Cours de Réseau et communication Unix n°8

Edouard THIEL

Faculté des Sciences

Université d'Aix-Marseille (AMU)

Septembre 2014

Les transparents de ce cours sont téléchargeables ici :
http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ens/rezo/

Lien court : http://j.mp/rezocom

Plan du cours n°8

1. Protocole TCP
2. Protocole SMTP
3. Protocole POP3
4. Protocole HTTP

Protocoles

Protocole = format de message ou syntaxe + règles d'échange

Exemples : $\underbrace{\text{IP, ICMP, TCP, UDP}}_{\text{binaire}}$, $\underbrace{\text{HTTP, SMTP, POP3}}_{\text{ascii 7 bits}}$

Protocoles binaires : (couches 3, 4)

- très compact
- non lisible humainement
- nécessite un programme pour dialoguer

Protocoles texte : (couche 7)

- moins compact, surcoût négligeable
- lisible et traçable
- on peut dialoguer directement au clavier avec `netcat`

1 - Protocole TCP

Protocole binaire, couche 4, RFC 793 (1981)

Protocole fiable

TCP = protocole fiable, au dessus de IP (non fiable).

Permet de

- multiplexer les échanges grâce au concept de port
- initialiser, maintenir et fermer une connexion
- remettre en ordre les segments TCP
- acquitter et réémettre les segments TCP
- adapter le flot de données à la charge du réseau

Segment TCP

Entête binaire : entre 20 et 60 octets

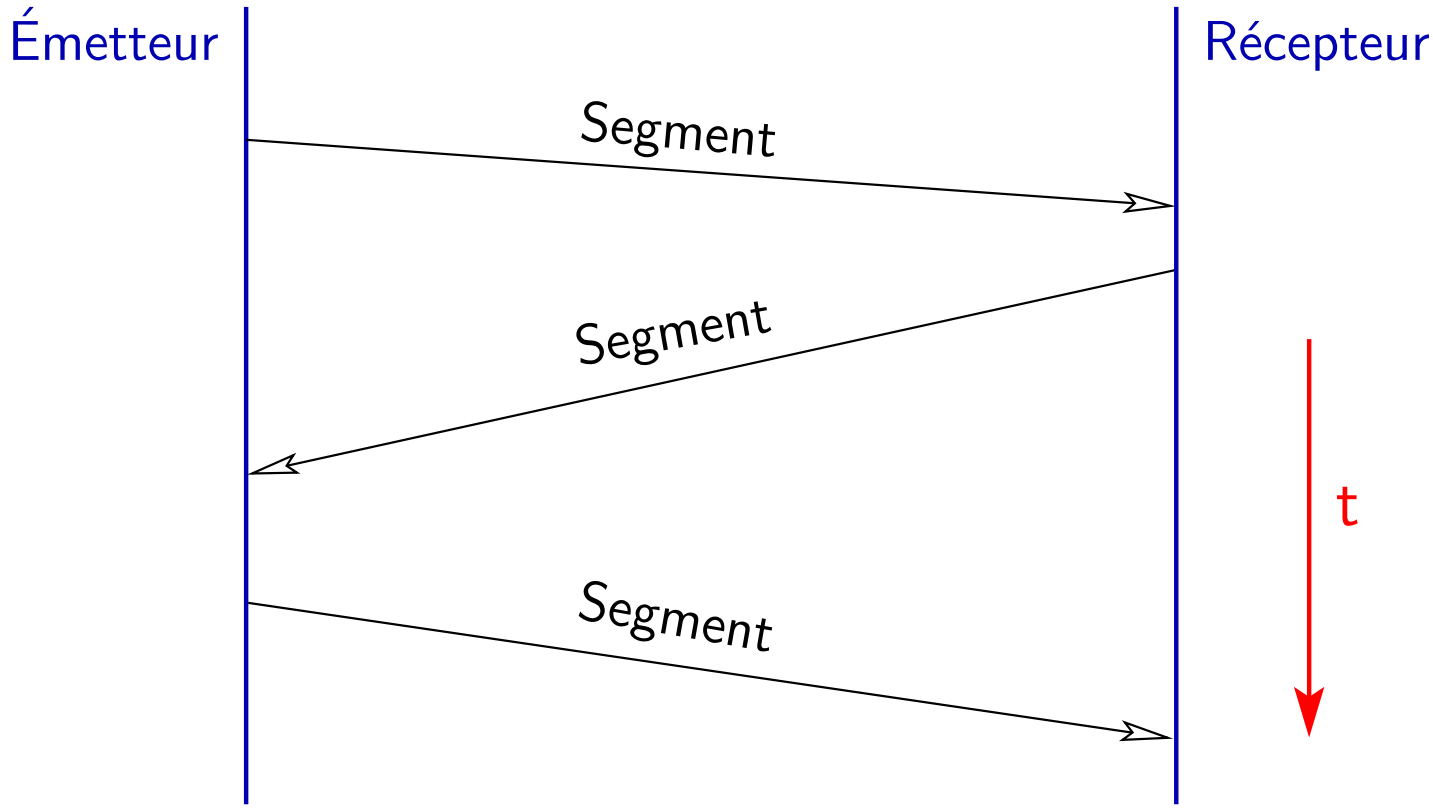
- Port source, destination (2*16 bits)
- Numéro d'ordre (32 bits)
- Numéro accusé de réception (32 bits)
- Taille de l'entête en multiples de 32 bits (4 bits)
- Drapeaux (bits) : ACK, RST, SYN, FIN, ... (12 bits)
- Fenêtre (16 bits)
- Somme de contrôle sur l'entête (16 bits)
- Options, etc
- Bourrage (alignement 32 bits)

