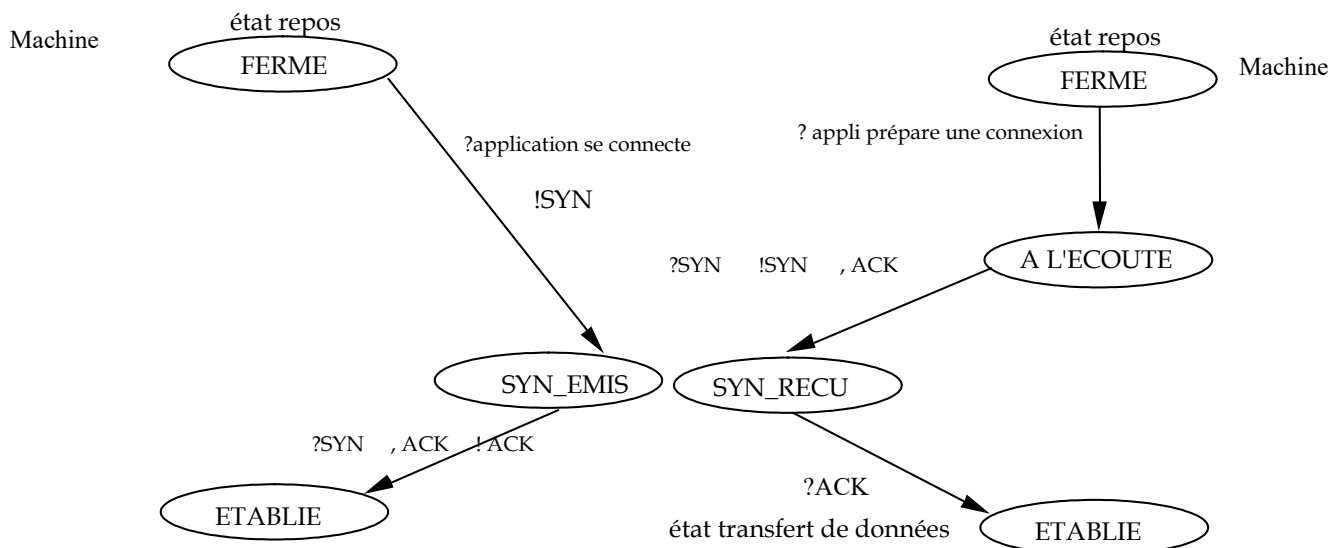
Question 5

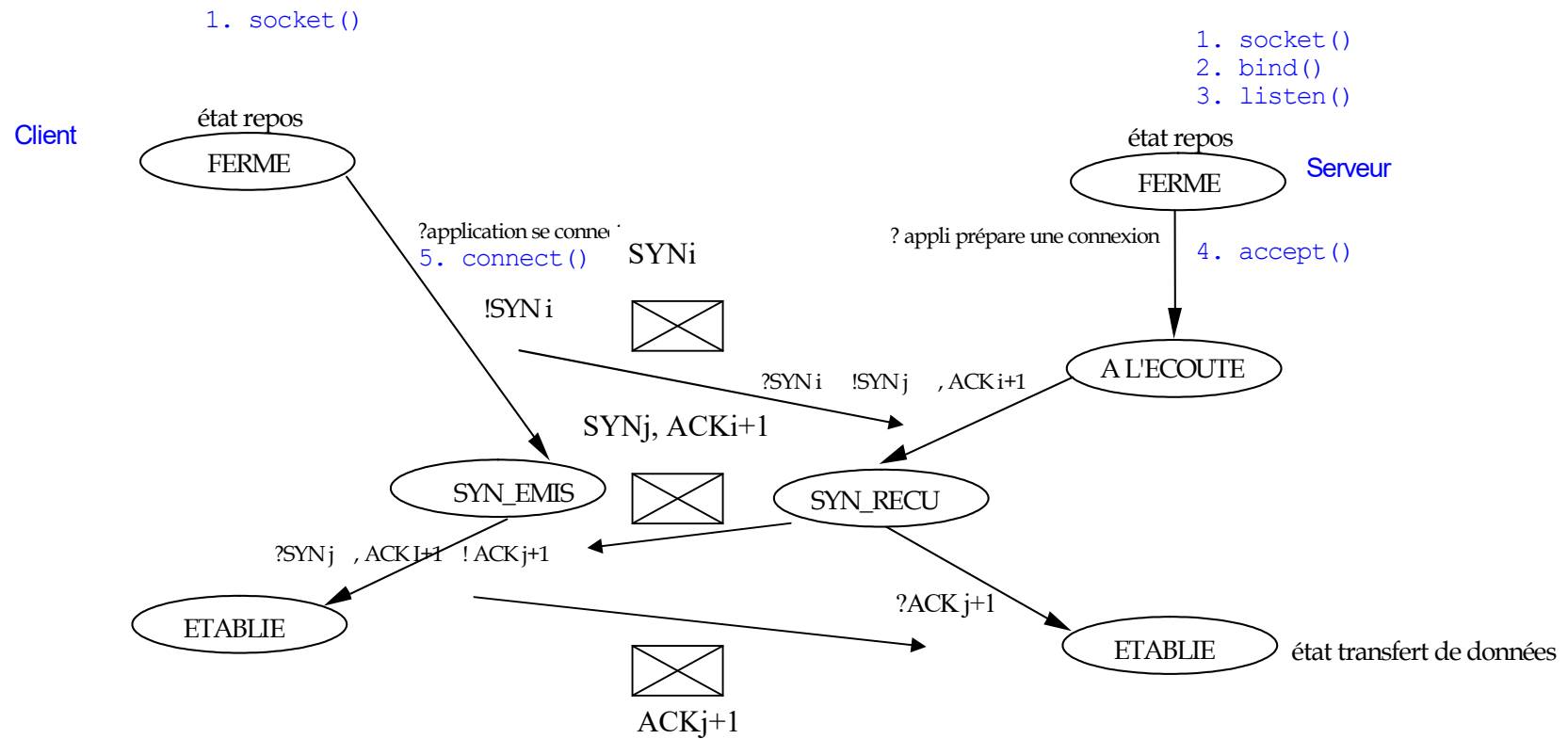
Voici une autre partie de l'automate du protocole TCP impliquée dans l'ouverture de connexion.



Sur le schéma ci-après, portez le type des machines (client ou serveur) pour chaque partie d'automate qu'elles exécutent, et les différents numéros de séquence issus de la trace des messages échangés indiquée en question 4. Il faut donner les valeurs de l'échange qui doivent remplacer les variables i et j dans les figures ci-dessus.

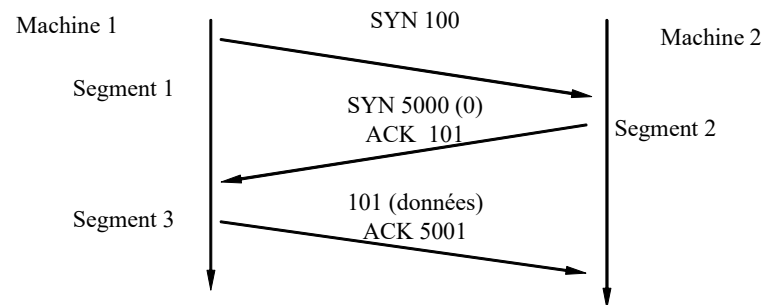
Puis, dans l'API Socket, à quels appels système de l'API socket (`listen()`, `close()`, `shutdown()`, `connect()`, `select()`, `accept()`, `read()`, `write()`, `recvfrom()`, `sendto()`...) correspondent les parties d'automates du protocole TCP dans les questions précédentes.



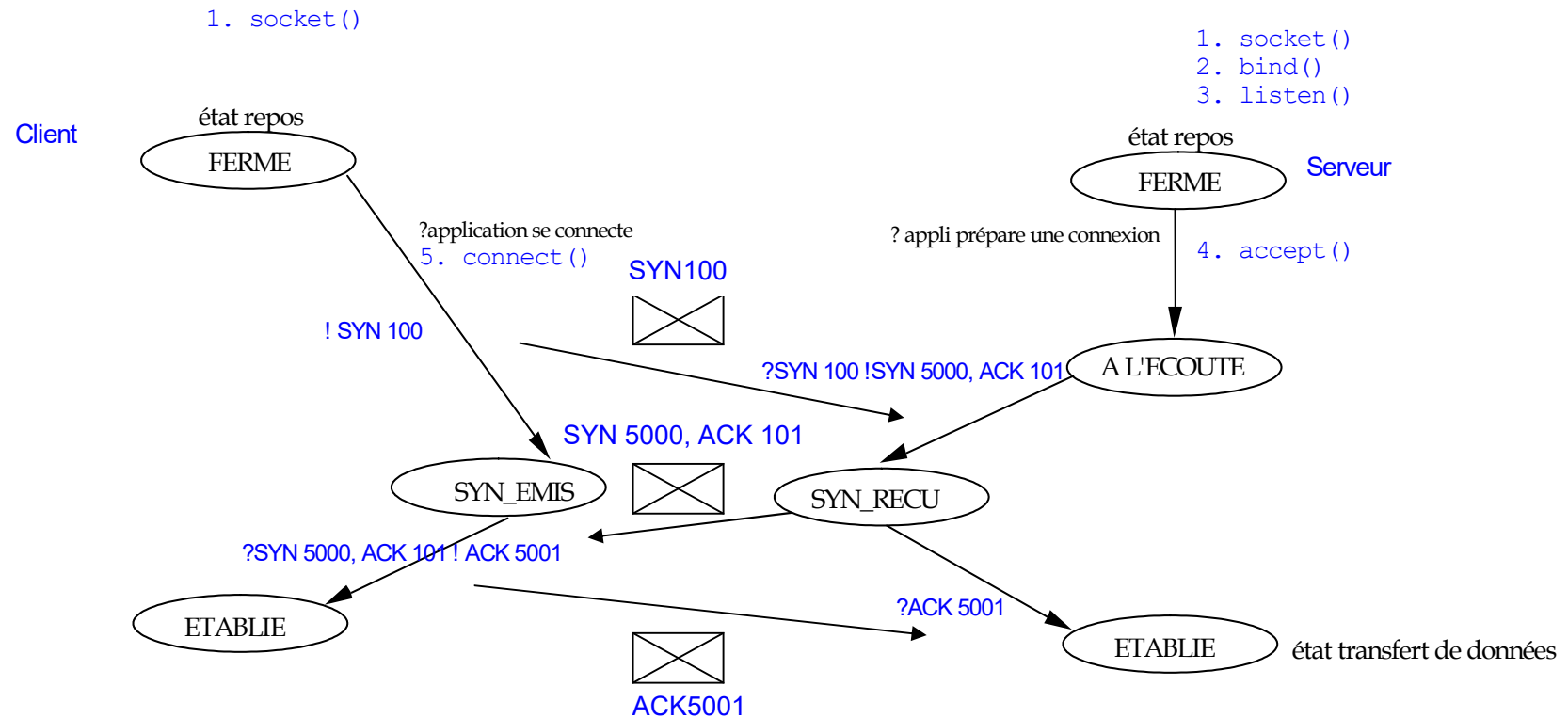
Correction :

Maintenant on ajoute sur le dessin les numéros de séquence échangés à travers les SYN et ACK de l'ouverture de connexion :

On a donc :



Ce qui donne sur la figure avec les automates TCP communiquants pour une ouverture de connexion :

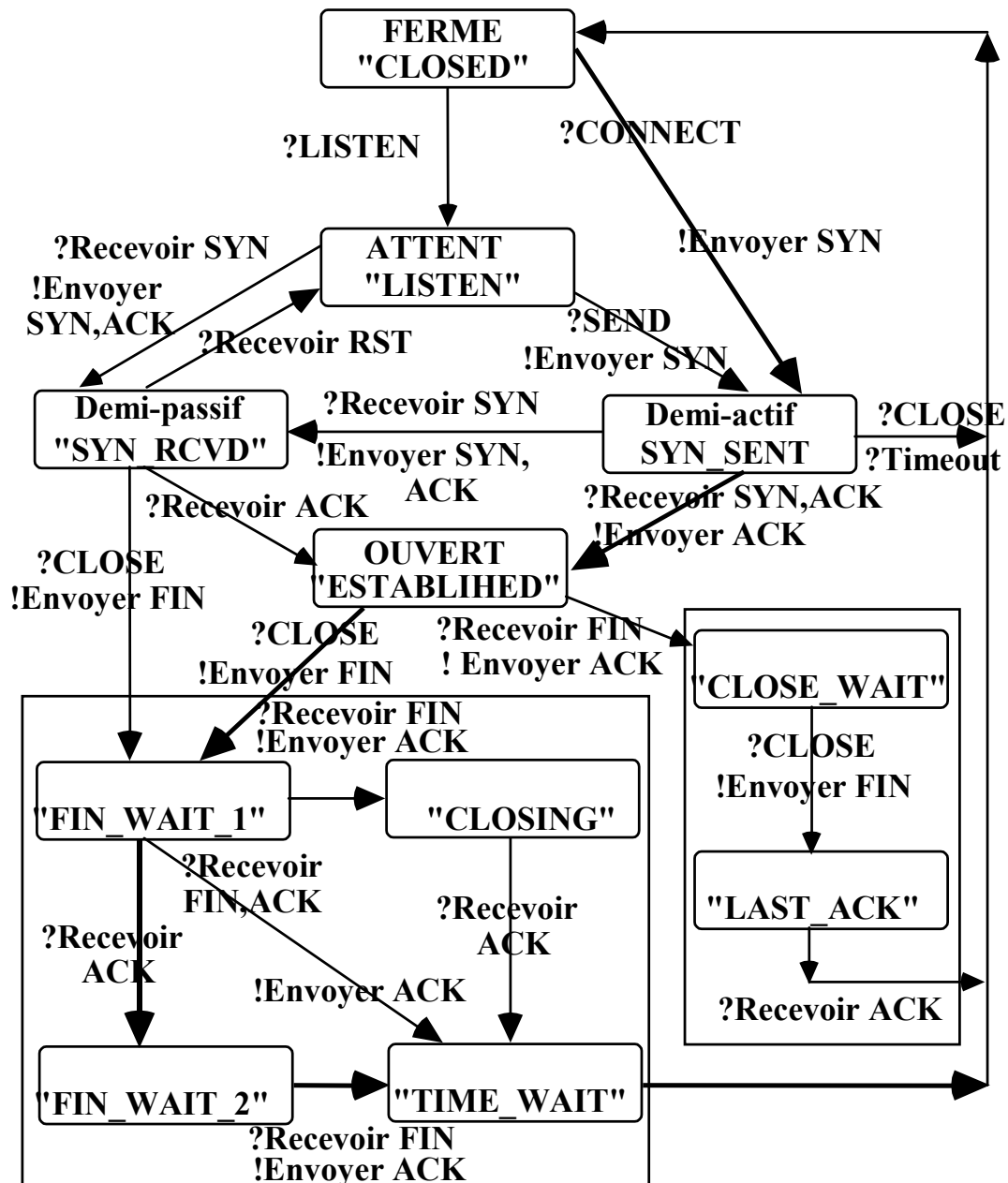


Exercice 3 : Automate protocolaire TCP – fermeture de connexion – Exercice cours (et pas court)

Rappelez les principales fonctions réalisées par TCP?

