Protocoles de transport

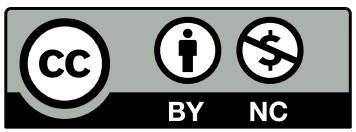
Pierre David
pda@unistra.fr

Université de Strasbourg – Master CSMI

2023 – 2024

Plan

Licence d'utilisation



Plan

TCP/UDP - Présentation

IP est un protocole qui achemine les datagrammes :

- d'une machine à une autre
- le mieux possible (*best effort*)

$\Rightarrow$ besoin d'une couche *transport* au dessus de IP.

TCP/UDP - Présentation

TCP (*Transmission Control Protocol*) est un protocole basé sur IP (encapsulé dans des datagrammes IP) qui :

- assure une *connexion*
- utilise la notion de port (16 bits) pour faire communiquer deux applications sur des machines
- contrôle les erreurs pour assurer une transmission fiable

TCP/UDP - Présentation

UDP (*User Datagram Protocol*) est également un protocole basé sur IP qui :

- fonctionne en mode *non* connecté
- utilise la notion de port (16 bits) pour faire communiquer deux applications sur des machines
- n'assure pas la fiabilité de la transmission

$\Rightarrow$ UDP = simplicité de IP accessible par les applications.

Plan

UDP

Caractéristiques :

- transport non fiable
- sans connexion
- ordre des messages non respecté

Non fiable car :

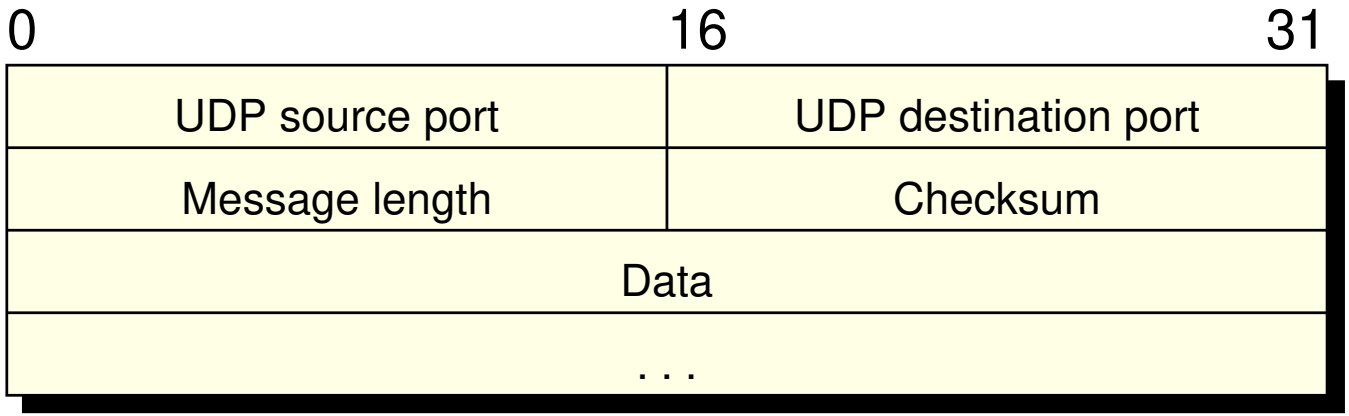
- IP non fiable
- les messages sont bufferisés par le système d'exploitation
  $\Rightarrow$ si l'application ou le système ne sont pas assez rapides, des messages sont perdus

UDP

Exemples de ports UDP :

Port	Nom	Signification
53	domain	serveur de noms
67	bootps	serveur BOOTP/DHCP
68	bootpc	client BOOTP/DHCP
69	tftp	trivial file transfer
123	ntp	network time protocol
161	snmp	gestion de réseau
162	snmp-trap	alertes SNMP
513	who	rwho

UDP


0
16
31
UDP source port
UDP destination port
Message length
Checksum
Data
. . .