Drapeaux

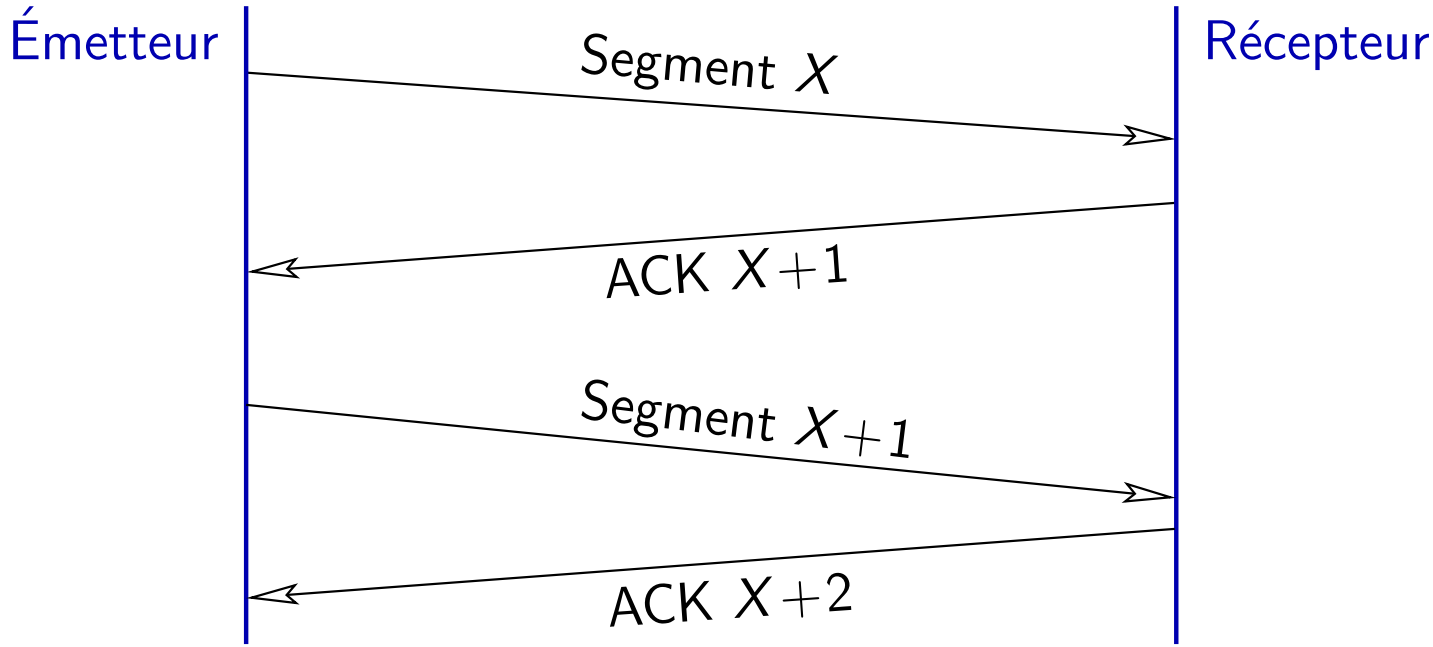
Drapeaux = 9 flags sur 1 bit + 3 bits réservés