Examinez l'automate en suivant les transitions portées en traits plus épais. Il s'agit d'une ouverture de connexion, un état de transfert en régime connecté, suivi d'une fermeture. Analysez les différentes étapes d'une ouverture de connexion entre un client et un serveur.

Que se passe-t-il lors de la fermeture de connexion?

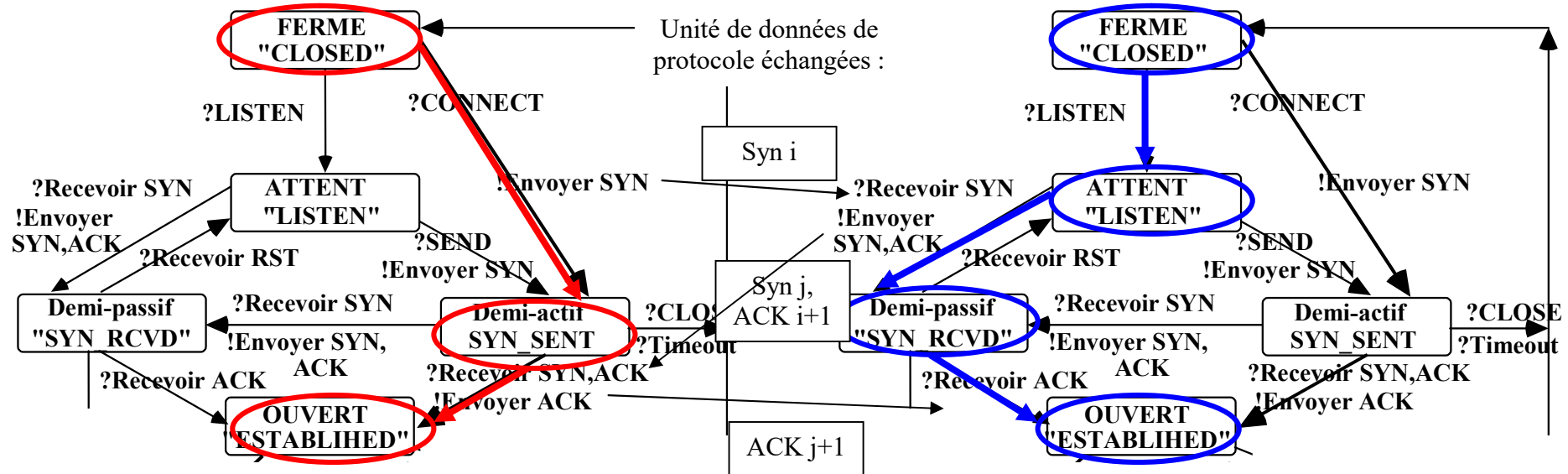


Correction :

TCP est un protocole de la couche Transport. Son rôle est d'acheminer les informations d'une application à une autre de façon fiable, sans erreur, sans saturation de la boîte aux lettres de réception du récepteur, en tentant d'anticiper la congestion du réseau Internet pour l'essentiel. C'est un protocole orienté connexion (il faut ouvrir une connexion pour communiquer, et, libérer la connexion une fois les échanges finis). C'est un protocole orienté flot d'octets, il ne préserve pas les limites d'un enregistrement.

Dans TCP les rôles des entités communicantes sont dysymétriques : il y a un rôle serveur (qui se met prêt à satisfaire des demandes d'ouverture de connexion), et un rôle client (qui initialise la connexion en faisant la requête d'ouverture auprès du serveur). Pour expliquer le fonctionnement du protocole, on va utiliser un automate TCP chez le client, et un automate TCP chez le serveur. Cela reflète exactement ce qu'il se passe lors d'une mise en relation d'entités TCP d'un bout à l'autre d'un réseau. Garder à l'esprit qu'un automate TCP travaille pour une application cliente ou une application serveur. Et qu'il y a autant d'instances d'un automate que d'applications (processus sur les unix en général) qui font des échanges client-serveur en mode flot d'octets fiable.

Rappelons que les carrés figurent des états, qu'on passe d'un état à un autre si une condition précédée par un "?" se réalise (devient vraie) et en exécutant une action précédée par un "!".

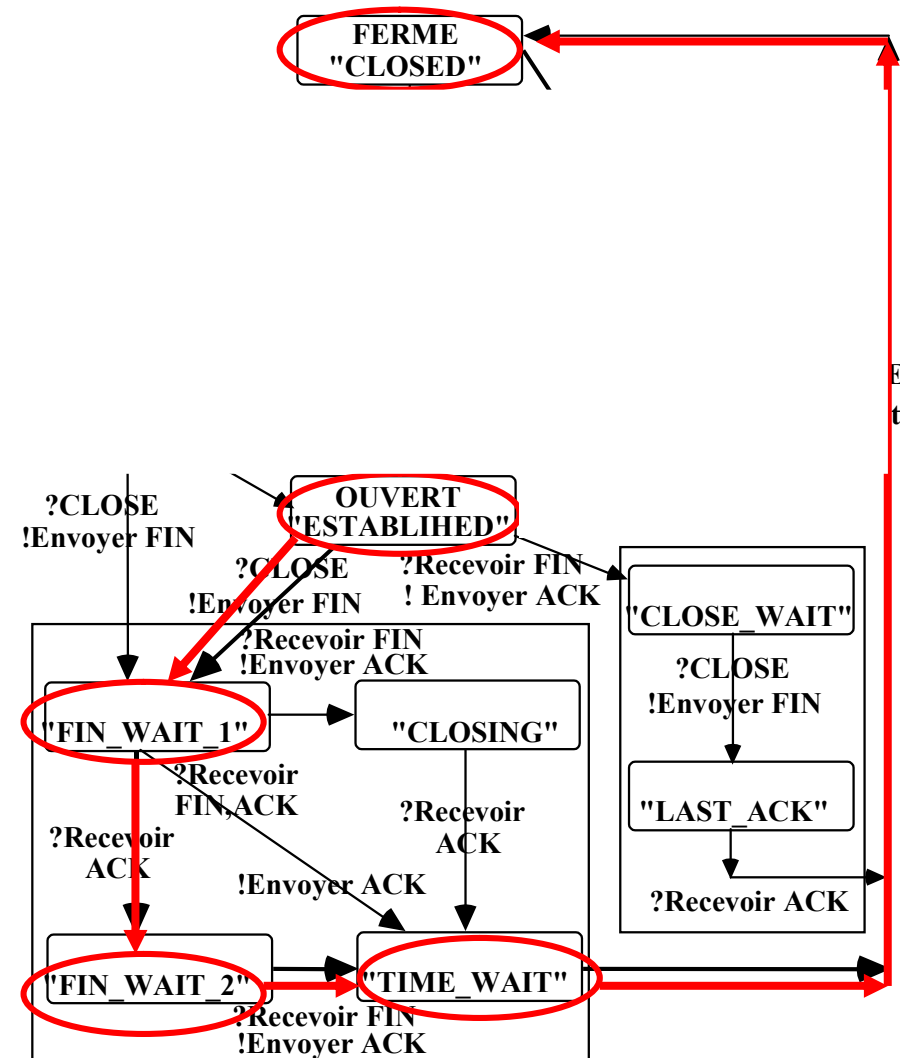
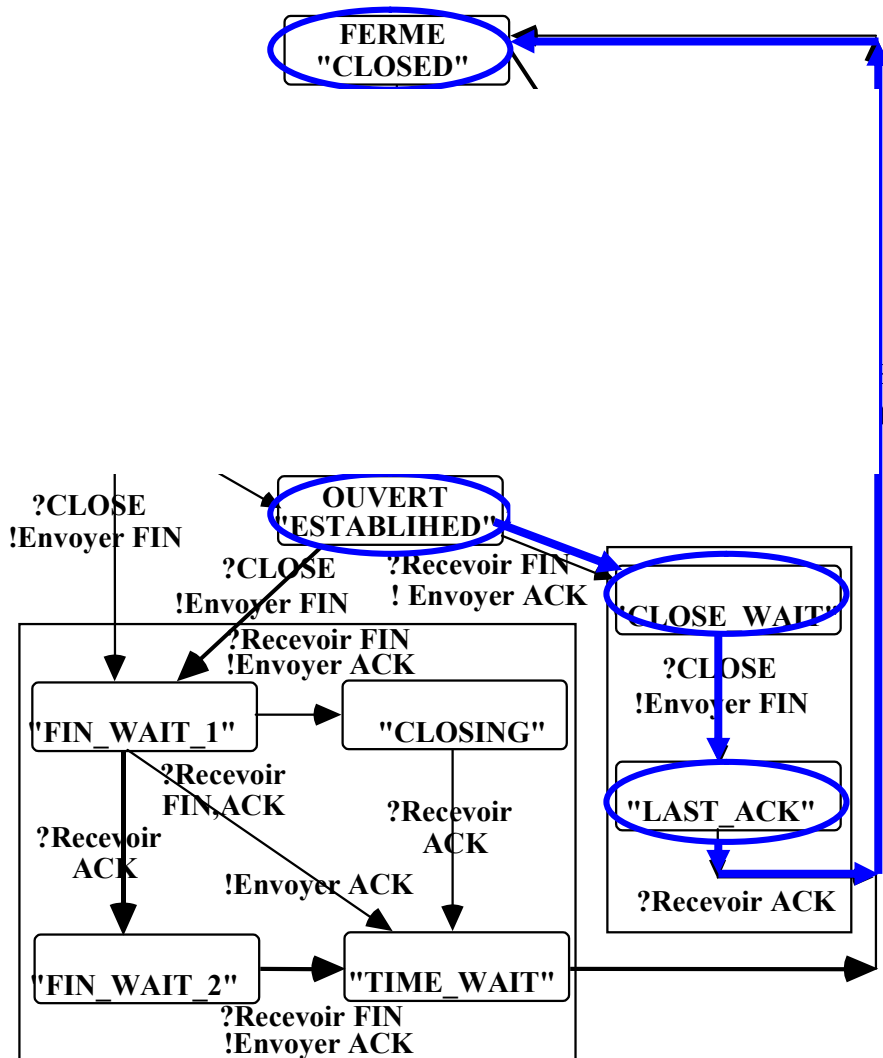


Ci-dessus les automates TCP client (à gauche), et TCP serveur (à droite) limités à la partie ouverture de connexion. L'évolution de l'automate TCP client est en rouge, l'évolution de l'automate TCP serveur est en bleu. On retrouve le three way handshake présenté en cours. Les autres arcs et changements d'états possibles sont liés à des formes d'ouvertures de connexion moins fréquentes :

- Passage de l'état ATTENT "LISTEN" à l'état Demi-actif SYN_SENT, se produit quand le serveur reçoit l'ordre de l'application d'envoyer un segment (nécessite de ne pas avoir bloqué le serveur dans l'état accept(), donc d'utiliser des E/S non bloquantes). Ceci revient à faire basculer le serveur d'un mode passif à un mode actif, comme une application dans le rôle client.
- Dans l'état Demi-actif SYN_SENT on peut passer à l'état FERME "CLOSED" soit par l'application qui effectue un close(), soit sans réponse du serveur par l'arrivée à échéance d'une temporisation (timeout).
- Dans l'état Demi-actif SYN_SENT on peut passer à l'état Demi-passif "SYN_RCVD" si on reçoit un SYN. Ceci s'interprète comme si les deux applications se comportaient dans un rôle client.
- Enfin, on peut passer de l'état Demi-passif "SYN_RCVD" à l'état ATTENT "LISTEN" si l'automate TCP dans le rôle serveur

Pour la phase de transfert des données à travers le réseau entre les deux automates protocolaires TCP, les deux applications sont dans l'état OUVERT "ESTABLISHED".

Pour la phase de déconnexion, on procède de la même façon que pour l'ouverture de connexion. Il y a a priori un rôle asymétrique : c'est le client qui demande le premier la fermeture de connexion, puis le serveur le demande à son tour.



Côté Client (en rouge):

- Le client ferme la connexion en premier sur une demande de fermeture, close(), de la part de l'application. Cela provoque l'envoi d'un segment FIN, le FIN compte un octet. L'automate protocolaire TCP passe à FIN_WAIT_1. Pour résumer, c'est comme si on fermait un côté de la connexion, celui qui correspond au sens client vers serveur, plus aucune donnée ne sera envoyée dans ce sens.

- Quand il reçoit un ACK pour son FIN, il passe à FIN_WAIT_2.
- Quand il reçoit la demande de fermeture du serveur, il acquitte et passe à l'état TIME_WAIT.
- L'automate TCP de l'exercice ne mentionne pas le fait qu'il reste dans cet état pendant $2 \times \text{MSL}$ (MSL = Maximum Segment Life Time, ce qui veut dire durée vie maximum d'un segment dans l'internet. C'est un paramètre de configuration, je renvoie au support de cours. C'est pour geler la référence à la connexion, pour qu'elle ne soit pas réutilisée pendant un certain temps afin d'éliminer les éventuels "vieux segments" qui s'inséreraient dans une nouvelle instance de cette connexion.
- Une fois la durée 2 fois MSL attendue, l'automate repasse à l'état initial pour la connexion FERME "CLOSED".