Plan

TCP

Caractéristiques :

- transport fiable
- circuit virtuel
- transferts bufferisés
- flux d'information non structuré
- full duplex

Une connexion TCP est identifiée par :

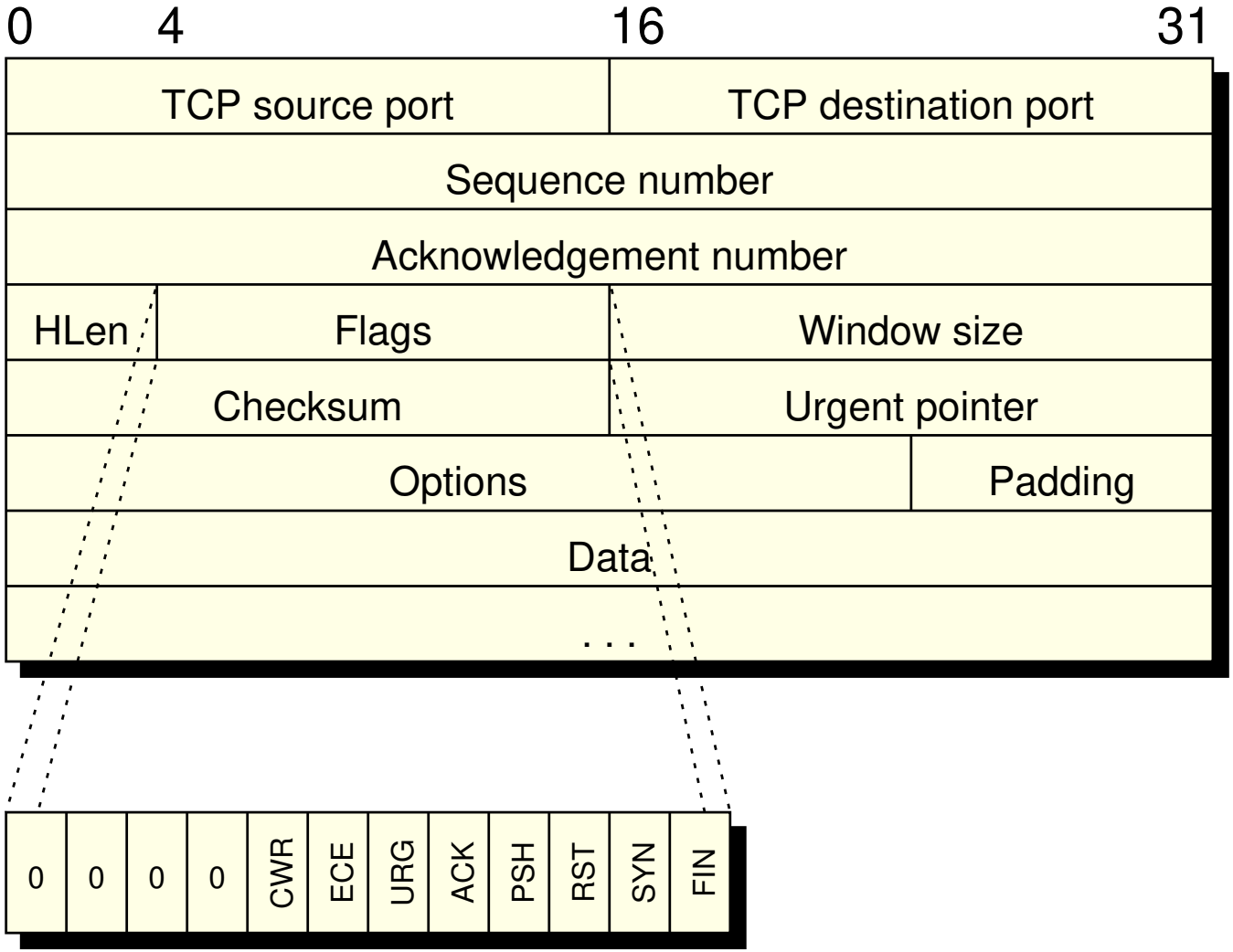
- l'adresse IP et le port du programme source
- l'adresse IP et le port du programme destination

TCP

Exemples de ports TCP :

