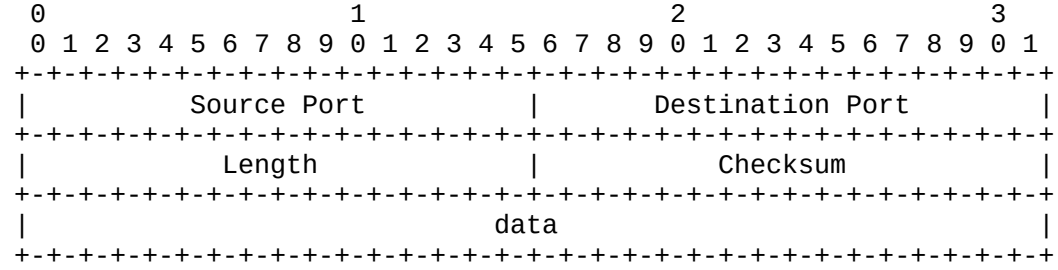
UDP ET TCP : PROTOCOLES DE TRANSPORT

1. UDP (USER DATAGRAM PROTOCOL)

1.1. Format de message



1.2. Exercice

1.2.1. Quelles sont les fonctionnalités assurées par UDP ?

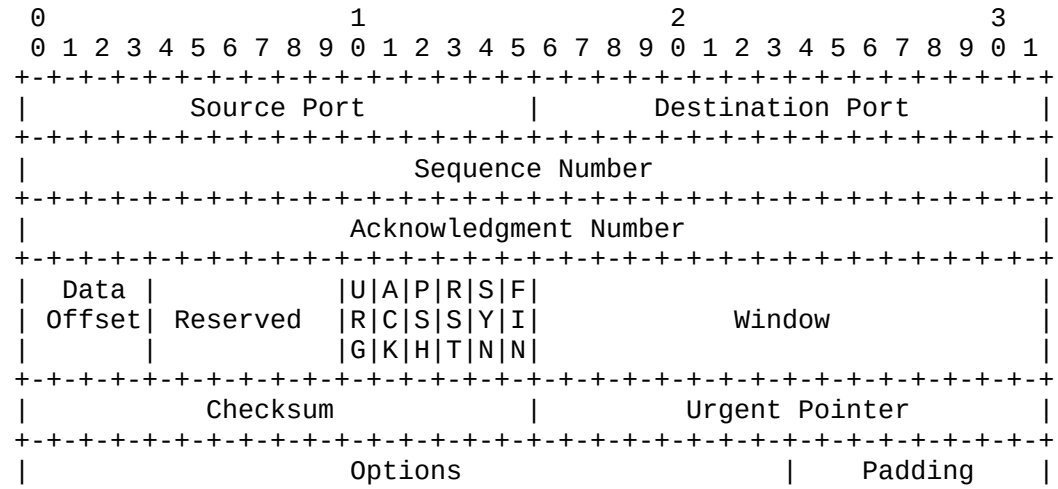
2. TCP (TRANSMISSION CONTROL PROTOCOL)

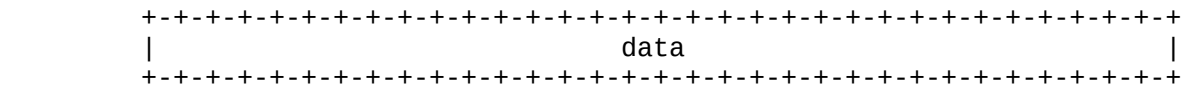
2.1. Généralités

TCP est un protocole de transport de bout en bout, offrant un service de remise **fiable** de flux d'octets en **full-duplex** et en **mode connecté**. Il met notamment en œuvre des mécanismes de :

- (dé)multiplexage ;
- contrôle d'erreur ;
- contrôle de flux ;
- contrôle de congestion.

A la différence des protocoles en mode connecté que nous avons pu voir jusqu'à présent, TCP n'utilise qu'un seul format de segment qui sert aussi bien à établir/libérer une connexion qu'à transférer des données :





2.2. Etablissement de connexion

L'établissement d'une connexion TCP se fait par échange de trois messages (*three-way handshake*).

2.2.1. Pourquoi trois messages ? Deux n'auraient-ils pas suffi ?

2.2.2. Rappeler sur un schéma les messages échangés en précisant les champs essentiels à l'établissement.

2.2.3. Pourquoi la numérotation n'est-elle pas commencée à 0 ?

2.2.4. Lorsqu'un hôte A reçoit deux segments SYN en provenance d'un même port d'un hôte B, le second SYN peut être une retransmission du SYN original ou alors une nouvelle requête de connexion (cas d'une panne suivie d'un redémarrage de B).

a. Comment A fait-il la différence entre ces deux cas ?

b. Décrire par un algorithme le comportement de TCP lors de la réception d'un segment SYN.