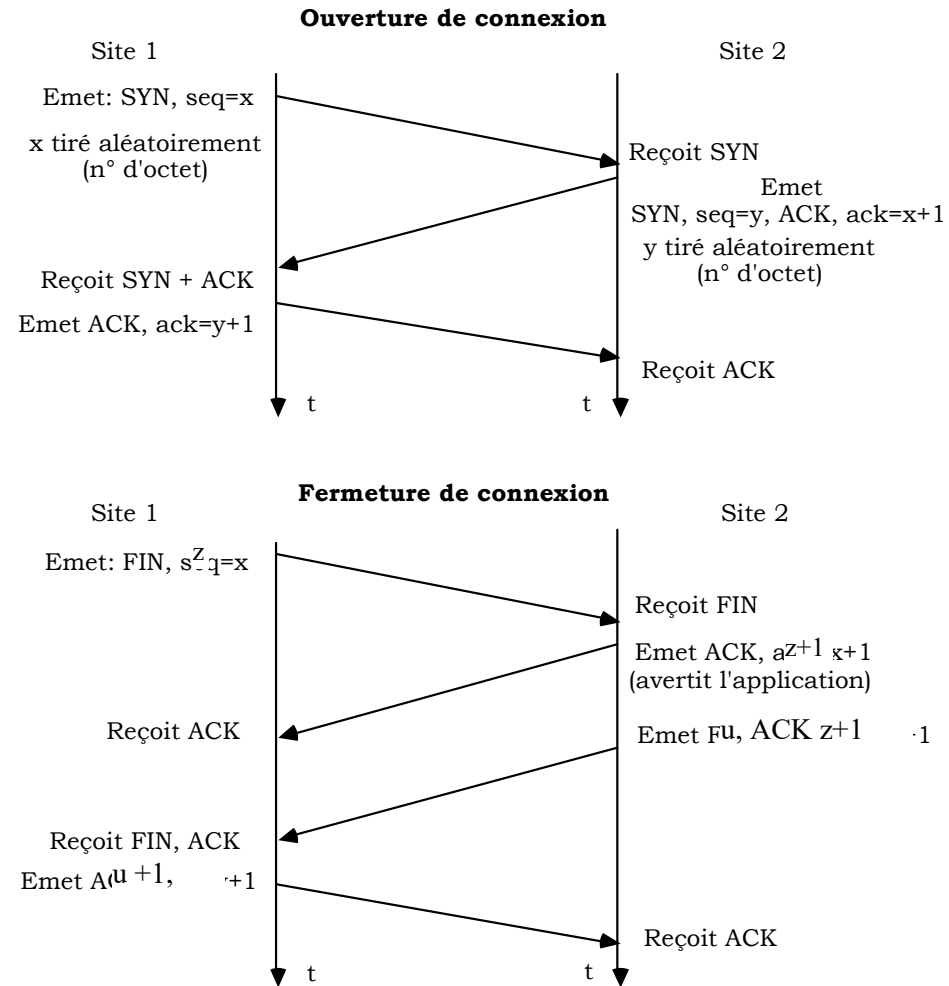
Côté Serveur (en bleu)

- Il attend une fermeture de connexion matérialisée par un segment marqué FIN du client. Quand il le reçoit il envoie un ACK et passe dans l'état "CLOSE_WAIT".
- Quand il reçoit un ordre de fermeture de l'application, il envoie un FIN au client à son tour et passe à l'état "LAST_ACK".
- Quand il reçoit l'ACK du client, l'automate protocolaire TCP bascule à l'état FERME "CLOSED", la connexion est devenue inactive.

Ceci, c'est le mode de fermeture de connexion le plus standard. On retrouve ce schéma lors d'un échange de requête WEB : envoi d'un lien puis réception de la page web. HTTP fonctionne au dessus de TCP.

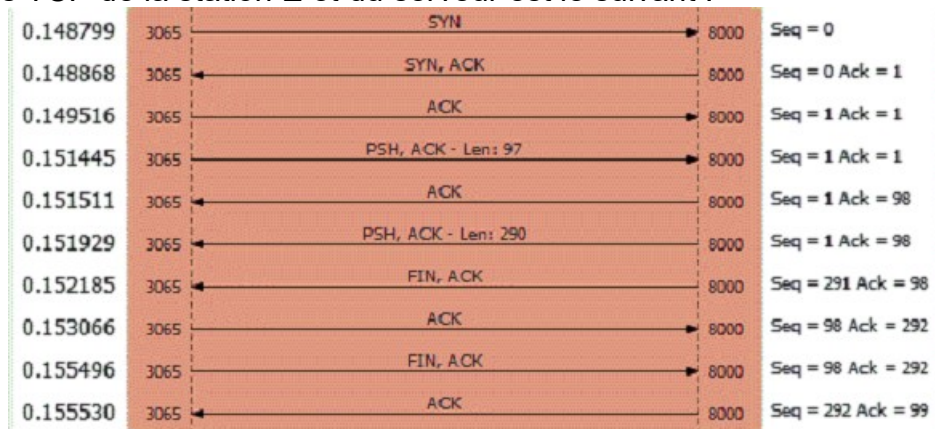
Les autres arcs pour les changements d'états correspondent à des modes de fermetures différents. Par exemple deux fermetures de connexion qui se croisent.

Dans le fonctionnement des automates on retrouve le comportement donné en cours, et souvent présenté avec des barres temporelles verticales avec un enchaînement d'échanges de PDU (protocol data units):



Exercice 4 : Fonctionnement TCP à travers un échange Web

Un navigateur E, client, 192.168.0.1 (à gauche) demande à un serveur web distant, 192.168.0.2 (à droite) une page via HTTP. Le graphe des échanges entre les automates protocolaires TCP de la station E et du serveur est le suivant :



Ce graphe est obtenu dans Wireshark avec la commande "Graphique des Flux" dans le menu "Statistiques". Quand il y a des données, le champ "Len : valeur" apparaît, sinon, c'est qu'il n'y a aucune donnée dans le segment.

L'identifiant de l'association entre E et le serveur si le numéro de port TCP de E est 3065 et si le numéro de port TCP du serveur est 8000 est :

(192.168.0.1, 3065, TCP, 192.168.0.2, 8000)

Cette association est mise en œuvre par une connexion TCP, comme indiquée. Pour nommer l'instance de la connexion TCP qui porte cette association, il faut ajouter les numéros de séquence initiaux (ISN, Initial Sequence Number) de chacune des entités communicantes aux informations de l'association.

La différence entre la connexion TCP, et l'instance de connexion TCP correspond à la différence : relation entre client et serveur, et, relation effectivement active entre le client et le serveur matérialisée par l'état "ESTABLISHED" des automates protocolaires respectifs.

Question 1

Les octets sont comptés en absolu. La valeur du numéro du premier octet envoyé par E pour ouvrir la connexion est : 3753C517 qui vaut en décimal 928236823. La valeur du numéro du premier octet envoyé par le serveur lors de l'ouverture de la connexion est : EC1E0D86 qui vaut en décimal 3961392518.

Donner l'identifiant complet de l'instance de cette connexion entre E et le serveur.

Correction :

L'identifiant de l'instance de la connexion TCP est donc

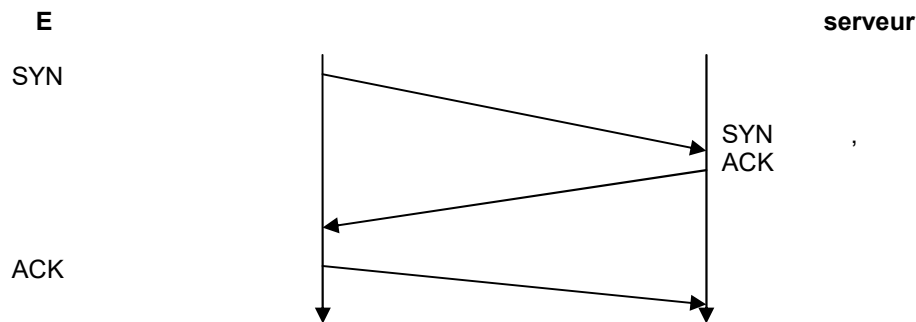
(192.168.0.1, 3065, 0x3753C517, TCP, 192.168.0.2, 8000, 0xEC1E0D86).

Ou avec la numérotation décimale :

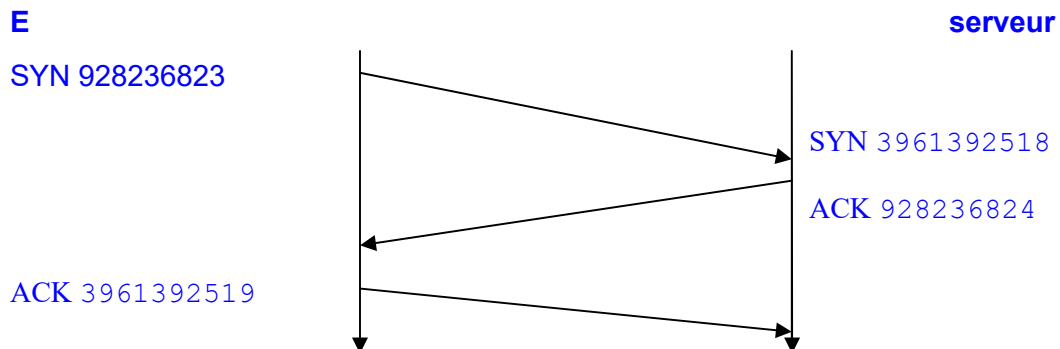
(192.168.0.1, 3065, 928236823, TCP, 192.168.0.2, 8000, 3961392518).

Question 2

Ajouter les numéros d'octets portés par les segments d'ouverture de connexion échangés entre E et le serveur sur le dessin ci-dessous :

**Correction :**

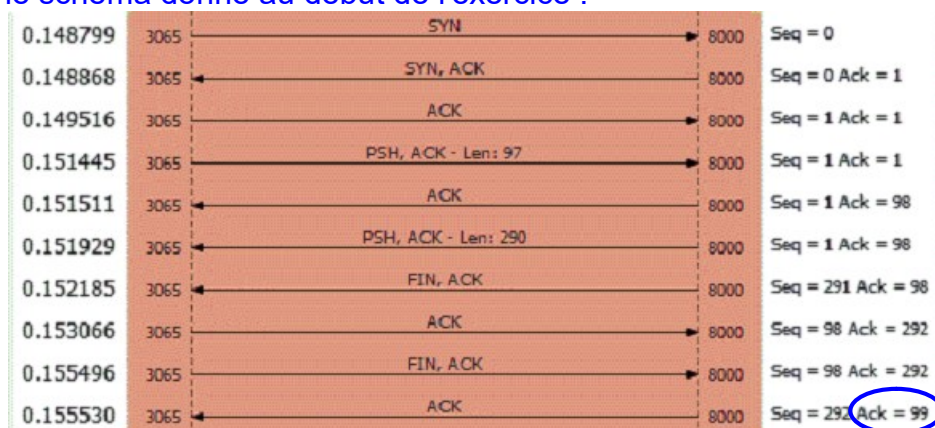
De E vers le serveur – du serveur vers E
(commence à 928236823)- (commence à 3961392518)

**Question 3**

La requête HTTP est portée par un segment de type PSH, ACK. Combien d'octets de données de charge utile pour TCP ont été envoyés de E vers le serveur pour cette requête ? Expliquez brièvement votre résultat. Aidez vous du dessin ci-dessous pour justifier votre réponse.

Correction :

On reprend le schéma donné au début de l'exercice :



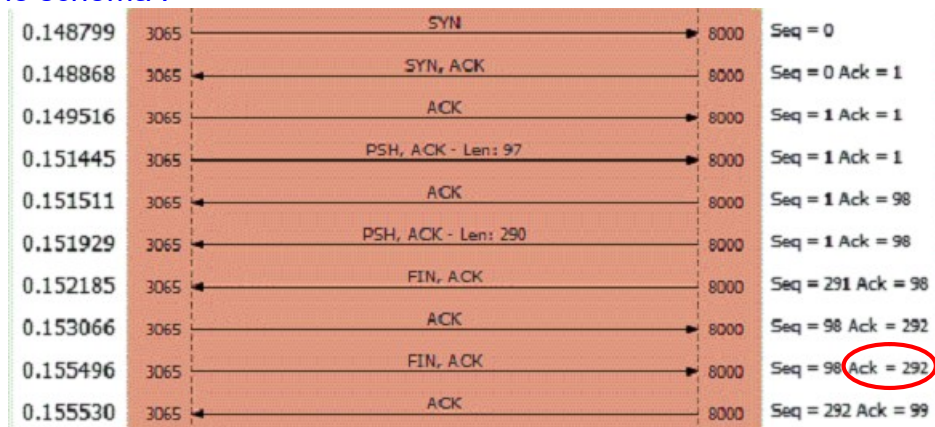
Le serveur dans son dernier envoi attend l'octet 99 puisqu'on a "Ack=99" (entouré en bleu). Cela veut dire qu'il acquitte tous les octets qui vont de 0 à 98 (en relatif). Mais l'octet 0, SYN, et le dernier octet FIN sont des octets fantômes, donc ce ne sont pas des données. Les octets de données du client vont donc de 1 à 97. Le navigateur E a envoyé au total 97 octets de données.

Question 4

A combien d'octets de charge utile pour TCP correspond la page Web ? Expliquez brièvement votre résultat. Aidez vous du schéma ci-dessous pour justifier votre réponse.

Correction :

On reprend le schéma :



L'avant-dernière ligne avec "Ack = 292" (information entourée en rouge) indique que le client acquitte tous les octets reçus de 1 à 291. Là aussi, il ne faut pas oublier que l'octet 0 correspond au SYN du serveur vers le client, c'est un octet fantôme, et même chose pour le FIN. Les octets de données vont de 1 à 291. Le serveur Web a envoyé 291 octets.

Question 5

Expliquer pourquoi on peut affirmer que la connexion TCP est complètement fermée entre E et le serveur d'après le graphe d'échanges ci-dessus ?

Correction :

Il y a des segments échangés portant l'indicateur FIN dans la trace émis par E et par le serveur. On peut conclure à une fermeture de connexion complète dans les deux sens.

La trace de la trame 4 est découpée en 3 parties pour être complètement visible avec l'entête IP, l'entête TCP, la partie HTTP, et son ensemble en Hexadécimal.

On remarquera que c'est une nouvelle instance de connexion TCP qui est mise en œuvre mais c'est toujours pour la même association ou presque... en effet, le numéro du port client change.