- ACK : accusé de réception (acknowledgement)
- RST : rupture anormale de la connexion (reset)
- SYN : établissement de connexion (synchronisation)
- FIN : demande la fin de la connexion
- ECN : signale la présence de congestion
- ...
- + bits réservés pour un usage futur

Diagramme temporel



Acquittement

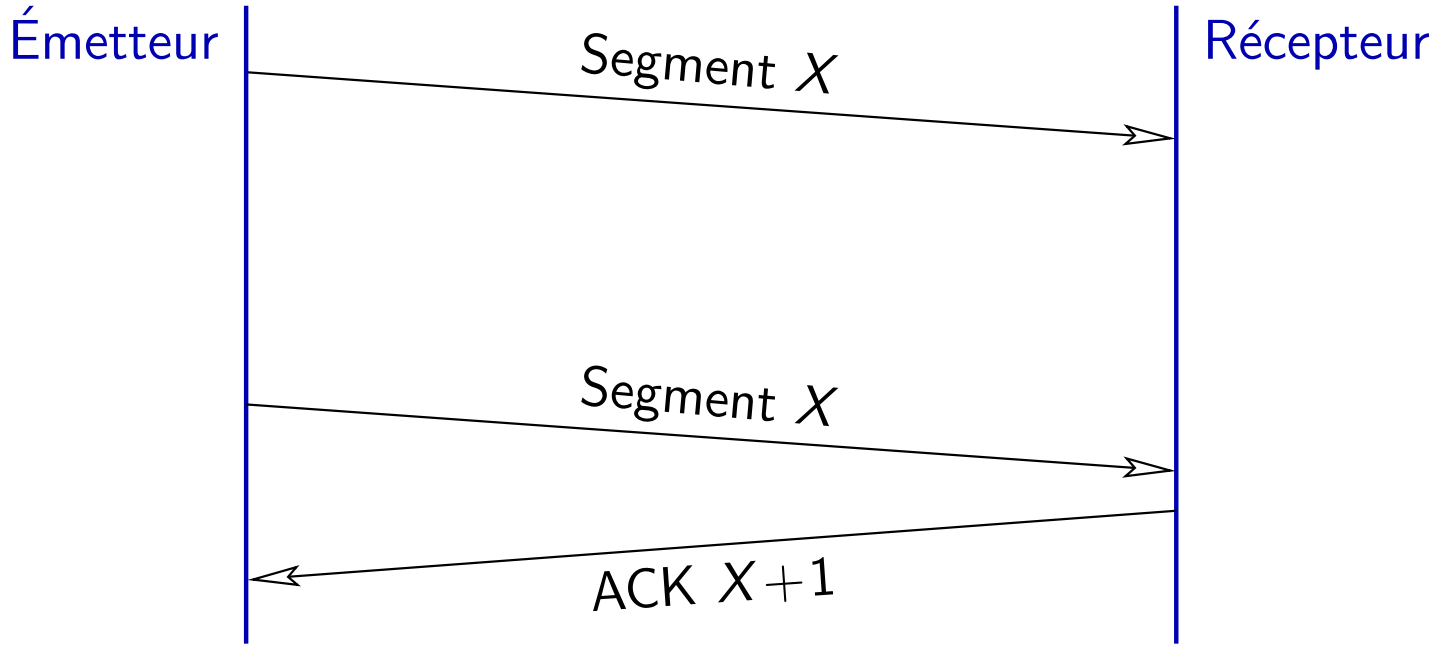