Port	Nom	Signification
21	ftp	transfert de fichiers
22	ssh	secure shell
23	telnet	connexion à distance
25	smtp	courrier électronique
53	domain	serveur de noms
79	finger	finger
143	imap3	lecture de mail
515	printer	impression déportée
631	ipp	internet printing protocol
6000	X11	X Window Version 11

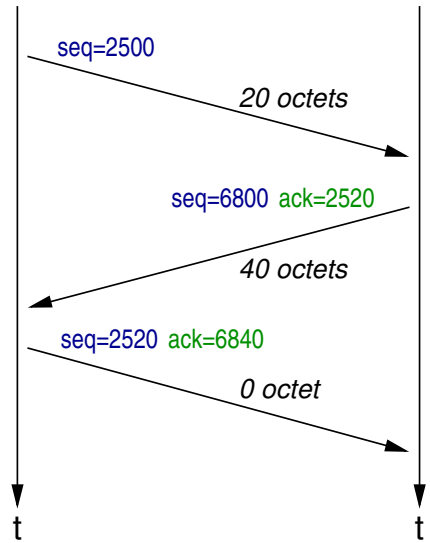
TCP


0
4
16
31
TCP source port
TCP destination port
Sequence number
Acknowledgement number
HLen
Flags
Window size
Checksum
Urgent pointer
Options
Padding
Data
. . .
0
0
0
0
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN

TCP

Fiabilité : lorsque A envoie un segment à B, il suffit que A attende l'accusé de réception avant d'envoyer le segment suivant
$\Rightarrow$ numéros de séquence et d'acquittement :

- valeur initiale aléatoire
- numéro de séquence = offset (relatif à la valeur initiale) du premier octet du segment dans le flux de données
- numéro d'acquittement = numéro de séquence du prochain segment attendu

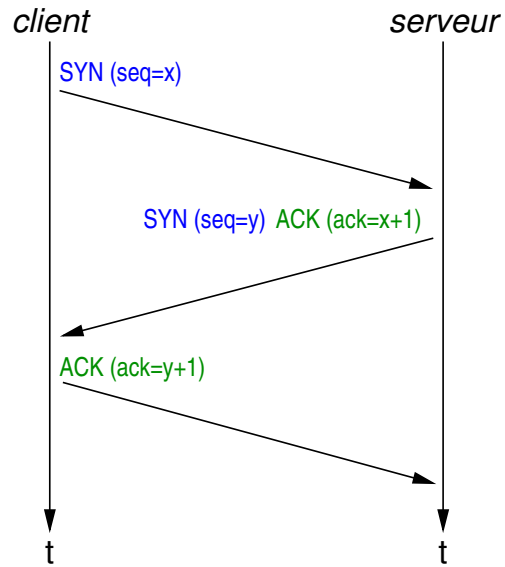


TCP

Problème : comment échanger les numéros ?
$\Rightarrow$ établissement de la connexion

Ouverture de connexion :

- ouverture *passive* : accepter et attendre les connexions (serveur)
- ouverture *active* : contacter l'application distante et s'y connecter (client)

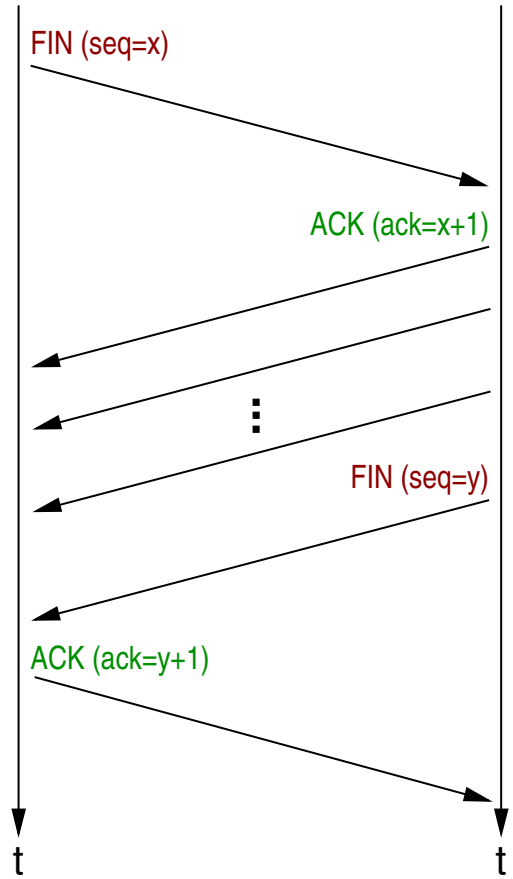


Utilisation des flags SYN et ACK pour ouvrir la connexion et échanger les numéros de séquence respectifs