4	0.003290	192.168.0.1	192.168.0.2	HTTP	133 GET / HTTP/1.0
---	----------	-------------	-------------	------	--------------------

Frame 4: 133 bytes on wire (1064 bits), 133 bytes captured (1064 bits)
Ethernet II, Src: SMC_Ether_68:8b:fb (00:e0:29:68:8b:fb), Dst: 3Com_1b:07:fa (00:20:af:1b:07:fa)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.2

- 0100 = Version: 4
- 0101 = Header Length: 20 bytes (5)
- > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
- Total Length: 119
- Identification: 0xc1c7 (49607)
- > Flags: 0x4000, Don't fragment
- Fragment offset: 0
- Time to live: 64
- Protocol: TCP (6)
- Header checksum: 0xf765 [validation disabled]
- [Header checksum status: Unverified]
- Source: 192.168.0.1
- Destination: 192.168.0.2

Transmission Control Protocol, Src Port: 3064, Dst Port: 8000, Seq: 1, Ack: 1, Len: 79

Source Port: 3064

Destination Port: 8000

[Stream index: 0]

[TCP Segment Len: 79]

Sequence number: 1 (relative sequence number)

Sequence number (raw): 3929668180

[Next sequence number: 80 (relative sequence number)]

Acknowledgment number: 1 (relative ack number)

Acknowledgment number (raw): 137897116

0101 = Header Length: 20 bytes (5)

✓ Flags: 0x018 (PSH, ACK)

000. = Reserved: Not set

...0 = Nonce: Not set

.... 0... = Congestion Window Reduced (CWR): Not set

.... .0.. = ECN-Echo: Not set

.... ..0. = Urgent: Not set

.... ...1 = Acknowledgment: Set

.... 1... = Push: Set

....0.. = Reset: Not set

....0. = Syn: Not set

....0 = Fin: Not set

[TCP Flags:AP...]

Window size value: 17520

[Calculated window size: 17520]

[Window size scaling factor: -2 (no window scaling used)]

Checksum: 0x3b9f [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

> [SEQ/ACK analysis]

> [Timestamps]

TCP payload (79 bytes)

```

v Hypertext Transfer Protocol
  v GET / HTTP/1.0\r\n
    > [Expert Info (Chat/Sequence): GET / HTTP/1.0\r\n]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.0
      User-Agent: Wget/1.5.3\r\n
      Host: 192.168.0.2:8000\r\n
      Accept: */*\r\n
      \r\n
      [Full request URI: http://192.168.0.2:8000/]
      [HTTP request 1/1]

```

0000	00 20 af 1b 07 fa 00 e0 29 68 8b fb 08 00 45 00	- - - - -)h---E-
0010	00 77 c1 c7 40 00 40 06 f7 65 c0 a8 00 01 c0 a8	-w--@-@- -e-----
0020	00 02 0b f8 1f 40 ea 39 fa 54 08 38 24 9c 50 18	----@-9 -T-8\$.P-
0030	44 70 3b 9f 00 00 47 45 54 20 2f 20 48 54 54 50	Dp;--GE T / HTTP
0040	2f 31 2e 30 0d 0a 55 73 65 72 2d 41 67 65 6e 74	/1.0--Us er-Agent
0050	3a 20 57 67 65 74 2f 31 2e 35 2e 33 0d 0a 48 6f	: Wget/1 .5.3--Ho
0060	73 74 3a 20 31 39 32 2e 31 36 38 2e 30 2e 32 3a	st: 192. 168.0.2:
0070	38 30 30 30 0d 0a 41 63 63 65 70 74 3a 20 2a 2f	8000--Ac cept: */
0080	2a 0d 0a 0d 0a	*....

Question 6 :

Est-ce que l'URL contenue dans cette requête est correctement formée ? Est-ce que la requête HTTP est correcte ?

Correction :

```

v Hypertext Transfer Protocol
  v GET / HTTP/1.0\r\n
    > [Expert Info (Chat/Sequence): GET / HTTP/1.0\r\n]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.0
      User-Agent: Wget/1.5.3\r\n
      Host: 192.168.0.2:8000\r\n
      Accept: */*\r\n
      \r\n
      [Full request URI: http://192.168.0.2:8000/]
      [HTTP request 1/1]

```

- L'URL (Uniform Resource Locator), qui est une catégorie particulière d'URI (Uniform Resource Identifier), est bien formée : Protocole://machine, ici une adresse IP:no port/chemin d'accès à la ressource. Ici, on prend la page d'accueil puisque le chemin d'accès est vide.
- La requête est-elle formée conformément aux exigences du protocole HTTP ? On retrouve l'opération GET l'accès à la ressource, ici / la racine de l'arbre des pages, donc la page d'accueil, le protocole, ici http, puis /, puis la version du protocole ici 1.0, le tout suivi d'un retour chariot et d'un saut de ligne.
En fait, en http 1.0, le client établit la connexion, envoie une requête, le serveur répond et ferme immédiatement la connexion.

Question 7 : Puzzle sur une communication TCP

Remettre les échanges listés sur chaque ligne dans le bon ordre. La suite correcte représente l'enchaînement des trames qui montre le déroulement d'une connexion TCP entre (192.168.0.1, 3064), navigateur Web, et (192.168.0.2, 8000), serveur Web. On doit retrouver dans l'ordre les 3 phases : ouverture de connexion-transfert de données-fermeture de connexion. On vous donne uniquement l'ordre du premier SYN et la position de la requête GET dans l'enchaînement.

Attention, dans les échanges Wireshark les numéros de séquence sont en relatif (ils démarrent à 0), ça facilite.

1 -	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [SYN] Seq=0 Win=16384 Len=0 MSS=1460
	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [ACK] Seq=296 Ack=81 Win=32120 Len=0
	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [ACK] Seq=1 Ack=1 Win=17520 Len=0
	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [ACK] Seq=80 Ack=296 Win=17520 Len=0
	192.168.0.2	192.168.0.1	TCP	348 8000 → 3064 [PSH, ACK] Seq=1 Ack=80 Win=32120 Len=294 [TCP se
	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [ACK] Seq=1 Ack=80 Win=32120 Len=0
4 -	192.168.0.1	192.168.0.2	HTTP	133 GET / HTTP/1.0
	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [FIN, ACK] Seq=295 Ack=80 Win=32120 Len=0
	192.168.0.2	192.168.0.1	TCP	58 8000 → 3064 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460
	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [FIN, ACK] Seq=80 Ack=296 Win=17520 Len=0

Expliquer brièvement comment vous trouvez votre réponse.

Correction :

On ajoute des lettres à gauche pour chaque ligne pour se repérer, mais on les mettra à droite dans la solution :

1.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [SYN] Seq=0 Win=16384 Len=0 MSS=1460
A.	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [ACK] Seq=296 Ack=81 Win=32120 Len=0
B.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [ACK] Seq=1 Ack=1 Win=17520 Len=0
C.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [ACK] Seq=80 Ack=296 Win=17520 Len=0
D.	192.168.0.2	192.168.0.1	TCP	348 8000 → 3064 [PSH, ACK] Seq=1 Ack=80 Win=32120 Len=294 [TCP se
E.	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [ACK] Seq=1 Ack=80 Win=32120 Len=0
4.	192.168.0.1	192.168.0.2	HTTP	133 GET / HTTP/1.0
F.	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [FIN, ACK] Seq=295 Ack=80 Win=32120 Len=0
G.	192.168.0.2	192.168.0.1	TCP	58 8000 → 3064 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460
H.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [FIN, ACK] Seq=80 Ack=296 Win=17520 Len=0

Le bon enchaînement est :

Ouverture de connexion TCP

1.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [SYN] Seq=0 Win=16384 Len=0 MSS=1460	
2.	192.168.0.2	192.168.0.1	TCP	58 8000 → 3064 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460	(G)
3.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [ACK] Seq=1 Ack=1 Win=17520 Len=0	(B)

Transfert de données (requête http/page Web)

4.	192.168.0.1	192.168.0.2	HTTP	133 GET / HTTP/1.0	
5.	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [ACK] Seq=1 Ack=80 Win=32120 Len=0	(E)
6.	192.168.0.2	192.168.0.1	TCP	348 8000 → 3064 [PSH, ACK] Seq=1 Ack=80 Win=32120 Len=294 [TCP se	(D)

Fermeture de connexion TCP

7.	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [FIN, ACK] Seq=295 Ack=80 Win=32120 Len=0	(F)
8.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [ACK] Seq=80 Ack=296 Win=17520 Len=0	(C)
9.	192.168.0.1	192.168.0.2	TCP	60 3064 → 8000 [FIN, ACK] Seq=80 Ack=296 Win=17520 Len=0	(H)
10.	192.168.0.2	192.168.0.1	TCP	54 8000 → 3064 [ACK] Seq=296 Ack=81 Win=32120 Len=0	(A)

- Le 1^{er} segment est une ouverture de connexion client vers serveur, car il contient un SYN. Le 2^{ème} segment doit être un SYN,ACK du serveur vers le client. Il n'y en a qu'un seul, le G. Pour les 2 premiers segments il apparaît MSS (Maximum Segment Size), c'est une information qui s'échange entre le client et le serveur pendant l'ouverture de connexion TCP.
- Le 3^{ème} segment doit être un ACK du client vers le serveur pour finaliser le three-way handshake de l'ouverture de connexion TCP. Il y en a plusieurs. Il doit porter un ACK avec le numéro d'acquittement Ack=1 car il acquitte le SYN émis par le serveur. Il n'y a qu'un seul segment avec cette caractéristique, le B.

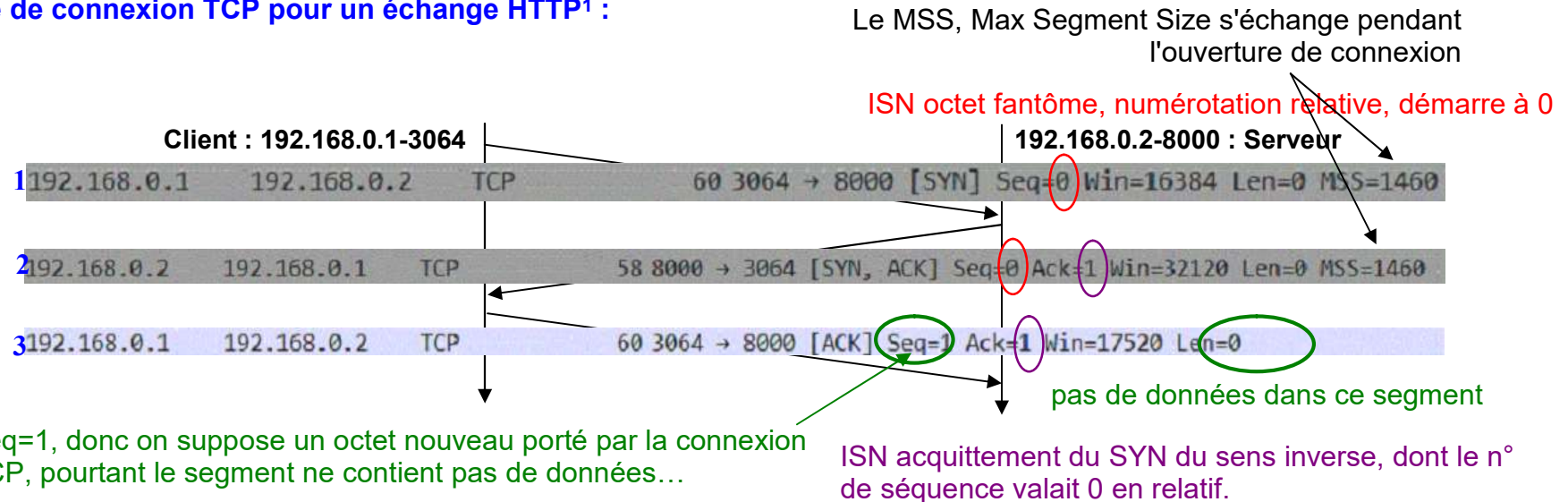
A partir d'ici, l'ouverture de connexion TCP est effectuée on bascule dans la phase transfert de données.

- Le 4^{ème} segment est fourni dans les données du problème, et indique une requête http GET du client vers le serveur.
- Le 5^{ème} segment doit être un ACK de la requête GET ou l'envoi de la page demandée.
 - Si c'était l'envoi de la page on aurait probablement un PSH, ce n'est pas le cas
 - On suppose donc que c'est un ACK sans données du serveur vers le client. Dans ce cas, il acquitte les octets reçus en sens inverse d'un segment TCP précédent. On a un segment sans données (Len=0) et avec Ack=80.
La requête GET est dans une trame de 133 octets. Il faut enlever 14+20+20 octets qui sont les entêtes Ethernet+IP+TCP. On a donc bien 79 octets de données (cf question 2) dans le segment qui contient le GET. Cela veut dire que le serveur attend l'octet 80 du client et qu'il a bien reçu sans erreur tous les octets qui précèdent, donc les 79 octets de la requête GET.
Il n'y a que le E qui répond à cette caractéristique.
- Le 6^{ème} segment correspond à l'envoi du contenu de la page web demandée. Il doit porter aussi l'acquittement en sens inverse, soit Ack = 80. Il n'y a que le D qui répond à ces caractéristiques. On observe qu'il porte aussi un PSH pour délivrer au plus vite les données au navigateur.
- L'échange de données est terminé, il faut fermer la connexion TCP :
 - Le serveur peut initier la fermeture ou le client.
 - Le FIN est un octet fantôme donc compté, il faut donc un Ack qui ait pris +1 par rapport au volume des données, et qui du serveur ou du client le fait le premier ? On n'a pas d'éléments probants pour décider sur cette base d'hypothèses.
 - On a vu en ED que le serveur Web fermait la connexion en premier. Si on fouille dans la partie en hexadécimal, on voit que c'est le protocole HTTP1.0 qui est utilisé. Comme on l'a vu, c'est le client qui initie la connexion, mais c'est le serveur qui la ferme en HTTP1.0. Par conséquent, on prend comme 7^{ème} segment, F.
- Du coup, l'ACK de la fermeture de la connexion vient du client et notre 8^{ème} segment est le C.
- La fermeture de connexion côté client avec un FIN donne le 9^{ème} segment et c'est le H. Et son acquittement par le serveur donne le 10^{ème} segment soit le A.

Finalement, on peut représenter cela sous la forme d'un échange entre le client et le serveur avec des axes temporels.

ISN = Initial Sequence Number, numéro de séquence du premier octet de la connexion qui n'est autre que l'octet fantôme du SYN.