Émission d'un segment avec un numéro d'ordre X
$\rightarrow$ Réponse avec drapeau ACK + numéro accusé réception (NAR)

Signifie : tous les segments $<$ NAR ont été reçus.

Remarque : NAR = numéro d'ordre du prochain segment attendu

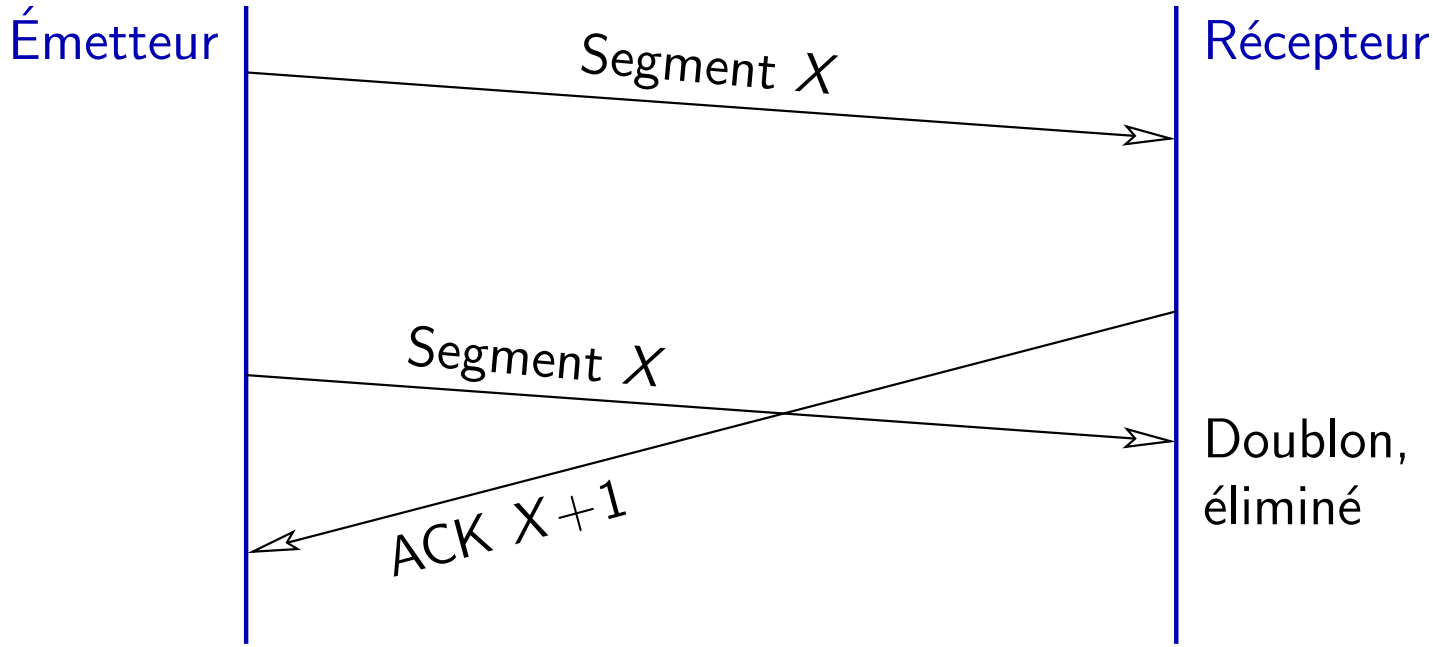
Réémission



Les segments émis sont mémorisés jusqu'à acquittement.