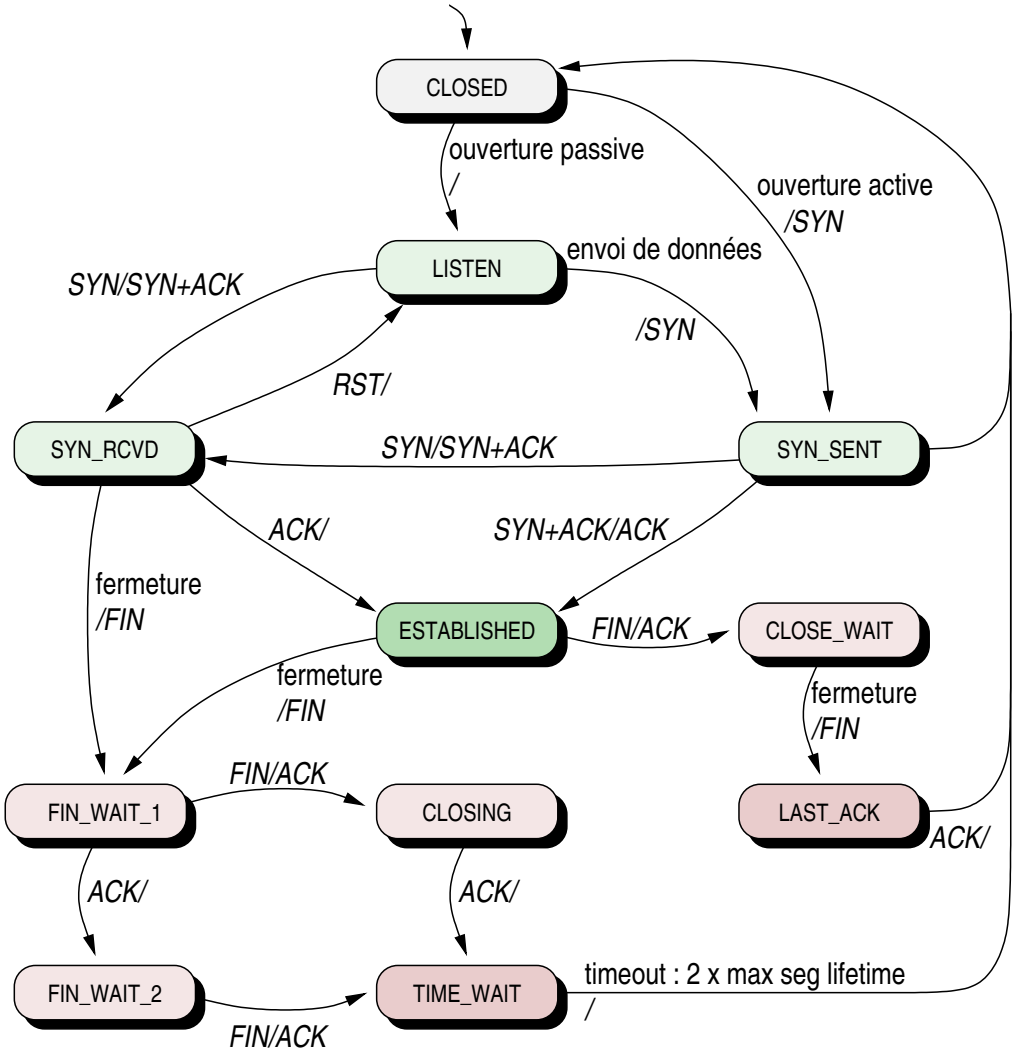
TCP

Fermeture de la connexion :