Ouverture de connexion TCP pour un échange HTTP¹ :

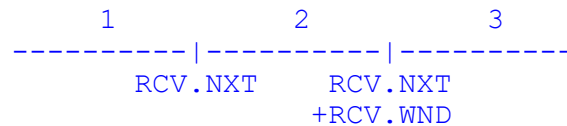


Pour savoir pourquoi Seq=1, on a la réponse dans la section 3.4, RFC 793, figure 7. On le constate sur la figure, mais ça n'explique pas. La suite de la correction sur ce point très particulier, c'est pour ceux qui veulent creuser les choses en réseaux. Pour aller plus en détails et comprendre, il faut creuser dans divers points de la RFC793 qui est en anglais dont les sections voisines de 3.4. La RFC se sert des variables suivantes (il y en a d'autres, mais pour ce qui nous intéresse elles ne sont pas utiles). Ces variables permettent de compter les octets envoyés/reçus, acquittés/non acquittés, prêts à être envoyés/recevables à travers les numéros de séquence/d'acquittement. Pour répondre à notre interrogation, on s'intéresse plutôt à ce qui est lié à la réception :

Receive Sequence Variables (Figure 5 de la RFC793)

RCV.NXT - receive next
RCV.WND - receive window

¹ Les informations données ci-dessous, sont les plus complètes possibles, et vont dans un détail que je ne demandais pas bien sûr. Elles sont données pour compléter votre compréhension.

Receive Sequence Space

- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers allowed for new reception
- 3 - future sequence numbers which are not yet allowed

Current Segment Variables

SEG.ACK = acknowledgment from the receiving TCP (next sequence number expected by the receiving TCP)

SEG.SEQ = first sequence number of a segment

SEG.LEN = the number of octets occupied by the data in the segment (counting SYN and FIN)

SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

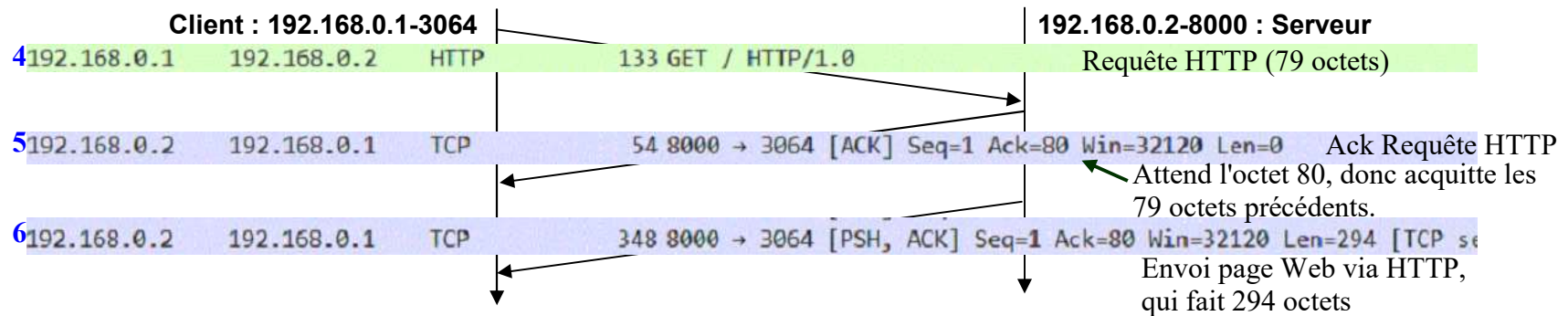
Due to zero length segments, we have for the acceptability of an incoming segment, the Segment Receive Test :
 Length (0) & Window (>0): $RCV.NXT \leq SEG.SEQ < RCV.NXT + RCV.WND$

Cette règle fait que la première fois, pour l'émetteur, quand il n'y a pas de données à transmettre dans le segment, le numéro de séquence progresse de 1. Après, ça ne bouge plus tant qu'il n'y a pas à nouveau des données à émettre. Si on ne fait pas ça, le segment n'est pas accepté par le récepteur... si j'ai bien tout compris ;-).

On applique ce test qui correspond à notre cas puisque dans le segment 3 il n'y a pas de données mais un acquittement :

Nous avons Serveur.RCV.WND = 32120 et Serveur.RCV.NXT = 1 (puisque l'Ack = 1 et Win vaut 32120 dans le segment de l'envoi en 2). Dans le segment de l'envoi 3 on a SEG3.SEQ = 1. Donc quand le Serveur reçoit le segment de l'échange 3, il l'accepte. Ca n'aurait pas été le cas si SEG3.SEQ = 0 car la condition $RCV.NXT \leq SEG3.SEQ$ n'aurait pas été vérifiée.

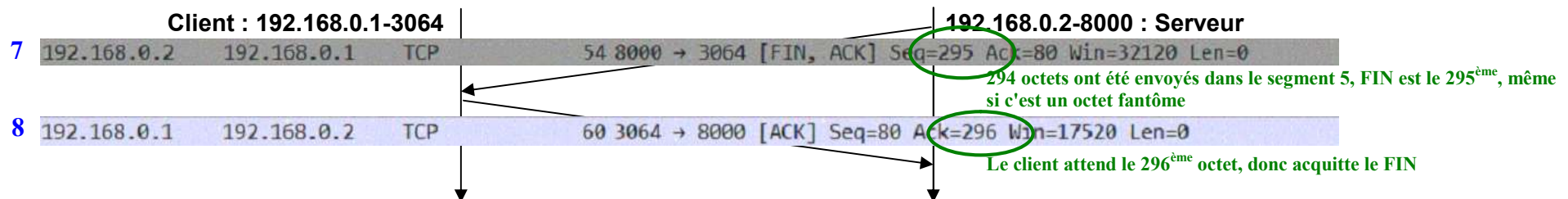
Transferts de données TCP qui portent la requête-réponse HTTP :



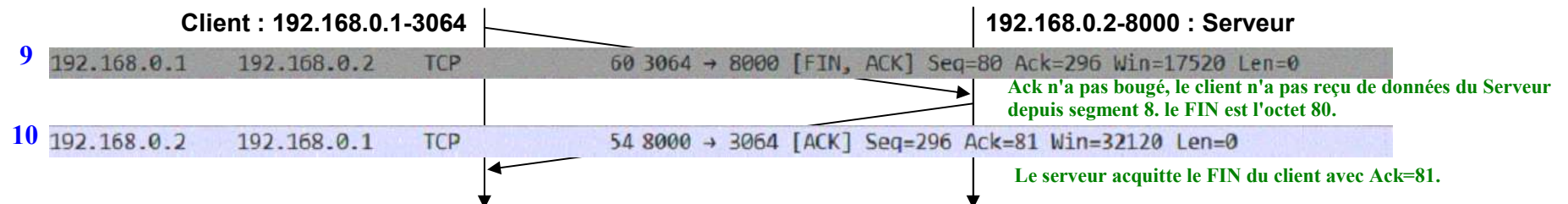
Fermeture de connexion TCP après échange HTTP :

Comme s'est dit plus haut dans le texte, avec HTTP 1.0, la connexion TCP n'est ouverte que pour la requête-réponse, c'est le serveur qui initie la fermeture de connexion TCP dès la page demandée envoyée.

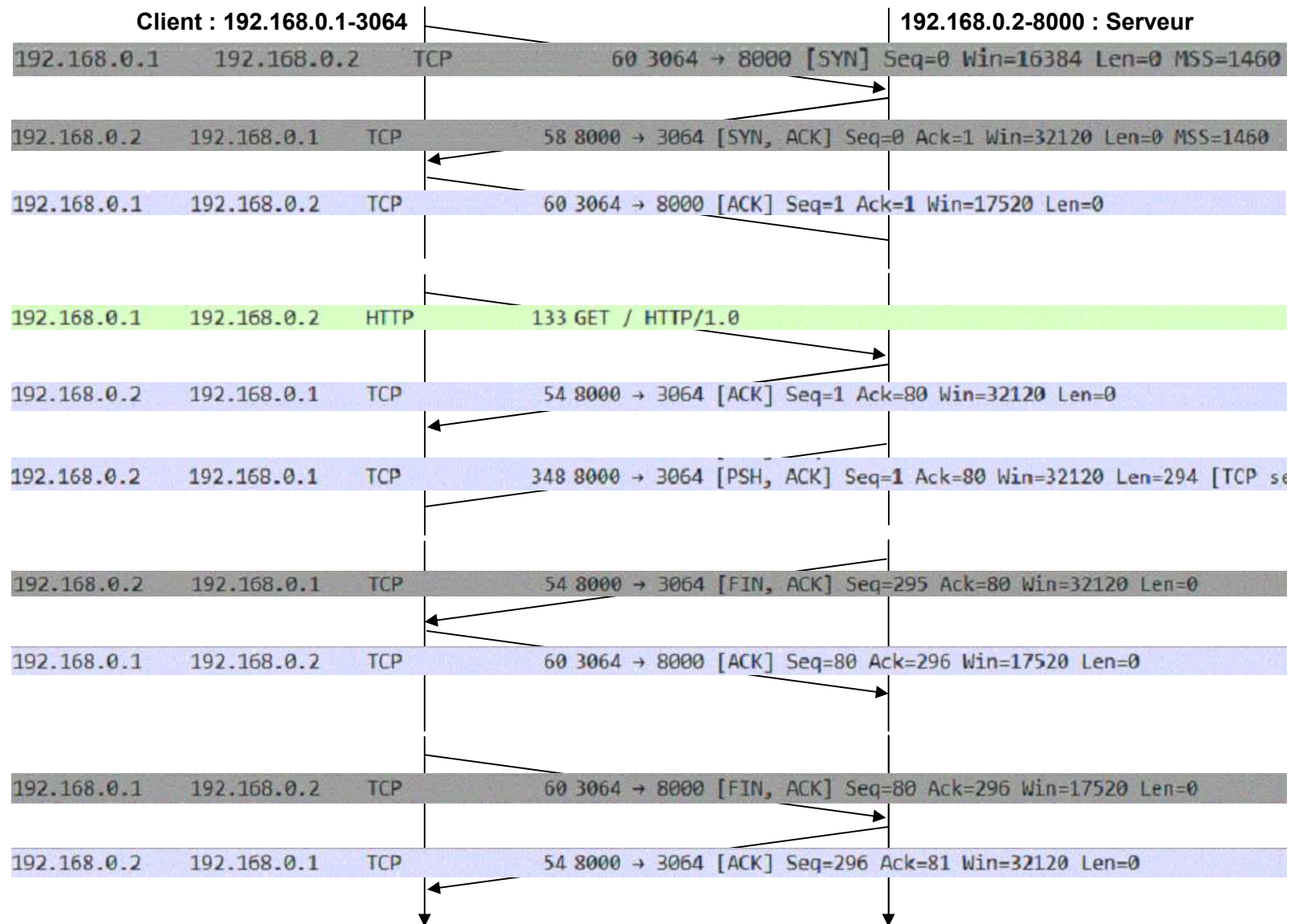
Une fermeture TCP correspond à 2 demi-fermetures, une pour chaque sens de la connexion TCP qui rappelons le est full-duplex. En premier la fermeture du sens Serveur vers Client :



En suite la fermeture du sens Client vers Serveur :



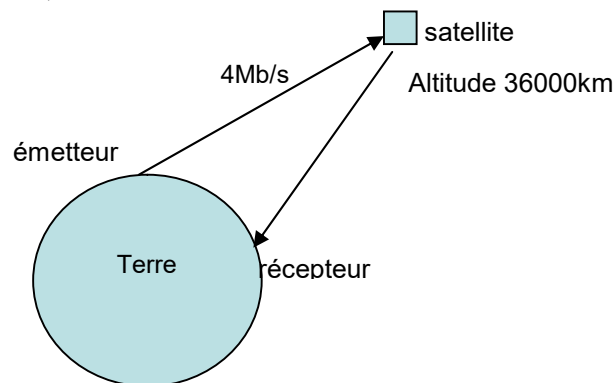
Vue d'ensemble :



Exercice 5 : Fenêtre d'Anticipation et communication TCP satellite. A faire soi-même, facile.

Soit un satellite en orbite géo-stationnaire. Il est situé à 36000 km de la terre. Les messages font 1500 octets et le débit nominal de la liaison satellite est de 4 Mb/s ($4 \cdot 10^6 \text{ b/s}$)² en montée (upload) et en descente (download)³.

On rappelle que le délai de propagation d'une onde électromagnétique dans l'espace est voisin de la vitesse de la lumière, soit 300000 km/s.



Les 1500 octets comptent pour : l'entête IP (20 octets), l'entête TCP (20 octets), et les données (1460 octets).

Pour construire une trame, on ajoute aux 1500 octets les informations de gestion : l'entête et le CRC de la trame qui comptent pour 20 octets (Longueur de la trame, Type de la trame, Adresse destination de liaison, Type de la charge acheminée, CRC).

On néglige dans les calculs la prise en compte du préambule.

1. Quelle est la taille d'une trame envoyée sur la liaison satellite ? Expliquez très brièvement votre calcul.

Correction :

La trame envoyée a une taille de :

1500 octets (données dont entêtes IP et TCP), 20 octets (information de gestion) soit 1520 octets partent sur le support hertzien en transmission, soit 12160 bits.

2. Quel est le délai de propagation de la terre au satellite en millisecondes (ms) ? Expliquez très brièvement votre calcul, il compte pour le calcul de la latence entre émetteur et récepteur.

Correction :

Pour aller de la terre au satellite on doit parcourir 36000 km, l'onde voyage à la vitesse de la lumière soit 300 000 km/s, le délai de propagation de l'onde de la trame pour aller de la terre au satellite est $36000/300000$ soit $36/300$, soit $12/100$, soit 0,12 s ou encore 120 ms

3. Quel est le temps, en ms, nécessaire pour émettre une trame ?

² Attention, on utilise de façon systématique le Mo qui exprime une puissance 1024 , alors qu'officiellement on devrait utiliser dans cette circonstance 1024^2 . Pour plus d'information consulter : https://fr.wikipedia.org/wiki/Pr%C3%A9fixe_binaire, merci à Loïc Cartier, auditeur d'UTC505 1^{er} semestre 2020-2021 pour ce lien. Dans l'exercice il est bien question de mégabit et pas de Mibi.

³ En réalité, le débit de montée est largement inférieur au débit de descente. Mais on fait cette hypothèse pour simplifier l'exercice.

Correction :

On émet une trame à la vitesse de 4Mb/s, la trame fait une longueur de 1520 octets soit 12160 bits comme vu plus haut. Le temps d'émission au sortir de la carte de communication avec le satellite est donc de $12160/4000000$ s, soit $3040 \cdot 10^{-6}$ s soit 0,003040s soit 3,04 ms.

4. Combien de temps met la trame pour arriver complètement au destinataire en passant par le satellite ? A peine arrivée sur le satellite, elle est renvoyée vers le récepteur sur la terre. Le satellite, malgré toute l'intelligence qu'il peut contenir n'est considéré qu'un élément de la couche physique, c'est si on veut résumer un répéteur. Justifiez très brièvement votre calcul.