Au bout d'un timeout, un segment non acquitté est réémis.

Doublon



Si le segment est réémis (par la source ou un nœud intermédiaire), cela peut créer un doublon.

Doublons détectés au niveau récepteur sur numéro d'ordre.

Connexion

Établissement d'une connexion : suppose

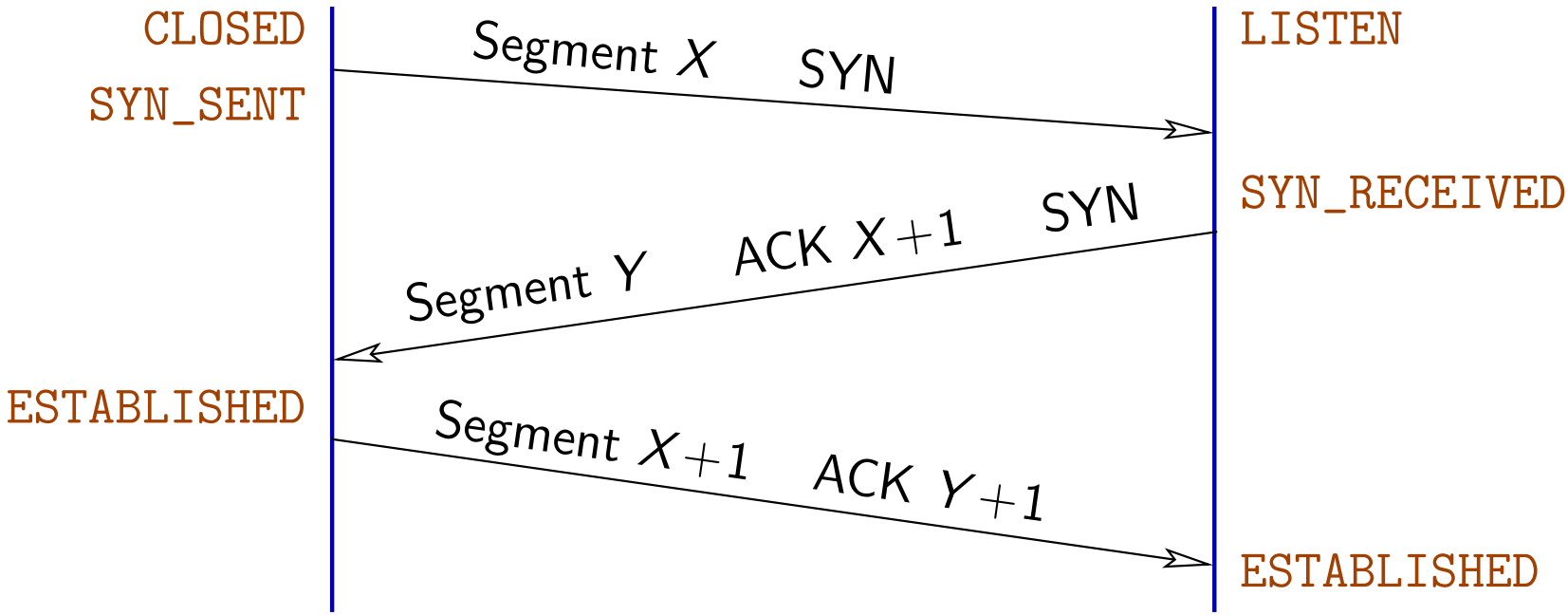
- une machine en demande de connexion : le client (état `CLOSED` $\rightarrow$ `SYN_SENT`)
- une machine en attente de connexion : le serveur (état `LISTEN`)

Chacune tire au sort un numéro d'ordre de départ : X pour le client, Y pour le serveur.