- nécessite les fermetures des deux parties
- la partie n'ayant pas fermé la connexion peut continuer à envoyer des données



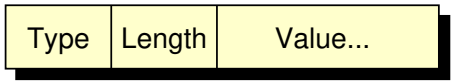
TCP – Diagramme d'états



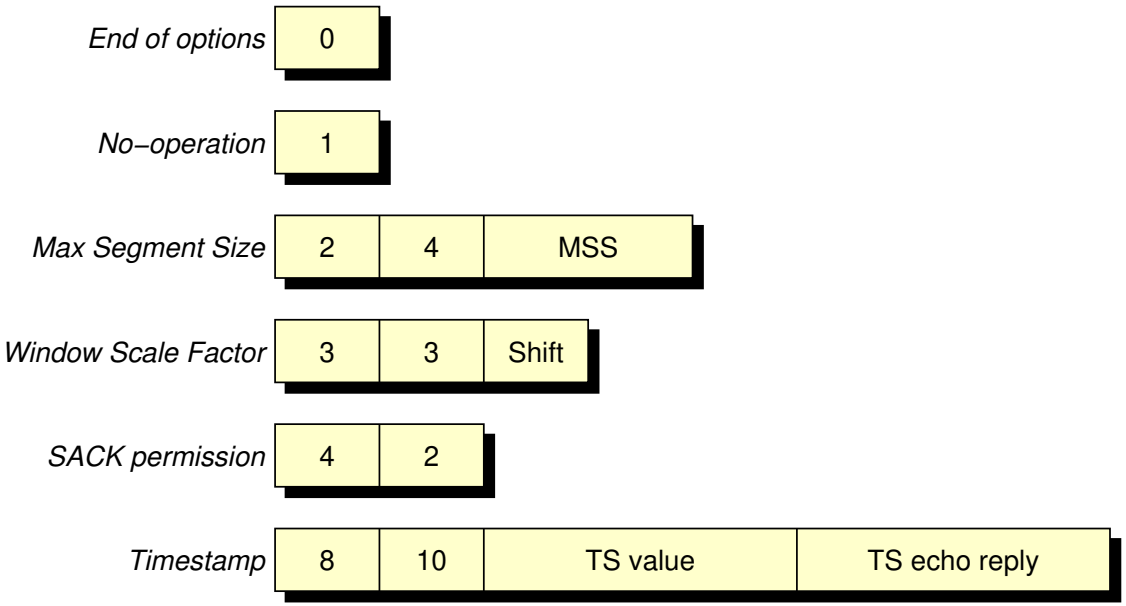
Note : *a*/*b* signifie « réception de *a*, émission de *b* »

TCP

Format des options de l'en-tête TCP :



Exemples d'options :



TCP

Option MSS = *Maximum Segment Size*

Quelle valeur choisir ?

- doit être adapté aux buffers de réception et d'émission
- ne doit pas être trop petit
  $\Rightarrow$ minimiser l'overhead
- ne doit pas être trop grand
  $\Rightarrow$ perte d'un fragment nécessite la retransmission de tout le datagramme

Initialisation :

- annonce par chaque partie à l'ouverture de connexion
- initialisé par défaut à la valeur du Path-MTU

TCP

Optimisations de TCP pour les grands réseaux :

- fenêtres mobiles (*sliding windows*)
- algorithme de Nagle
- acquittements retardés (*delayed acks*)
- démarrage lent (*slow start*)
- évitement de congestion (*congestion avoidance*)
- retransmission rapide (*fast retransmit*)

TCP

Fiabilité : lorsque A envoie un segment à B, il suffit que A attende l'accusé de réception avant d'envoyer le segment suivant
$\Rightarrow$ algorithme de type *stop-and-wait*