Correction :

La trame pour aller jusqu'au satellite met le temps d'émission de la trame + le délai pour aller de la terre côté émetteur au satellite + délai pour aller du satellite à la terre côté récepteur. Aucun délai de traversée des circuits électroniques du satellite n'est spécifié, dans la réalité il y en a. On a donc : $3,04 + 2 \cdot 120 = 243,04$ ms.

On utilise TCP pour envoyer les données. On suppose qu'on envoie un segment puis le récepteur l'acquitte. Tant que l'émetteur n'a pas reçu l'acquittement, il n'envoie pas de nouveau segment. L'acquittement est un segment qui ne contient aucune donnée. On suppose qu'il n'y a pas de perte de segment lors des transmissions, ce qui est une hypothèse optimiste.

5. Quelle est la taille d'une trame qui contient un segment d'acquittement ?

Correction :

Le segment d'acquittement positionne le marqueur ACK dans l'entête TCP, donc il ne prend aucun octet supplémentaire. La trame d'ACK émise va contenir 20 octets d'entête IP, 20 octets d'entête TCP, tous les champs de la trame 20 octets. On a donc une trame de 60 octets pour acquitter les données.

6. Quel est le temps, en ms, nécessaire pour transmettre ce segment d'acquittement vers la base source ?

Correction :

Suivant le même raisonnement que pour l'émission :

- on compte le temps d'émission de la trame qui contient l'acquittement $60 \cdot 8/4000000$ s, $480/4000000$ s, soit $120 \cdot 10^{-6}$ s, soit 0,000120s soit 0,12 ms

Pour le temps de transmission complet de la trame contenant l'acquit

- on compte le délai terre-satellite-terre, soit 240ms

Le délai total de transmission de l'acquittement est $240 + 0,12$ soit 240,12ms

7. Combien de temps se passe-t-il entre le début d'émission du segment de données et la réception de son acquittement par la source ? Justifiez très brièvement votre calcul.

Correction :

Le délai A/R entre l'émetteur et le récepteur pour avoir la séquence "émission de la donnée"/"réception de l'acquittement" prend $243,04 + 240,12$ ms, soit 483,16 ms au total.

8. Quel est le débit utile avec cette solution qui consiste à envoyer segment par segment, et à ne pas envoyer le segment suivant tant que l'acquittement du précédent n'est pas arrivé à l'émetteur.

On rappelle que le débit utile se calcule en faisant le rapport taille d'un message en bits sur temps pour le transmettre en secondes.

Correction :

Le débit utile correspond au débit applicatif soit 1460 octets divisé par 483,16 ms. Il faut convertir ces quantités en bits et en secondes pour avoir un débit exprimé en bits par seconde. On a alors $1460 \times 8 / 0,48316 \text{ b/s}$ soit 24174 b/s environ. Exprimé autrement $24,174 \text{ Kb/s}$

9. Quelle est la valeur de l'efficacité de cette liaison ? Qu'en concluez vous ?

On rappelle que l'efficacité d'une ligne est le rapport débit utile sur débit nominal.

Correction :

L'efficacité c'est le rapport débit utile (applicatif) sur débit nominal du lien.

L'efficacité est donc $24,174 \text{ Kb/s}$ que divise 4000 Kb/s (4 Mb/s). On obtient la valeur $0,0060435$ soit environ $0,6\%$. Ce protocole est plus que très inefficace.

On propose une autre solution. On met en place une fenêtre d'anticipation à l'émission dans TCP. Ce qui veut dire qu'on envoie plusieurs segments avant de recevoir le premier acquittement.

10. Quelle pourrait la valeur K de cette fenêtre d'anticipation, exprimée en nombre de messages, qui maximise l'efficacité de la communication satellitaire pour qu'elle s'approche des 100%.

Correction :

- 1^{er} raisonnement : On peut raisonner directement avec l'efficacité pour atteindre le 100%. On cherche x tel que $x \times 24,174 / 4000 = 1$ (100%).

$x = 4000 / 24,174 = 165,467$, d'après ce raisonnement, il faudrait envoyer successivement 165 messages de données avant de recevoir le premier acquittement.

C'est un calcul très approximatif, il faudrait le valider avec des trames pour s'assurer que 165 est la juste valeur.

- 2^{ème} raisonnement : On peut raisonner par rapport au débit applicatif exprimé au niveau du medium (trame) et par rapport au débit nominal. On cherche x tel que $x \times 1520 \times 8 / 0,48316 = 4000000$. C'est une autre formulation de l'objectif 100% d'efficacité.

$x = 4000000 \times 0,48316 / 1520 \times 8 = 4 \times 48316 / 152 \times 8 = 48316 / 304 = 158,9$ soit environ 159 trames.

- Pourquoi les deux chiffres diffèrent ? En fait, on ne fait pas exactement le même calcul. Dans le premier raisonnement, on se fonde sur le débit applicatif, et on essaie de le maximiser par rapport au débit nominal... mais ça ne tient pas compte des octets de services ajoutés par la trame, par IP et par TCP, ce qu'on appelle l'overhead en anglais. Le deuxième raisonnement produit un résultat plus juste.

11. TCP gère une fenêtre d'anticipation, mais exprimée en octets. Quelle valeur proposeriez vous pour configurer la fenêtre d'émission de TCP à partir de la réponse à la question précédente.

Attention, la taille de la fenêtre doit s'exprimer sous la forme d'un chiffre qui est une puissance de 2, on la comptera en Kilo octets (1024 octets).

Correction :

Un message fait 1460 octets de données. La fenêtre d'anticipation serait de 165×1460 , soit 240900 octets avec le premier raisonnement. Avec le 2^{ème} raisonnement elle serait de 159×1460 , soit 232 140 octets.

- $240900 = 235 \times 1024 + 260$ octets, soit environ 235 Ko => première puissance de 2 supérieure 256 Ko, première puissance de 2 inférieure 128Ko
- $232140 = 226 \times 1024 + 716$ octets, soit environ 227 Ko => on a le même encadrement que ci-dessus

12. La taille d'une fenêtre TCP est au maximum de 65536 octets (numérotés de 0 à 65535) car c'est un champ sur 16 bits (et au minimum de 2 octets). Est-ce suffisant ? Quel mécanisme imagineriez vous pour augmenter la taille de la fenêtre sans modifier l'entête d'un segment TCP.

Correction :

65535 octets, c'est inférieur aux nombres qu'on vient de trouver dans la question 11.

Il faudrait inventer un mécanisme d'agrandissement de la taille maximum de la fenêtre. En fait, cela existe dans TCP, cette option s'appelle window scale.

Elle est échangée lors de l'ouverture de connexion de part et d'autre entre émetteur et récepteur. C'est un facteur multiplicatif de la taille de la fenêtre qui permet de l'augmenter jusqu'à 1Go.

Estimons un facteur d'agrandissement. De la question 11 on a : 256 Ko > fenêtre nécessaire > 128 Ko. La fenêtre maximum paramétrable est de 64 Ko. Si on peut utiliser un facteur multiplicatif on a les options suivantes :

- Si on multiplie par 4, on a une fenêtre possible de 256Ko (262 144 octets).
- Si on multiplie par 2, on a une fenêtre possible de 128Ko (131 072 octets).

A 128Ko, le tuyau de communication satellitaire ne sera pas rempli au maximum, alors qu'avec 256Ko ça le sera, on pourra même bufferiser plus que nécessaire.

Le facteur d'agrandissement est donc 4, soit 2^2 .

Exercice 6 : Etude de Trace TCP portant un échange http (à faire en autonomie)

On s'intéresse maintenant à TCP qui porte des échanges HTTP. La trace ci-après est relevée au niveau de l'outil Wireshark :

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	DellEsgP_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

Attention, les numéros de séquence affichés sont relatifs, le véritable numéro de séquence dans le segment est transformé pour numéroté les octets d'une connexion à partir de 0. Cette règle est appliquée dans les deux sens.

Pour vous aider dans votre analyse des échanges, vous pouvez vous appuyer sur la trace WireShark correspondante obtenue avec la commande "Follow TCP stream" (en **violet** la requête client, et en **turquoise** la réponse du serveur) :

```
GET / HTTP/1.1
Host: lmil7.cnam.fr
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Connection: keep-alive
HTTP/1.1 200 OK
Date: Wed, 27 Mar 2013 16:12:20 GMT
Server: Apache/2.0.53 (Linux/SUSE)
Last-Modified: Tue, 20 Sep 2005 16:16:03 GMT
ETag: "1734016-b7c-401366056b6c0"
Accept-Ranges: bytes
Content-Length: 2940
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
  <META NAME="GENERATOR" CONTENT="Mozilla/4.03 [fr] (Win95; I) [Netscape]">
  <TITLE>Laboratoire de Micro-Informatique du CNAM-PARIS</TITLE>
</HEAD>
<BODY TEXT="#330000" BGCOLOR="#FFFFBB" LINK="#0000EE" VLINK="#FF0000" ALINK="#FF0000">
&nbsp;
... une partie de cette page a été retirée de la trace car elle ne présentait aucun intérêt pour répondre...
<HR>Pour vos commentaires et questions : <B><A HREF="mailto:jb@cnam.fr">Web
Master</A></B>
<BR>Dernière modification : Monday, November 03 1997 10:58:19
</BODY>
</HTML>
GET /logo_cnam.gif HTTP/1.1
Host: lmil7.cnam.fr
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Connection: keep-alive
Referer: http://lmil7.cnam.fr/
HTTP/1.1 200 OK
Date: Wed, 27 Mar 2013 16:12:20 GMT
Server: Apache/2.0.53 (Linux/SUSE)
Last-Modified: Tue, 04 Nov 1997 08:46:02 GMT
```



```
ETag: "1734010-1b95-31f1cb9824680"  
Accept-Ranges: bytes  
Content-Length: 7061  
Keep-Alive: timeout=15, max=99  
Connection: Keep-Alive  
Content-Type: image/gif  
GIF87a.....}}>{{{yyywwwuuu  
...la suite de la trace a été enlevée elle ne présente pas d'intérêt pour l'exercice
```

Question 1

Question 1.1

Les lignes 3, 4, 5 matérialisent l'ouverture de connexion, expliquer :

- pourquoi ces lignes correspondent à l'ouverture de connexion,
- qui fait l'ouverture de connexion passive et qui fait donc l'ouverture de connexion active
- pourquoi, selon vous, le MSS (Maximum Segment Size) vaut 1460,

Correction :

- L'ouverture de connexion TCP se caractérise dans la très grande majorité des cas par un échange de messages en 3 phases (three-way handshake) qui contiennent respectivement : SYN (client vers serveur), SYN-ACK (serveur vers client), ACK (client vers serveur). Dans la trace, lignes 3, 4, 5 on reconnaît ce pattern de communication.
- Compte tenu des numéros de port, on sait que le serveur web est associé généralement au port 80, ici, c'est donc 163.173.228.17 (lmi17.cnam.fr). Le client va être le navigateur, il est sur 163.173.231.107 (ulyse.cnam.fr) et qui utilise le port 50074. Celui qui envoie le SYN est 163.173.231.107, 50074, c'est donc lui qui fait l'ouverture de connexion active. Celui qui répond avec SYN-ACK est 163.173.228.17, 80 a fait l'ouverture de connexion passive.
- Le MTU (Maximum Transfer Unit) de la trame Ethernet est de 1500 octets maximum (taille max des données dans la trame). Ce MTU est occupé par 20 octets d'entête IP, et 20 octets d'entête TCP. Donc 40 octets sont occupés par des données de service. Il ne reste donc pour des données TCP que 1460 octets de disponible, ce qui définit la valeur de MSS (Maximum Segment Size). Attention se calcul ne se fait que si on n'a pas utilisation d'options dans les entêtes IP et TCP.

Question 1.2

Les lignes 24, 25, 26, 27 matérialisent la fermeture de connexion, expliquer :

- pourquoi ces lignes correspondent à la fermeture de connexion,
- combien de messages pour fermer la connexion des deux côtés,
- finalement pour obtenir la page qui s'affiche sur le navigateur, combien de connexions ont été ouvertes



Correction :

- Une connexion dans TCP est full duplex. Le protocole TCP considère qu'il faut fermer chaque sens de la connexion pour fermer la connexion complète. La fermeture d'un sens se fait par l'échange d'un FIN puis d'un ACK. On observe un FIN (serveur vers client) puis ACK (client vers serveur). Puis on observe un FIN (client vers serveur) puis un ACK (serveur vers client).
- D'après la description ci-dessus, la connexion a été complètement fermée à l'aide de 4 messages.
- Il y a eu une ouverture de connexion, puis échange de données, puis fermeture de connexion. Donc une seule connexion a été suffisante pour transférer complètement le contenu de la page du serveur vers le navigateur.

Question 1.3

Combien de données ont été échangées :

- du client vers le serveur
- du serveur vers le client

Correction :

La notion de données doit se comprendre comme volume de données du point de vue de la couche TCP échangées entre le navigateur et le serveur. Ce peut être des données "contenu de page" mais aussi des données protocolaires spécifiques à http.

Quels sont les éléments qui peuvent nous aider ? Le flux de données compté en nombre d'octets par le protocole TCP côté client et côté serveur peut nous aider. Il faut par contre décompter l'octet associé au SYN et l'octet associé au FIN.

Dans la ligne 26 le segment que le client envoie un numéro de prochain octet à recevoir qui vaut 10601 (ACK). Il a donc reçu les octets 0 à 10600. Soit 10601 octets. Mais on retire l'octet SYN (de l'ouverture de connexion) et l'octet FIN (de la fermeture de connexion). Cela fait donc **10599** octets dans le sens serveur (lmi17.cnam.fr/163.173.228.17,80) vers client (ulyse.cnam.fr, 163.173.231.107,50074).

Dans le sens inverse, avec la ligne 27, on obtient par le même raisonnement le volume de données dans le sens client vers serveur est **995** octets (l'ACK envoyé par le serveur web porte la valeur 997, le serveur attend le 997^{ème} octet, donc l'envoi des octets de 0 à 996 est acquitté, soit 997 octets au total moins FIN et SYN).

En fait, le reste de la trace n'apporte aucune information supplémentaire.