2.3. Numérotation des données

L'une des principales propriétés de TCP est la fiabilité (extraits du RFC 793) :

```
The TCP must recover from data that is damaged, lost, duplicated, or
delivered out of order by the internet communication system.  This
is achieved by assigning a sequence number to each octet
transmitted, and requiring a positive acknowledgment (ACK) from the
receiving TCP.  If the ACK is not received within a timeout
interval, the data is retransmitted.  At the receiver, the sequence
numbers are used to correctly order segments that may be received
out of order and to eliminate duplicates.  Damage is handled by
adding a checksum to each segment transmitted, checking it at the
receiver, and discarding damaged segments.
```

TCP numérote donc ses octets de données et non pas ses segments. Le numéro de séquence du premier octet de données est transmis avec le segment, dans le champ `SequenceNumber` de l'en-tête TCP (32 bits), et est appelé numéro de séquence du segment.

2.3.1. IP possède un champ `Identification` pour ses datagrammes. N'y-a-t-il pas redondance avec le champ `SequenceNumber` de TCP ?

2.3.2. Les 32 bits sont suffisants pour couvrir 4 milliards d'octets de données. Même en considérant qu'une telle quantité de données ne sera jamais transférée sur une seule connexion, pourquoi est-il possible de voir le numéro de séquence passer de 2^{31}-1 à 0 ?

2.3.3. On considère TCP opérant sur un lien à 1 Gbit/s.

a. Combien de temps met le numéro de séquence pour reboucler complètement ?

b. On utilise dans les options une estampille temporelle, codée sur 32 bits, qui s'incrémente 1000 fois pendant le temps trouvé précédemment. Combien de temps met l'estampille pour reboucler ?

2.4. Acquittement des données

Les segments TCP véhiculent également un numéro d'acquittement dans le champ `AcknowledgmentNumber`, qui représente le numéro de séquence du prochain octet de données attendu dans le flux de données en sens inverse.

2.4.1. On considère une connexion TCP inactive (pas de transfert de données en cours) entre des entités A et B. Une tierce partie a piraté la communication et a pris connaissance des numéros de séquence utilisés des deux côtés.

a. Supposons que la tierce partie envoie à A, en se faisant passer pour B, un segment contenant 100 octets de données. Que se passe-t-il ?

Indice :

```
In response to sending data the TCP will receive acknowledgments.  The
 following comparisons are needed to process the acknowledgments.
       SND.UNA = oldest unacknowledged sequence number
       SND.NXT = next sequence number to be sent
       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
 A new acknowledgment (called an "acceptable ack"), is one for which
 the inequality below holds:
     SND.UNA < SEG.ACK =< SND.NXT
If an incoming segment is not acceptable, an acknowledgment
should be sent in reply (unless the RST bit is set, if so drop
the segment and return):
     <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
After sending the acknowledgment, drop the unacceptable segment
and return (* to the current state *)
```

b. Supposons maintenant que la tierce partie envoie à chaque extrémité un segment de 100 octets en se faisant passer pour l'autre extrémité. Que se passe-t-il ? Et que se serait-il passé si A avait envoyé plus tard 200 octets de données à B ?

2.5. Contrôle de flux

Pour son contrôle de flux, TCP utilise un mécanisme de fenêtre coulissante de taille variable :

```
TCP provides a means for the receiver to govern the amount of data
sent by the sender.  This is achieved by returning a "window" with
every ACK indicating a range of acceptable sequence numbers beyond
the last segment successfully received.  The window indicates an
allowed number of octets that the sender may transmit before
receiving further permission.
```

La taille courante de la fenêtre est communiquée dans le champ `Window` (également appelé `AdvertisedWindow`) du segment :

```
The number of data octets beginning with the one indicated in the
acknowledgment field which the sender of this segment is willing to
accept.
```

2.5.1. Il s'agit de concevoir un protocole fiable, orienté flux d'octets, à fenêtre coulissante. Ce protocole est destiné à fonctionner sur un réseau à 100 Mbit/s. Le RTT (*Round-Trip Time*) du réseau est de 100 ms, et le MSL (*Maximum Segment Lifetime*) de 60 s.

a. Sur combien de bits doit-on le champ `AdvWin` du protocole ?

b. Sur combien de bits doit-on coder le champ `SeqNum` du protocole ?

c. Parmi les valeurs données dans l'énoncé, quelles sont celles qui peuvent être obtenues de manière fiable ?

2.5.2. Un émetteur sur une connexion TCP qui reçoit un `AdvWin` à 0 sonde périodiquement le récepteur pour détermine quand la fenêtre s'ouvre. Pourquoi le récepteur aurait-il besoin d'un temporisateur supplémentaire s'il était responsable de signaler explicitement et spontanément l'ouverture de la fenêtre ?

2.5.3. TCP est orienté-octet : la numérotation et la fenêtre portent sur des octets. Quand est-ce que TCP décide de constituer et d'envoyer un segment ?