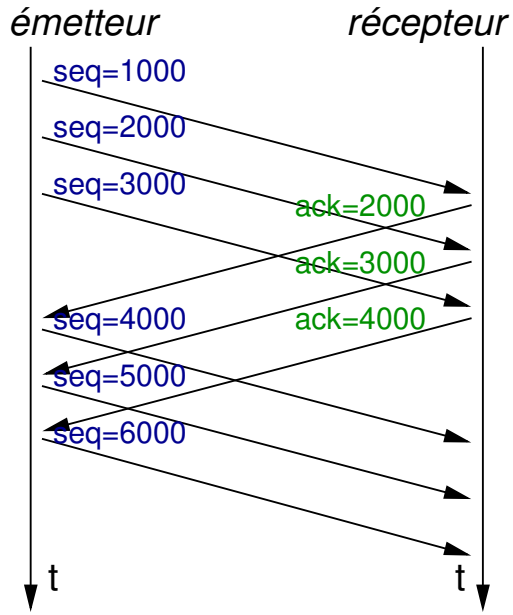
Problème : mécanisme trop lent (entre 100 et 500 ms pour traverser l'Atlantique) car synchronisation à chaque segment

Solution : *Sliding Windows* (fenêtre mobile)

TCP

Principe avec une fenêtre mobile de taille 3000 octets :

1. l'émetteur envoie les 3000 premiers octets (3 segments de taille 1000)
2. quand l'émetteur reçoit l'acquittement des 1000 premiers octets (premier segment), il peut envoyer 1000 octets supplémentaires (quatrième segment)



TCP

Comment déterminer la taille des fenêtres ?
Principes de la taille de fenêtre :

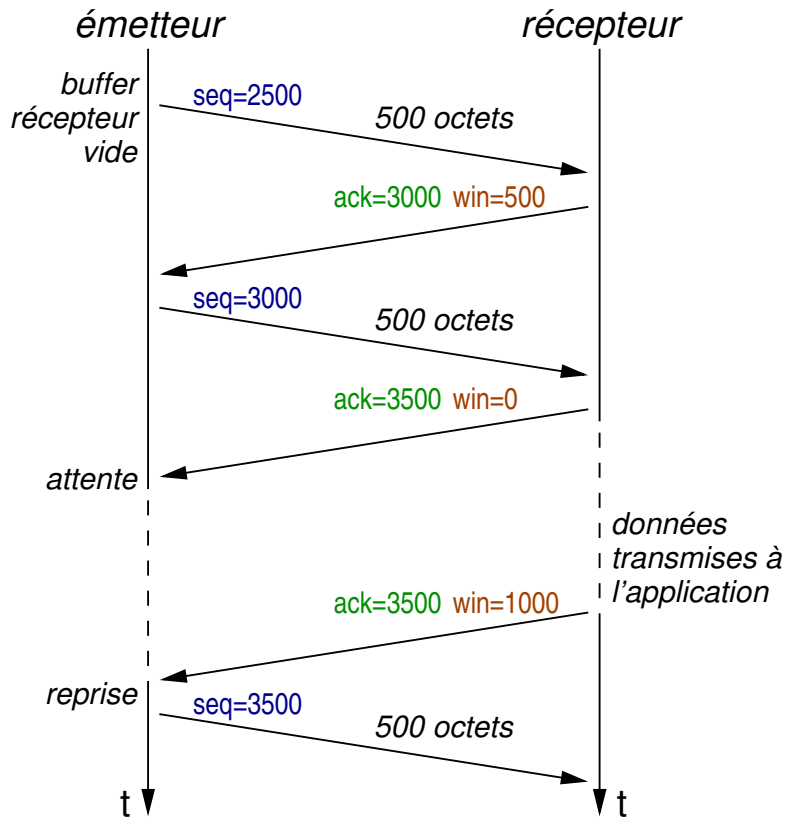
- mesurée en octets
- place disponible dans la mémoire du récepteur
- c'est au récepteur d'annoncer la taille de la fenêtre disponible
- champ « Window Size » de l'en-tête TCP

Attention : ne pas confondre numéro d'acquittement et taille de fenêtre

TCP

Exemple :

- taille de fenêtre initiale = 1000 octets
- taille maximum de segment = 500 octets



TCP

Problème : taille de fenêtre sur 16 bits
$\Rightarrow$ limite = 65535 octets
$\Rightarrow$ bien peu pour des connexions à plusieurs Gb/s !

