La couche 4 : Transport

- Elle permet aux applications d'utiliser le réseau sous-jacent (couche 3).
- Elle assure le transport de bout en bout des données, c'est à dire de la couche 4 de l'expéditeur a celle du destinataire.
- Elle doit offrir à l'utilisateur une qualité de service et une transparence vis à vis des réseaux traversés sous-jacents.

Port

- Afin d'identifier chaque applications accédant aux services réseau, la couche application utilise la notion de port. Un port est le point d'entrée d'une application.
- Les numéros de ports étant codés sur 16 bits, il en existe 2^{16}. Les numéros de port inférieur à 1024 sont réservés (21 FTP, 80 HTTP, 143 IMAP, etc)
- Le système d'exploitation affecte aux applications demandant un service réseau un numéro de port.

Socket

- A chaque application utilisant des services réseau, le système d'exploitation lui affecte un identifiant nommé socket : couple (numéro de port/adresse IP)
- Lorsque 2 applications échangent des données, le système d'exploitation identifie cet échange par l'association (protocole,IP local, port local, IP distant, port distant).

Transmission Control Protocol

- Fonctionne en mode connecté
- Apporte une fiabilité
- Offre une qualité de service
- Contrôle de flux
- Séquençage

ATM Adaptation layer

- Protocole Asynchronous Transfert Mode (protocole couche 2) qui assure l'interface avec les couches supérieures
- Divisée en 2 sous-couches :
  - Une assurant la liaison avec la couche supérieure
  - Une assurant la gestion des messages (découpage, assemblage)
- Ce protocole assure la transmission de bout en bout des messages et fonctionne sur IP

AAL

- Il offre 3 classes de service
  - Classe 1 : émule un circuit virtuel, débit fixe (parole en téléphonie)
  - Classe 2 : idem avec débit variable (utilisé par l'Universal Mobile Telecommunications System – 3G)
  - Classe 5 : transport de données

Format en-tête TCP

Format en-tête TCP

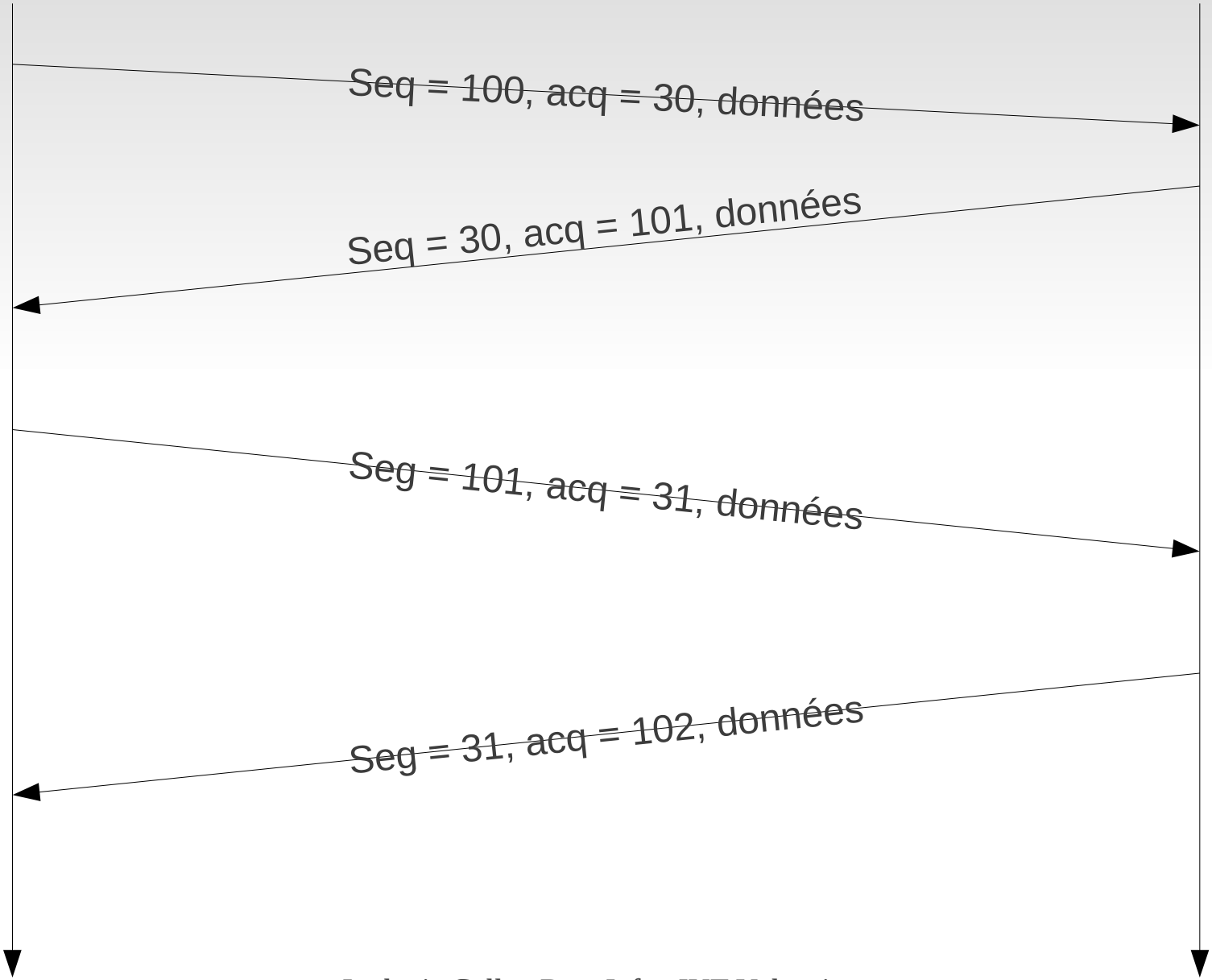
mots	bits 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	port source																port destination															
2	numéro de séquence (seq)																															
3	numéro d'acquittement (acq)																															
4	long. En-tête				réserve						urg	ack	psh	rst	syn	fin	fenêtre															
5	checksum																pointeur d'urgence															
6	options																															