Connexion =

- Se passer les numéros d'ordre (synchronisation) ;
- passer à l'état `ESTABLISHED`

Connexion : poignée de main en 3 temps



On utilise le flag SYN

Déconnexion

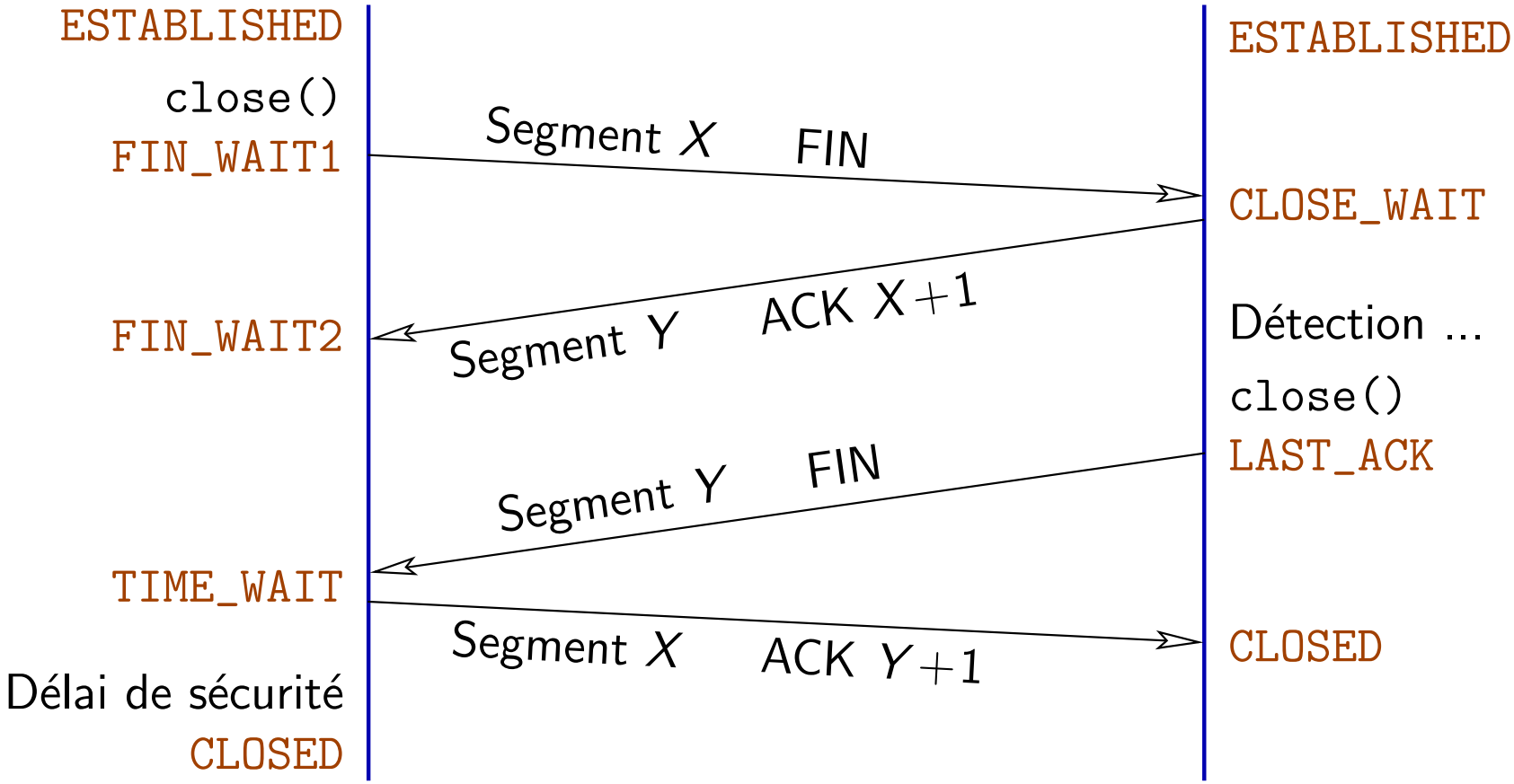
Plus complexe : ne doit pas être confondu avec une panne

Principe :

- je vais fermer / je sais que tu vas fermer
- je suis fermé / moi aussi
- je suis complètement fermé ...