Option *Window Scale Option* lors de la connexion TCP :
si une des parties donne une valeur n, toutes ses annonces de taille de fenêtre seront à multiplier par 2^n

TCP – Mesure du RTT

Hauts débits + délais importants $\Rightarrow$ numéros de séquence rapidement épuisés
$\Rightarrow$ comment savoir si un segment est déjà reçu ou non ?

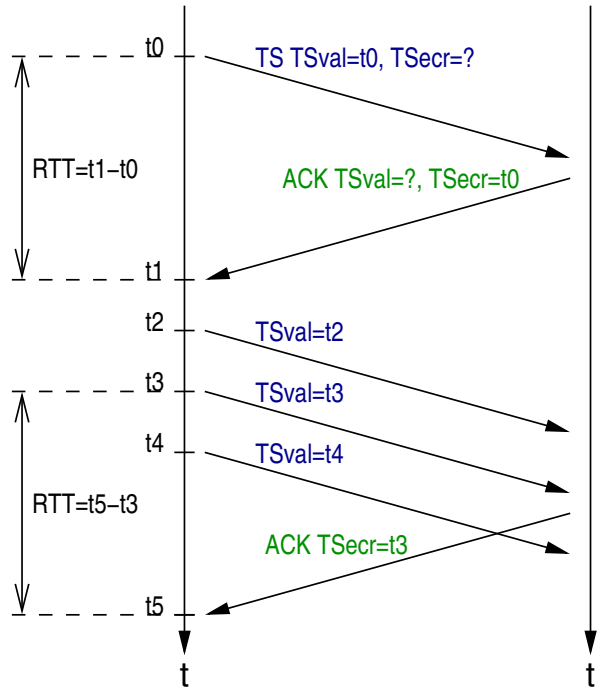
Exemple : avec un RTT de 500 ms et un débit de 1 Gb/s, les 2^{32} numéros possibles sont épuisés en environ 2 secondes

$\Rightarrow$ mesurer le RTT moyen sur un lien
$\Rightarrow$ calculer le RTO (*retransmission timeout*) minimal

TCP – Mesure du RTT

Option « Timestamp » pour mesurer le RTT sur une machine

- l'émetteur place une estampille temporelle dans TSval
- le récepteur renvoie TSval dans TSecr (*echo reply*)
- l'émetteur calcule le RTT
- en cas d'acquittement groupé, le récepteur renvoie l'estampille du dernier message arrivé



(en pratique, les t_i dans l'option Timestamp ne sont pas directement l'horloge de l'émetteur, pour ne pas donner d'indication à un éventuel pirate $\Rightarrow$ facteur multiplicatif secret par connexion)