port source : port sur lequel on va communiquer
port destination : port que l'on contacte
seq : numéro du premier octet du segment
acq : indique le numéro seq du prochain serment à envoyer (acquitte les données reçu précédemment)
longueur en tête en nombre de mot de 32 bits
reservé : remplissage de bits à 0
urg : signale l'utilisation du champ pointeur d'urgence ci dessous
ack : indique l'utilisation du champ numéro d'acquittement ci dessus
psh : demande un envoi rapide des données
rst : demande de réinitialisation de la connexion TCP
fin : demande de fermeture de la connexion
fenêtre : indique la taille (la fenêtre) des messages que le destinataire peut recevoir en octet
checksum : vérification des erreurs
pointeur d'urgence : pointe le dernier octet d'un message urgent

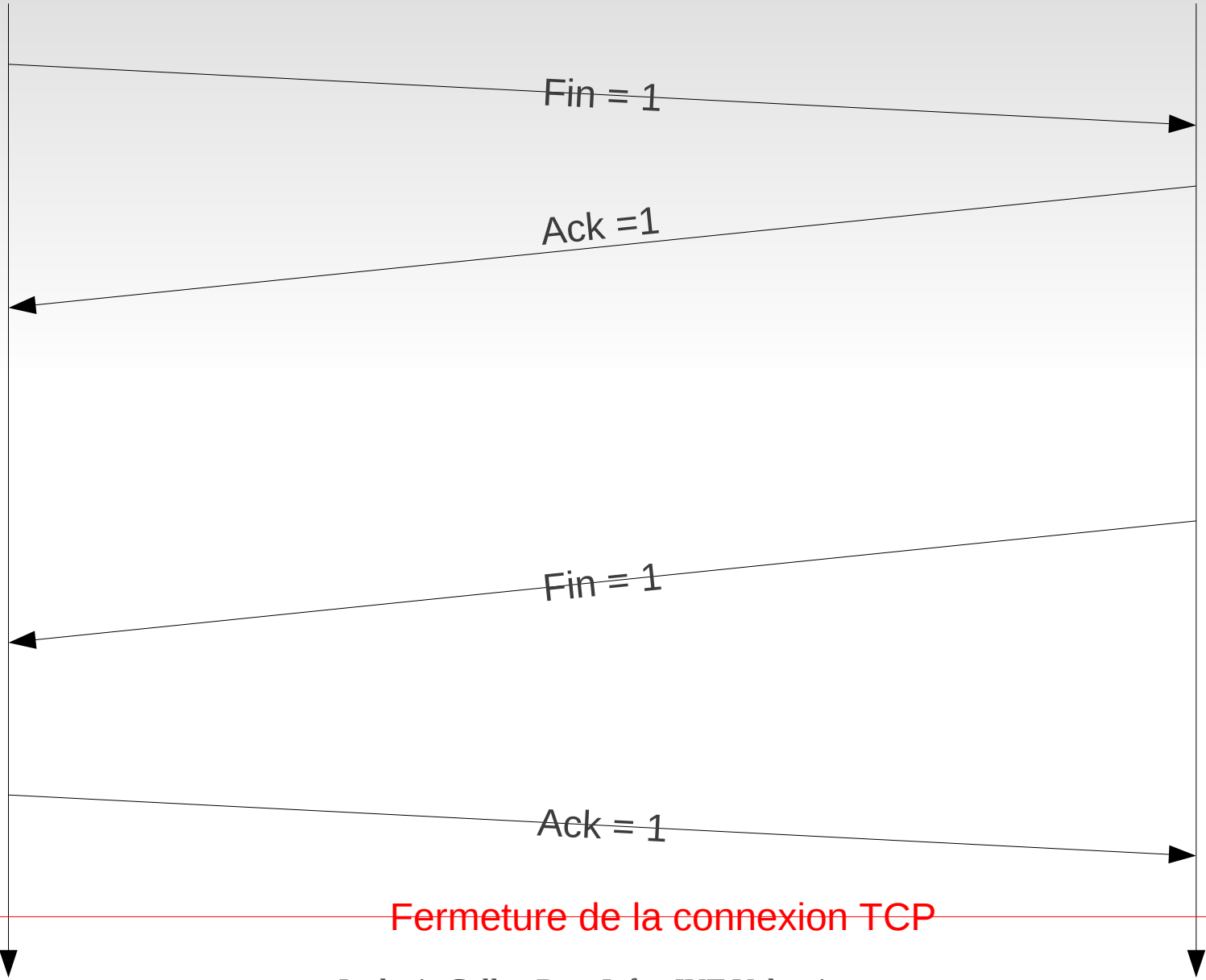
Initialisation de la connexion

- Démarrage du serveur en mode passif
- Client fait une ouverture active
- Client envoie un segment de connexion (syn=1,seq= valeurX, acq=0)
- Serveur acquitte (syn=1,ack=1, seg = valeurY, acq=valeurX + 1)
- Client acquitte (ack=1, seg = valeurX +1, acq = valeurY + 1)
- Début de l'échange de données

Échange de données



Échange de données



Fermeture de la connexion TCP

Interface entre la couche transport et l'application

- Une application communique grâce aux sockets.
- Il faut donc mettre en œuvre les sockets coté client et coté serveur

Coté serveur

- Il faut créer le socket en indiquant le numéro de port (commande bind)

```
BIND(2)                 Linux Programmer's Manual                 BIND(2)

NAME
       bind - bind a name to a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int bind(int sockfd, const struct sockaddr *addr,
                socklen_t addrlen);
```

Coté serveur

- Puis attendre une connexion client (commande listen)

```
LISTEN(2)                Linux Programmer's Manual                LISTEN(2)

NAME
       listen - listen for connections on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int listen(int sockfd, int backlog);
```

Coté client

- Créer le socket et demander la connexion au client (commande connect)