Les numéros d'ordre ne changent pas pour éviter des problèmes subtils.

Déconnexion en 4 temps



On utilise le flag FIN

Il existe des variantes en 3 temps.

2 - Protocole SMTP

Protocole texte pour envoyer des mails.

RFC 821 (1982, ...)

Protocole Ascii 7 bits, sur TCP

Utilise le port 25

Échange de questions-réponses avec le serveur.

Exemple

`$ netcat` *adr-serveur* `25`	
	`220` *adr-serveur* `SMTP Ready`
`HELO` *adr-client*	
	`250` *adr-serveur*
`MAIL FROM:` *email-auteur*	
	`250 Sender ok`
`RCPT TO:` *email-dest*	
	`250 Recipient ok`
`DATA`	
	`354 Enter mail, end with "."`
`Subject:` *sujet* *Ceci est le corps* *du message à envoyer.* `.`	
	`250 Ok`
`QUIT`	
	`221 Closing connection`
`$`	

3 - Protocole POP3

Protocole texte pour récupérer des mails.

RFC 1939 (1996, ...)

Protocole Ascii 7 bits, sur TCP

Utilise le port 110

Échange de questions-réponses avec le serveur.

Exemple (1/3)

`$ netcat` *adr-serveur* `110`	
	`+OK POP3 ready`
`USER` *user*	
	`+OK`
`PASS` *mauvais mdp*	
	`-ERR Invalid login or password`
`PASS` *bon mdp*	
	`-ERR Invalid command`
`USER` *user*	
	`+OK`
`PASS` *bon mdp*	
	`+OK Mailbox locked and ready`
`STAT`	
	`+OK 3 9251`

Exemple (2/3)

`LIST`	`+OK scan listing follows`
	`1 1747`
	`2 5808`
	`3 1696`
	`.`

`RETR 2`	`+OK Message follows`
	Entête et message
	`.`

`DELE 2`	`+OK message deleted`
`DELE 2`	`-ERR No such message`

Exemple (3/3)

LIST	
	+OK scan listing follows
	1 1747
	3 1696
	.
RSET	
	+OK
LIST	
	+OK scan listing follows
	1 1747
	2 5808
	3 1696
	.
DELE 2	
	+OK message deleted
QUIT	
	+OK
$	

4 - Protocole HTTP

Protocole texte, permettant de véhiculer texte, liens, images, sons, vidéos, etc

Protocole texte

Créateur de HTTP : Tim Berners-Lee

Trois versions :

0.9	implémentation originelle (1991)	abandonnée
1.0	RFC 1945 (1996)	obsolète mais utilisable
1.1	RFC 2616 (1999)	actuel

Protocole Ascii 7 bits, sur TCP

Principe : le client se connecte, envoie une requête ;
le serveur envoie une réponse puis déconnecte le client.

1.1 : possibilité de maintenir la connexion

Structure d'une requête

```
Méthode /URL HTTP/version <cr-lf>
Clé: valeur <cr-lf>
...
Clé: valeur <cr-lf>
<cr-lf>
Corps de la requête
```

Fin de l'entête : ligne vide (double `<cr-lf>`)

Retour chariot `<cr-lf>` : `\r\n` (`\n` toléré)

Liste de (clé,valeur) : dictionnaire de propriétés

Version : `1.0` ou `1.1`

Méthodes

- `GET` : demander une ressource
- `HEAD` : demander informations sur ressource
- `POST` : transmettre des informations
- `PUT` : transmettre par URL
- `PATCH` : modification partielle
- `OPTIONS` : obtenir les options de communication
- `CONNECT` : pour utiliser un proxy comme tunnel
- `TRACE` : écho de la requête pour diagnostic
- `DELETE` : pour supprimer une ressource
- ...