2.5.4. L'algorithme de Nagle permet une amélioration de l'efficacité proposée pour les applications produisant des données octet par octet. Elle est intégrée dans

la plupart des implémentations TCP, et impose à l'émetteur de stocker les octets de données passées par l'application source (même s'il y a des `push`) avant de constituer un segment, tant qu'il n'a pas MSS (*Maximum Segment Size*) octets ou tant qu'il n'a pas reçu le ACK du segment précédent.

a. Supposons que les caractères `abcdefghi` soient envoyées, à raison d'une lettre par seconde, sur une connexion TCP dont le RTT est de 4,1 s. Faire un schéma temporel des émissions de segments.

b. Si les caractères sont tapés sur une connexion Telnet full-duplex, que peut observer l'utilisateur ?

c. On considère maintenant des mouvements de souris qui sont envoyés sur la connexion. En supposant que les changements de position de souris sont envoyés tous les RTT, qu'observe l'utilisateur lorsque l'algorithme de Nagle est ou n'est pas utilisé ?

2.5.5. Le syndrome de *Silly Window* correspond aux situations où l'émetteur a de gros volumes de données à envoyer mais où les segments sont de petite taille suite à des valeurs de fenêtre de contrôle de flux faible. Donner un exemple de telles situations et proposer des mesures préventives.

2.6. Temporisateur de retransmission

Un élément important du contrôle d'erreur est le temporisateur de retransmission. La procédure de dimensionnement proposée initialement dans le RFC 793 est la suivante :

```
Measure the elapsed time between sending a data octet with a
particular sequence number and receiving an acknowledgment that
covers that sequence number (segments sent do not have to match
segments received).  This measured elapsed time is the Round Trip
Time (RTT).  Next compute a Smoothed Round Trip Time (SRTT) as:
  SRTT = ( ALPHA * SRTT ) + ((1-ALPHA) * RTT)
and based on this, compute the retransmission timeout (RTO) as:
  RTO = min[UBOUND,max[LBOUND,(BETA*SRTT)]]
where UBOUND is an upper bound on the timeout (e.g., 1 minute),
LBOUND is a lower bound on the timeout (e.g., 1 second), ALPHA is
a smoothing factor (e.g., .8 to .9), and BETA is a delay variance
factor (e.g., 1.3 to 2.0).
```

2.6.1. Pourquoi la valeur du temporisateur doit-elle être ajustée dynamiquement et ce, pour chaque connexion ?

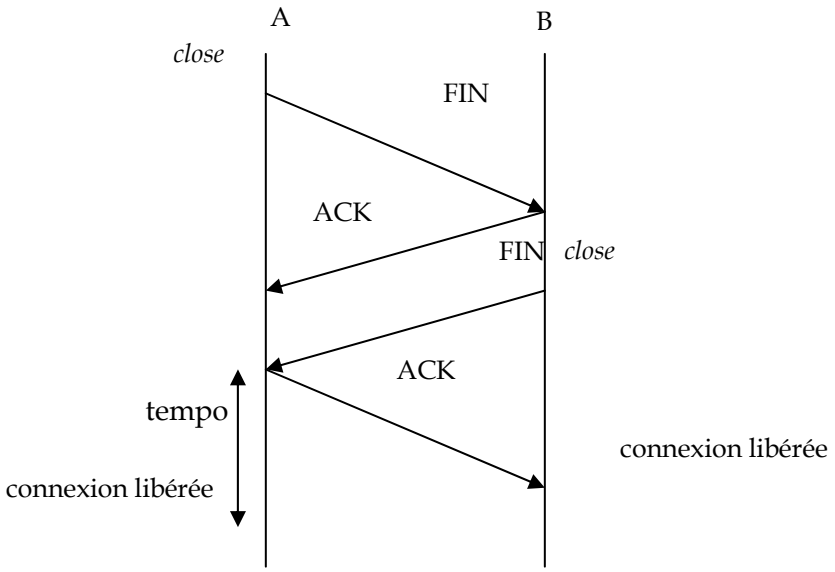
2.6.2. La valeur courante de SRTT est de 30 ms. Les acquittements des trois segments de données suivants reviennent après 26 ms, 32 ms et 24 ms respectivement. Que devient l'estimation du délai aller-retour ? (On prendra `ALPHA` = 0,9.)

2.6.3. On suppose que, lorsqu'un segment est envoyé plus d'une fois, la mesure du RTT est prise entre l'instant de la première émission et l'instant de réception du premier ACK. Montrer que pour une connexion dont la fenêtre est de un segment et pour laquelle chaque segment est transmis deux fois, on a SRTT qui tend vers l'infini. Que peut-on proposer ?

2.6.4. L'algorithme de Karn-Partridge propose de doubler la valeur du temporisateur à chaque nouvelle retransmission, et ce jusqu'à ce qu'un segment soit acquitté du premier coup (auquel cas, la valeur de SRTT est recalculée et la nouvelle valeur de TO déduite). Comment justifier cette proposition ?

2.7. Libération de connexion

L'échange de segments est illustré ci-dessous :



2.7.1. Lorsque TCP envoie un [SYN, SeqNum = x] ou un [FIN, SeqNum = x], le segment ACK correspondant porte AckNum = x+1. En d'autres termes, les segments SYN et FIN consomment une unité de l'espace de numérotation. Est-ce vraiment nécessaire ? Aurait-on pu avoir à la place [ACK, AckNum = x] ?