Question 1.4

Dans le segment ci-dessous, trame 3 de la trace, retrouvez le numéro de séquence réel du SYN dans la trame en octets au format hexadécimal.

```
Ethernet II, Src: Inventec 9a:4d:aa (00:26:6c:9a:4d:aa), Dst: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)
  Destination: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)
    Address: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)
      .... ..0. .... .. = LG bit: Globally unique address (factory default)
```



```
.... 0 .... = IG bit: Individual address (unicast)
Source: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)
Address: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)
.... 0 .... = LG bit: Globally unique address (factory default)
.... 0 .... = IG bit: Individual address (unicast)
Type: IP (0x0800)
Internet Protocol Version 4, Src: 163.173.231.107 (163.173.231.107), Dst: 163.173.228.17 (163.173.228.17)
Version: 4
Header length: 20 bytes
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
Total Length: 52
Identification: 0x4d92 (19858)
Flags: 0x02 (Don't Fragment)
    0... .... = Reserved bit: Not set
    .1.. .... = Don't fragment: Set
    ..0. .... = More fragments: Not set
Fragment offset: 0
Time to live: 128
Protocol: TCP (6)
Header checksum: 0x9a59 [correct]
    [Good: True]
    [Bad: False]
Source: 163.173.231.107 (163.173.231.107)
Destination: 163.173.228.17 (163.173.228.17)
Transmission Control Protocol, Src Port: 50074 (50074), Dst Port: http (80), Seq: 0, Len: 0
Source port: 50074 (50074)
Destination port: http (80)
[Stream index: 0]
Sequence number: 0 (relative sequence number)
Header length: 32 bytes
Flags: 0x002 (SYN)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    .... 0... = Congestion Window Reduced (CWR): Not set
    .... .0.. = ECN-Echo: Not set
    .... ..0. = Urgent: Not set
    .... ...0 = Acknowledgment: Not set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
Window size value: 8192
```

```

[Calculated window size: 8192]
Checksum: 0xcfa1 [validation disabled]
Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP),
SACK permitted
  Maximum segment size: 1460 bytes
    Kind: MSS size (2)
    Length: 4
    MSS Value: 1460
  No-Operation (NOP)
    Type: 1
    0... .... = Copy on fragmentation: No
    .00. .... = Class: Control (0)
    ...0 0001 = Number: No-Operation (NOP) (1)
  Window scale: 2 (multiply by 4)
    Kind: Window Scale (3)
    Length: 3
    Shift count: 2
    [Multiplier: 4]
  No-Operation (NOP)
    Type: 1
    0... .... = Copy on fragmentation: No
    .00. .... = Class: Control (0)
    ...0 0001 = Number: No-Operation (NOP) (1)
  No-Operation (NOP)
    Type: 1
    0... .... = Copy on fragmentation: No
    .00. .... = Class: Control (0)
    ...0 0001 = Number: No-Operation (NOP) (1)
  TCP SACK Permitted Option: True
    Kind: SACK Permission (4)
    Length: 2

```

Les entêtes ont été colorées (MAC, IP, TCP).

0000	00 0f 1f f9 ad 5a 00 26 6c 9a 4d aa 08 00 45 00Z.&l.M...E.
0016	00 34 4d 92 40 00 80 06 9a 59 a3 ad e7 6b a3 ad	.4M.@....Y...k..
0032	e4 11 c3 9a 00 50 3c ac 6c 05 00 00 00 00 80 02P<.l.....
0048	20 00 cf a1 00 00 02 04 05 b4 01 03 03 02 01 01
0064	04 02	..

On rappelle la grille d'analyse d'une entête TCP :

identifiant émetteur		identifiant récepteur									
no de séquence du premier octet émis contenu dans ce segment											
no d'acquittement : no de séquence du prochain octet à recevoir par celui qui envoie ce segment											
bits indicateurs											
longueur entête + options		Ré sé rvé	E C N R	C W G	U R K	A C H	P S T	R S I	F I N	taille de la fenêtre	
contrôle d'erreur sur l'entête						fin des données urgentes placées en début des données utilisateur dans le segment					
options s'il y en a											
données s'il y en a											

20octets

20octets

Correction :

Remarquons tout d'abord que ce segment est un segment d'ouverture de connexion (présence de l'indicateur SYN dans l'entête, "02" en hexadécimal soit "0000 0010" en binaire). Donc il y a un numéro de séquence mais pas de numéro d'acquittement, on a bien "Acknowledgment: Not set" dans l'entête TCP et 00°00°00°00 dans le champ correspondant "n° d'ack" de l'entête du segment TCP de la trace. D'après le format d'un segment, on trouve que le numéro de séquence est 3c°ac°6c°05. Il est bien lisible dans le bon sens puisqu'on est en stockage bigendian (octet le plus significatif dans l'octet de poids faible). On remarque que la trace affichée par WireShark affiche un nombre différent, en relatif plutôt qu'en absolu. C'est effectivement plus facile de faire des vérifications sur le volume de données transférées avec cette approche.

Question 2

On s'intéresse à l'état des connexions sur une des deux machines. On lance la commande netstat -a dont une partie du résultat est affiché ci-dessous. On obtient un ensemble de connexions actives.

Proto	Adresse locale	Adresse distante	État
TCP	163.173.231.107:139	netinfo67:0	LISTENING
TCP	163.173.231.107:2869	wi225:59696	TIME_WAIT
TCP	163.173.231.107:2869	wi225:59699	ESTABLISHED
TCP	163.173.231.107:2869	wi225:59704	ESTABLISHED
TCP	163.173.231.107:2869	bering:32927	TIME_WAIT
TCP	163.173.231.107:49997	wpad:http	CLOSE_WAIT
TCP	163.173.231.107:50074	lmi17:http	ESTABLISHED
TCP	169.254.8.118:139	netinfo67:0	LISTENING

L'automate TCP dispose de 3 types d'états : les états liés à l'ouverture de connexion, les états liés au transfert de données, les états liés à la fermeture de connexion.

Question 2.1

A quels types d'états correspondent les états TIME_WAIT, CLOSE_WAIT, ESTABLISHED, LISTENING, suivants relevés dans les résultats de la commande netstat.

Correction :

D'après l'automate TCP vu en cours :

- LISTENING correspond à un état d'ouverture de connexion, plutôt connexion passive côté serveur.
- ESTABLISHED correspond à un état de transfert de données (la connexion ayant été établie)
- TIME_WAIT, CLOSE_WAIT correspondent à un état fermeture de connexion

Question 2.2

Dans quel état est la connexion TCP qui achemine le trafic http que nous étudions d'après les résultats de netstat.

Correction :

La ligne dans le résultat de la commande netstat est la suivante, on y reconnaît les adresses IP/nom du client et du serveur et les numéros de port :

TCP	163.173.231.107:50074	lmi17:http	ESTABLISHED
-----	-----------------------	------------	-------------

La connexion est en mode transfert.

Question 2.3

Quel appel système de l'API sockets déclenche l'ouverture de connexion active sur le client.

Correction :

C'est l'appel système `connect()`.

Question 3

Fragmentation IP ou fragmentation TCP ?

On s'intéresse à la fragmentation de segments TCP. Ligne 8 et 9 on voit apparaître dans la ligne :

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]

La trace détaillée correspondante pour ces deux lignes donne :

No.	Time	Source	Destination	Protocol	Length	Info
8	0.003607000	163.173.228.17	163.173.231.107	TCP	353	[TCP segment of a reassembled PDU]

Frame 8: 353 bytes on wire (2824 bits), 353 bytes captured (2824 bits) on interface 0

Interface id: 0

Frame Number: 8

Frame Length: 353 bytes (2824 bits)

Capture Length: 353 bytes (2824 bits)

[Frame is marked: True]

[Frame is ignored: False]

[Protocols in frame: eth:ip:tcp]

Ethernet II, Src: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a), Dst: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)

Destination: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)

Address: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)

.... ..0. = LG bit: Globally unique address (factory default)

.... ..0. = IG bit: Individual address (unicast)

Source: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)

Address: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)

.... ..0. = LG bit: Globally unique address (factory default)

.... ..0. = IG bit: Individual address (unicast)

Type: IP (0x0800)

Internet Protocol Version 4, Src: 163.173.228.17 (163.173.228.17), Dst: 163.173.231.107 (163.173.231.107)



```
Version: 4
Header length: 20 bytes
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
Total Length: 339
Identification: 0x72eb (29419)
Flags: 0x02 (Don't Fragment)
    0... .... = Reserved bit: Not set
    .1... .... = Don't fragment: Set
    ..0. .... = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (6)
Header checksum: 0xb3e1 [correct]
    [Good: True]
    [Bad: False]
Source: 163.173.228.17 (163.173.228.17)
Destination: 163.173.231.107 (163.173.231.107)
Transmission Control Protocol, Src Port: http (80), Dst Port: 50074 (50074), Seq: 1, Ack: 491, Len: 299
Source port: http (80)
Destination port: 50074 (50074)
[Stream index: 0]
Sequence number: 1      (relative sequence number)
[Next sequence number: 300      (relative sequence number)]
Acknowledgment number: 491      (relative ack number)
Header length: 20 bytes
Flags: 0x018 (PSH, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 1... = Push: Set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
Window size value: 1728
```

```
[Calculated window size: 6912]
[Window size scaling factor: 4]
Checksum: 0xe4e3 [validation disabled]
[SEQ/ACK analysis]
  [Bytes in flight: 299]
TCP segment data (299 bytes)
```

No.	Time	Source	Destination	Protocol	Length	Info
9	0.003885000	163.173.228.17	163.173.231.107	TCP	1514	[TCP segment of a reassembled PDU]

Frame 9: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0

```
Interface id: 0
Frame Number: 9
Frame Length: 1514 bytes (12112 bits)
Capture Length: 1514 bytes (12112 bits)
[Frame is marked: True]
[Frame is ignored: False]
[Protocols in frame: eth:ip:tcp]
[Coloring Rule Name: HTTP]
[Coloring Rule String: http || tcp.port == 80]
```

Ethernet II, Src: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a), Dst: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)

```
Destination: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)
Address: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)
.... ..0. .... = LG bit: Globally unique address (factory default)
.... ...0 .... = IG bit: Individual address (unicast)
Source: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)
Address: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)
.... ..0. .... = LG bit: Globally unique address (factory default)
.... ...0 .... = IG bit: Individual address (unicast)
```

Type: IP (0x0800)

Internet Protocol Version 4, Src: 163.173.228.17 (163.173.228.17), Dst: 163.173.231.107 (163.173.231.107)

```
Version: 4
Header length: 20 bytes
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
Total Length: 1500
Identification: 0x72ed (29421)
Flags: 0x02 (Don't Fragment)
```



```
0... .... = Reserved bit: Not set
.1... .... = Don't fragment: Set
..0. .... = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (6)
Header checksum: 0xaf56 [correct]
    [Good: True]
    [Bad: False]
Source: 163.173.228.17 (163.173.228.17)
Destination: 163.173.231.107 (163.173.231.107)
Transmission Control Protocol, Src Port: http (80), Dst Port: 50074 (50074), Seq: 300, Ack: 491, Len: 1460
Source port: http (80)
Destination port: 50074 (50074)
[Stream index: 0]
Sequence number: 300      (relative sequence number)
[Next sequence number: 1760      (relative sequence number)]
Acknowledgment number: 491      (relative ack number)
Header length: 20 bytes
Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
Window size value: 1728
[Calculated window size: 6912]
[Window size scaling factor: 4]
Checksum: 0xd169 [validation disabled]
[SEQ/ACK analysis]
    [Bytes in flight: 1759]
[Reassembled PDU in frame: 12]
TCP segment data (1460 bytes)
```

Question 3.1

- Est-ce que le paramètre MSS (Maximum Segment Size) a un impact sur la taille des segments au niveau TCP.
- Quelle est la valeur du MTU (Maximum Transfer Unit) pour les datagrammes IP sur un réseau local Ethernet ?
- Est-ce que le MTU a un impact sur la fragmentation au niveau de la couche IP ?

Correction :

- Le MSS définit la taille de la partie donnée des segments envoyés par TCP
- Le MTU d'Ethernet est de 1500 octets
- Le MTU a impact sur la fragmentation IP, si le datagramme dépasse 1500 octets, IP fragmente pourvu que le bit DF ne soit pas à 1..

Question 3.2

Est-ce IP ou TCP ou les deux qui fragmentent dans le cas de l'observation ci-dessus ? Expliquez comment vous comprenez les mécanismes mis en œuvre pour que le transfert de données soit le plus efficace possible ?

Correction :

A cause de l'utilisation de l'option MSS (Maximum Segment Size) les datagrammes ne dépasseront pas la taille de 1500 octets (1460 + 20 d'entête IP + 20 d'entête TCP). Du coup, IP n'aura pas besoin de fragmenter puisque le client et le serveur sont sur le même réseau local Ethernet. D'après les informations sur la trace, tout laisse à penser que TCP fragmente.

Question 3.3

Quelle QoS est associée à chacun des datagrammes 8 et 9 ? Pensez vous que c'est un bon choix, pourquoi ?

Correction :

La trace indique dans les différents datagrammes un champ DSCP qui vaut 0x00. La QoS est associée est du Best Effort. C'est un bon choix, le trafic web n'est pas d'une grande importance.

Question 4

Une commande WireShark permet de dessiner le flot TCP à partir de la capture faite. L'outil WireShark affiche le graphique ci-après. Mettre en correspondance les différentes primitives de l'API Sockets AFINET, SOCKSTREAM qui correspondent à la mise en œuvre de TCP (ouverture/fermeture de connexion, envoi/réception de données) avec les différentes parties de la trace.

Correction :

Les primitives de l'API socket sont données dans les cadres en rouge en fonction des phases de la connexion : ouverture, transfert, fermeture.