Propriétés

`Clé: valeur <cr-lf>`

- `Host` : site web demandé (si plusieurs pour même IP)
- `Referer` : source du lien, fournie par le client
- `User-Agent` : navigateur utilisé
- `Date` : date génération réponse
- `Server` : serveur utilisé (Apache, ...)
- `Content-Type` : type MIME (`image/png`, ...)
- `Content-Length` : taille en octets
- `Expires` : date d'obsolescence, pour gestion du cache
- `Last-Modified` : date dernière modification
- ...

Propriétés supplémentaires de HTTP/1.1

`Clé: valeur <cr-lf>`

- `Connection` : la maintenir ou non
- `Accept` : types MIME acceptés
- `Accept-Charset` : encodages acceptés
- `Transfer-Encoding` : type d'encodage
- ...

Méthode GET

Une requête HTTP pour la méthode `GET` est de la forme :

```
GET /fichier HTTP/version <cr-lf>
Clé: Valeur <cr-lf>
...
Clé: Valeur <cr-lf>
<cr-lf>
```

Pas de corps de requête après l'entête.

Si la version est `1.1`, il doit y avoir la propriété
`Host: adresse-serveur[:port]` (par défaut le port 80).

Réponse HTTP

Une réponse HTTP est de la forme :

```
HTTP/version code explication <cr-lf>
Clé: Valeur <cr-lf>
...
Clé: Valeur <cr-lf>
<cr-lf>
Corps de la réponse
```

Codes :	20x	Succès
	30x	Redirection
	40x	Erreur du client
	50x	Erreur du serveur

Exemple de TRACE (1/4)

`netcat -C` transforme les `\n` en `\r\n`

```
$ netcat -C sol.dil.univ-mrs.fr 80
TRACE /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 12 Nov 2014 10:07:04 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Content-Type: message/http

TRACE /index.html HTTP/1.0
```

Exemple de TRACE (2/4)

```
$ netcat -C sol.dil.univ-mrs.fr 80
TRACE index.html HTTP/1.0

HTTP/1.1 400 Bad Request
Date: Wed, 12 Nov 2014 10:07:29 GMT
Server: Apache/2.2.3 (CentOS)
Content-Length: 310
Connection: close
Content-Type: text/html; charset=iso-8859-1

<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a bad request...</p>
</body></html>
```

Exemple de TRACE (3/4)

```
$ netcat -C sol.dil.univ-mrs.fr 80
TRACE /index.html HTTP/1.1

HTTP/1.1 400 Bad Request
Date: Wed, 12 Nov 2014 10:08:06 GMT
Server: Apache/2.2.3 (CentOS)
Content-Length: 310
Connection: close
Content-Type: text/html; charset=iso-8859-1

<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a bad request...</p>
</body></html>
```

Exemple de TRACE (4/4)

```
$ netcat -C sol.dil.univ-mrs.fr 80
TRACE /index.html HTTP/1.1
Host: toto

HTTP/1.1 200 OK
Date: Wed, 12 Nov 2014 10:08:31 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Transfer-Encoding: chunked
Content-Type: message/http

2a
TRACE /index.html HTTP/1.1
Host: toto

0
```

Proxy

Entre un navigateur et un serveur web : à la fois client, serveur et base de données.

Côté navigateur :

- Accélère l'obtention de fichiers stockés dans la BD du proxy
- Anonymise le client

Côté serveur :

- Réduit le trafic

Coté proxy :

- Permet de filtrer les URLs
- Permet de loguer le trafic