```
CONNECT(2)              Linux Programmer's Manual              CONNECT(2)

NAME
       connect - initiate a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int connect(int sockfd, const struct sockaddr *addr,
                   socklen_t addrlen);
```

Coté serveur

- Accepter la connexion demandé par le client (commande accept)

```
ACCEPT(2)                Linux Programmer's Manual                ACCEPT(2)

NAME
       accept - accept a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

       #define _GNU_SOURCE
       #include <sys/socket.h>

       int accept4(int sockfd, struct sockaddr *addr,
                   socklen_t *addrlen, int flags);
```

Envoi de données

- *Send*

```
SEND(2)                 Linux Programmer's Manual                SEND(2)

NAME
       send, sendto, sendmsg - send a message on a socket

SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t send(int sockfd, const void *buf, size_t len, int flags);

       ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                      const struct sockaddr *dest_addr, socklen_t addrlen);

       ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

Réception de données

- *Recv*

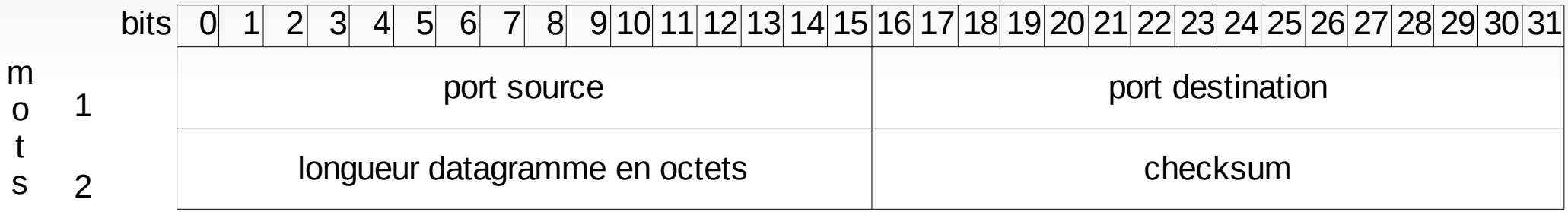
```
RECV(2)                 Linux Programmer's Manual                 RECV(2)

NAME
       recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t recv(int sockfd, void *buf, size_t len, int flags);

       ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                        struct sockaddr *src_addr, socklen_t *addrlen);

       ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

User Datagram Protocol

- Non connecté
- Peu fiable
- Plus simple que TCP
- Permet des débits élevés (applications temps réels, flux audio et vidéo)
- Exemple de protocoles utilisant UDP : SNMP, DNS, RIP, traceroute, DHCP

En-tête

Format en-tête UDP



Interface avec l'application

- Utilisation de socket mais sans connexion
  - Coté serveur : création (Bind) puis listen
  - Coté client : création
  - Transfert de données send et recv