	Time	163.173.231.107	163.173.228.17	Comment
connect ()	0.000627000	(30074) → (80) SYN	Seq = 0	socket(), bind(), listen(), accept() ont été faits avant
	0.001867000	(30074) ← (80) SYN, ACK	Seq = 0 Ack = 1	
	0.001912000	(30074) → (80) ACK	Seq = 1 Ack = 1	
send(), receive(), read(), write()	0.002414000	(30074) → (80) PSH, ACK - Len: 490	Seq = 1 Ack = 1	send(), receive(), read(), write()
	0.003139000	(30074) ← (80) ACK	Seq = 1 Ack = 491	
	0.003607000	(30074) → (80) PSH, ACK - Len: 299	Seq = 1 Ack = 491	
	0.003885000	(30074) ← (80) ACK - Len: 1460	Seq = 300 Ack = 491	
	0.004164000	(30074) → (80) ACK	Seq = 491 Ack = 1760	
	0.005002000	(30074) → (80) ACK - Len: 1460	Seq = 1760 Ack = 491	
	0.005030000	(30074) → (80) PSH, ACK - Len: 20	Seq = 3220 Ack = 491	
	0.005285000	(30074) → (80) ACK	Seq = 491 Ack = 3240	
	0.035639000	(30074) → (80) PSH, ACK - Len: 505	Seq = 491 Ack = 3240	
	0.036613000	(30074) → (80) PSH, ACK - Len: 299	Seq = 3240 Ack = 996	
	0.036797000	(30074) ← (80) ACK - Len: 1460	Seq = 3539 Ack = 996	
	0.036914000	(30074) → (80) ACK	Seq = 996 Ack = 4999	
	0.037032000	(30074) ← (80) ACK - Len: 1460	Seq = 4999 Ack = 996	
	0.037871000	(30074) ← (80) ACK - Len: 1460	Seq = 6459 Ack = 996	
	0.037915000	(30074) → (80) ACK	Seq = 996 Ack = 7919	
	0.037924000	(30074) ← (80) ACK - Len: 1460	Seq = 7919 Ack = 996	
	0.037973000	(30074) → (80) PSH, ACK - Len: 1221	Seq = 9379 Ack = 996	
	0.038008000	(30074) → (80) ACK	Seq = 996 Ack = 10600	
close() ou shutdown()	15.037206000	(30074) ← (80) FIN, ACK	Seq = 10600 Ack = 996	close() ou shutdown()
	15.037283000	(30074) → (80) ACK	Seq = 996 Ack = 10601	
	15.037527000	(30074) → (80) FIN, ACK	Seq = 996 Ack = 10601	
close() ou shutdown()	15.037947000	(30074) → (80) ACK	Seq = 10601 Ack = 996	

- Questions avancées - pour travailler votre côté Geek -

Question 5

On s'intéresse à la gestion de la fenêtre de contrôle de flux, à la taille de la fenêtre (Window Size) et à l'option Window Scale. Ils interviennent lors de l'ouverture de connexion de part et d'autre, au moment de l'échange des segments avec l'indicateur SYN.

L'option Window Scale est nécessaire pour que l'échange de données puisse remplir au maximum la connexion sur le réseau sous-jacent (il faut prendre la métaphore d'un tuyau, pipe en anglais). Cette option est utilisée quand le produit débit x délai A/R qui matérialise la capacité de remplissage maximale d'une connexion TCP est supérieur à 64 K octets.

Le réseau local utilisé dans le problème est un réseau Ethernet 1Gb/s (10^9 b/s). La proximité réseau du client et du serveur fait que le délai A/R est estimé à 1,5ms (0.0015 s) environ.

Question 5.1

Calculez le produit débit x délai A/R (BDP, bandwidth-delay product) pour la connexion que nous étudions.

Correction :

Dans notre cas, le débit x délai AR vaut $10^9 \times 0,0015$ soit 1 500 000 soit 1,5 Mb. Cette valeur donne le volume de données estimé que peut contenir la connexion en phase de transfert. Cette formule est utilisée pour les réseaux haut débit et à longue distance : Long Fat Network (LFN). On peut voir une connexion comme une sorte de pipeline dont on cherche à évaluer la taille par cette formule théorique. Il n'est pas donné plus d'information, mais on peut se poser la question si cette valeur est valable pour chaque sens indépendamment l'un de l'autre, ou, pour les deux sens à la fois. Il faut très probablement interpréter cette formule en fonction du contexte : Ethernet Half Duplex ou Full Duplex.

La valeur annoncée dans la trame 3 (question 4.4) au serveur http indique une taille de fenêtre de 8192 octets (valeur par défaut sur un système windows) et un facteur d'échelle de 4 (2^2). Cela définit l'asservissement du serveur http quand il a un rôle émetteur vis-à-vis de la capacité du navigateur quand celui-ci a un rôle récepteur⁴.

En continuant l'échange d'ouverture de connexion, avec la trame 4, le serveur http accepte la valeur de l'option window scale proposée. A ce stade de l'échange, le serveur http pourrait donc émettre au maximum 32784 octets ($8192 * 4$) sans être freiné par le contrôle de flux de TCP. Cela représente 262 272 bits pour se rapprocher d'une formulation BDP.

Complément : Dans l'entête TCP, le champ window (fenêtre) est sur 16 bits, le crédit peut prendre la 65 536 octets au maximum.

⁴ Avec le contrôle de flux en général, et dans TCP en particulier, le récepteur asservit l'émetteur à sa capacité mémoire en réception



Question 5.2

Comparez la capacité de remplissage théorique au remplissage visé par la connexion TCP entre notre client et notre serveur. D'après vous pourquoi cette connexion ne spécifie pas des paramètres pour mieux remplir le réseau ?

Correction :

Tout d'abord quelques compléments sur l'option Window Scale. Si on reprend la trace de la trame 3 on a :

```

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

Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa), Dst: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a)
Internet Protocol Version 4, Src: 163.173.231.107 (163.173.231.107), Dst: 163.173.228.17 (163.173.228.17)
Transmission Control Protocol, Src Port: 50074 (50074), Dst Port: http (80), Seq: 0, Len: 0
  Source port: 50074 (50074)
  Destination port: http (80)
  [Stream index: 0]
  Sequence number: 0 (relative sequence number)
  Header length: 32 bytes
  Flags: 0x002 (SYN)
  Window size value: 8192
    [Calculated window size: 8192]
  Checksum: 0xcfa1 [validation disabled]
  Options: (12 bytes), Maximum segment size, No-Operation (NOP), window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
    Maximum segment size: 1460 bytes
    No-Operation (NOP)
    Window scale: 2 (multiply by 4)
    No-Operation (NOP)
    No-Operation (NOP)
    TCP SACK Permitted Option: True

0000  00 0f 1f f9 ad 5a 00 26 6c 9a 4d aa 08 00 45 00  ....Z.& l.M...E.
0010  00 34 4d 92 40 00 80 06 9a 59 a3 ad e7 6b a3 ad  .4M.@... .Y...k..
0020  e4 11 c3 9a 00 50 3c ac 6c 05 00 00 00 00 80 02  ....P<. l.....
0030  20 00 cf a1 00 00 02 04 05 b4 01 03 03 02 01 01  ..... ..
0040  04 02

```

Dans la partie interprétée, comme dans la partie en hexadécimal, on voit bien que window scale, dans le champ option de l'entête, a la valeur 2 (décalage multiplicatif à appliquer). Il s'interprète comme 2^2 , et on retrouve bien le facteur d'échelle 4 annoncé en clair dans la trace de la trame 3.

D'après la RFC 1323, ce champ d'option de 3 octets a le format suivant :




```
TCP Window Scale Option (WSopt):      Kind: 3 Length: 3 bytes
```

```
+-----+-----+-----+
| Kind=3 |Length=3 |shift.cnt|
+-----+-----+-----+
```

`shift.cnt` est la puissance de 2 à appliquer pour obtenir le facteur d'échelle.

Le remplissage théorique tel que calculé est bien au-dessus du remplissage spécifié à travers les paramètres de crédit. 1,5Mb vs 262 272 octets. On peut imaginer que le client ne connaît pas les capacités réelles du serveur (latence réelle entre le client et le serveur, taille de la mémoire en réception, rapidité de l'application à retirer les données...). Il annonce donc des valeurs par défaut en faisant juste l'hypothèse que le facteur d'échelle n'engage pas à beaucoup de provisionnement de ressources. La taille de la fenêtre peut toujours être refusée par l'autre correspondant.

Lors de la réponse, il aura une meilleure estimation du délai A/R, il pourrait alors décider de demander à augmenter la taille de la fenêtre, ou à autoriser une fenêtre plus grande quand il joue le rôle de récepteur. Mais dans TCP, la valeur de la taille de fenêtre maximale n'est annoncée qu'à l'ouverture de connexion, et plus jamais après.

Je n'ai pas les réponses aux questions subsidiaires ci-dessous... plutôt des pistes de réponses dont je ne suis pas absolument certain.

Question subsidiaire 6 :

A partir de la trame 5 la taille de la fenêtre annoncée par le navigateur est augmentée : elle passe à 16425 x 4 soit 65 700 octets. Elle ne changera plus pendant tout le reste de l'échange. Pourquoi ?

Correction proposée :

65 700 octets c'est 525 600 bits pour se rapprocher de la formule BDP. C'est moins que la valeur théorique calculée en 8.1 mais c'est mieux que la valeur définie à l'issue des échanges de trames 3 et 4.

On ne peut faire que des conjectures pour expliquer pourquoi, il faudrait avoir plus de détails sur la mise en œuvre de TCP sur le client. On peut penser que l'arrivée de la trame 4 permet l'estimation plus précise du délai A/R et qu'un mécanisme d'adaptation se met en route. Comme la valeur de `WindowoScale` ne peut plus être changée. Afin de s'adapter, le navigateur annonce une taille de fenêtre plus grande dans le champ `Window Size`, qui influe sur la taille globale de la fenêtre, même si celle-ci reste en dessous du maximum théorique.

En effet, Windows 7 ou Vista utilisent une fonction "TCP Window Auto-tuning". Avec la commande `netsh interface tcp show global`,



effectuée sur le client on obtient :

Recherche du statut actif...

Parametres TCP globaux

```
-----  
Etat de mise a l'echelle cote reception      : enabled  
Etat de dechargement Chimney                 : automatic  
Etat NetDMA                                 : enabled  
Acces direct au cache                       : disabled  
Reglage auto fenetre de reception           : normal  
Fournisseur de ctrl surcharge comp.         : none  
Fonctionnalite ECN                          : disabled  
Horodatages RFC 1323                       : disabled
```

Dans notre cas, le client a donc la possibilité de changer la taille de sa fenêtre de réception puisque on a la ligne :

"Reglage auto fenetre de reception : normal "

est positionné (<http://www.speedguide.net/articles/windows-7-vista-2008-tweaks-2574>). En effet, à l'url précédente il est mentionné :

"**normal**: default value, allows the receive window to grow to accommodate most conditions".

Le lien est d'ailleurs intéressant pour avoir d'autres informations complémentaires sur les aspects internes de TCP et de son réglage sur le système windows.

Pour information, un lien intéressant aussi : <http://www.speedguide.net/articles/linux-tweaking-121>

Remarque : La fenêtre de contrôle de flux pourrait encore changer pendant la vie de la connexion.

Question subsidiaire 7

Appliquez le même raisonnement dans le sens inverse quand le serveur http est récepteur et le navigateur est émetteur. Il faut prendre la trame 4 dans l'échange donné en question 4, les paramètres window size et window scale sont annoncés dans la partie option du segment TCP.

Correction proposée :

Dans le sens inverse serveur http récepteur et navigateur, le serveur annonce en trame 4 un crédit 5840 octets (46 720 bits). Cf ci-après.

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]

Transmission Control Protocol, Src Port: http (80), Dst Port: 50074 (50074), Seq: 0, Ack: 1, Len: 0	
Source port: http (80)	
Destination port: 50074 (50074)	
[Stream index: 0]	
Sequence number: 0 (relative sequence number)	
Acknowledgment number: 1 (relative ack number)	
Header length: 32 bytes	
Flags: 0x012 (SYN, ACK)	
window size value: 5840	
[Calculated window size: 5840]	
Checksum: 0xc222 [validation disabled]	
Options: (12 bytes), Maximum segment size, No-operation (NOP), No-operation (NOP), SACK permitted, No-operation (NOP), window scale	
+ Maximum segment size: 1460 bytes	
+ No-operation (NOP)	
+ No-operation (NOP)	
+ TCP SACK Permitted Option: True	
+ No-operation (NOP)	
+ window scale: 2 (multiply by 4)	
+ [SEQ/ACK analysis]	

0000	00 26 6c 9a 4d aa 00 0f	1f f9 ad 5a 08 00 45 00	.&l.M... ..Z..E.
0010	00 34 00 00 40 00 40 06	27 ec a3 ad e4 11 a3 ad	.4..@.@.
0020	e7 6b 00 50 c3 9a f9 ab	1c f2 3c ac 6c 06 80 12	.k.P.... ..<.l...
0030	16 d0 c2 22 00 00 02 04	05 b4 01 01 04 02 01 03	...".....
0040	03 02		..

Dans cette trame, on voit que le champ window scale vaut 0x16d0, soit 5840 octets (on a bien $1 \cdot 16^3 + 6 \cdot 16^2 + 13 \cdot 16$, "d" vaut "13", soit $4096 + 1536 + 208 = 5840$), donc le facteur d'échelle n'a pas été appliqué. Mais potentiellement il représente une taille de fenêtre de 23 360 octets soit 186 880 bits pour se rapprocher d'un raisonnement BDP.

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]

+

Frame 7: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

+

Ethernet II, Src: DellEsgP_f9:ad:5a (00:0f:1f:f9:ad:5a), Dst: Inventec_9a:4d:aa (00:26:6c:9a:4d:aa)

+

Internet Protocol Version 4, Src: 163.173.228.17 (163.173.228.17), Dst: 163.173.231.107 (163.173.231.107)

+

Transmission Control Protocol, Src Port: http (80), Dst Port: 50074 (50074), Seq: 1, Ack: 491, Len: 0

Source port: http (80)

Destination port: 50074 (50074)

[Stream index: 0]

Sequence number: 1 (relative sequence number)

Acknowledgment number: 491 (relative ack number)

Header length: 20 bytes

+

Flags: 0x010 (ACK)

Window size value: 1728

[Calculated window size: 6912]

[window size scaling factor: 4]

+

Checksum: 0x1116 [validation disabled]

+

[SEQ/ACK analysis]

[This is an ACK to the segment in frame: 6]

[The RTT to ACK the segment was: 0.000725000 seconds]

0000

00 26 6c 9a 4d aa 00 0f 1f f9 ad 5a 08 00 45 00

.&l.M... ..Z..E.

0010

00 28 72 e9 40 00 40 06 b5 0e a3 ad e4 11 a3 ad

.(r.@.@.

0020

e7 6b 00 50 c3 9a f9 ab 1c f3 3c ac 6d f0 50 10

.k.P.... ..<.m.P.

0030

06 c0 11 16 00 00 00 00 00 00 00 00

.

En frame 7 un crédit de 6912 octets avec le facteur d'échelle pris en compte est annoncé ($4 \times 1728 = 6912$ octets soit 55 296 bits, remarquons que 0x06c0 c'est $6 \times 16^2 + 12 \times 16 = 1728$ octets, "c" en hexadécimal c'est 12 en décimal). La première valeur de crédit annoncée dans la réponse à l'ouverture de connexion active du navigateur, était 23 360 octets et là on a 6912 octets. Le serveur ne modifie donc pas son crédit et donc n'adapte pas la taille de la fenêtre de contrôle de flux de façon dynamique.

52/52