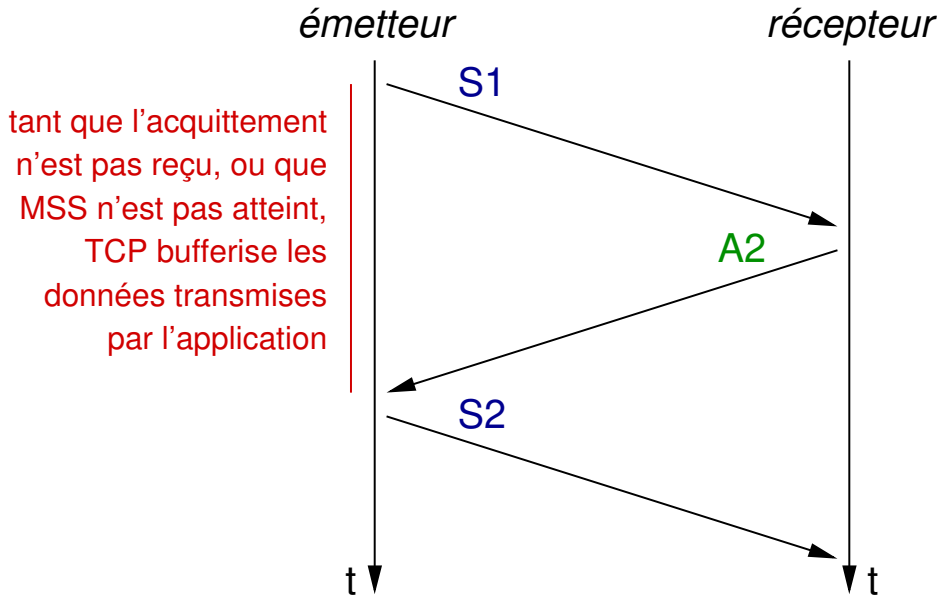
TCP – Algorithme de Nagle

Problème : certains protocoles comme TELNET envoient un caractère par segment (*tinygrams*)
$\Rightarrow$ gâchis de ressources réseau
$\Rightarrow$ congestion sur des réseaux lents

Solution :

- pas plus d'un « petit segment » non acquitté
- si on a d'autres petits segments à envoyer, on attend l'acquittement du dernier petit segment
  $\Rightarrow$ *bufferisation* automatique
- « petit segment » = plus petit que MSS (taille maximum d'un segment)

TCP – Algorithme de Nagle



Bénéfices :

- algorithme auto-adaptatif
- les courtes données n'encombrent pas le réseau
- les grandes données sont envoyées rapidement

Contre-exemples : X-Window (déplacement de souris)

$\Rightarrow$ désactivation : `setsockopt(..TCP_NODELAY..)`

TCP

Problème : les aquittements encombrent le réseau
Solutions :

- *piggyback* : inclure les acquittements dans des segments contenant des données
- *delayed ack* : attendre (200 à 500 ms, ou le deuxième segment de données par exemple) et ne renvoyer éventuellement qu'un seul acquittement au lieu de plusieurs
  $\Rightarrow$ éventuellement avec un segment de données

TCP

Problème : si l'émetteur émet à vitesse maximum
$\Rightarrow$ saturation des liaisons
$\Rightarrow$ bufferisation par les routeurs intermédiaires
$\Rightarrow$ écroulement du réseau

Solution :

- l'émetteur conserve pour chaque connexion :
  - une « fenêtre de congestion »
  - une copie de la fenêtre de réception annoncée par le récepteur
- l'émetteur n'envoie que le minimum des deux fenêtres
- la fenêtre de congestion sert à ajuster le flux de l'émetteur en fonction des congestions observées dans le réseau.

TCP

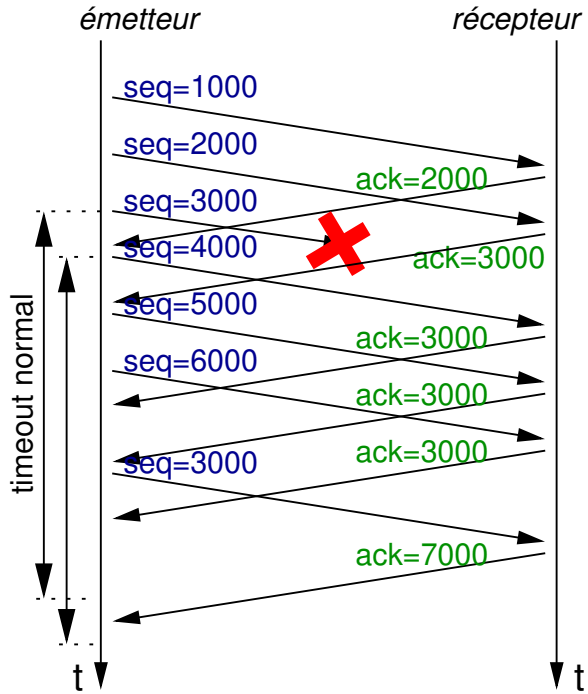
Traitement des congestions :

- démarrer lentement (algorithme *slow-start*) :
  la fenêtre démarre avec une taille de 1 segment, elle augmente de 1 à chaque retour d'acquittement
- réagir rapidement en cas de congestion :
  1 segment perdu $\Rightarrow$ taille de fenêtre divisée par 2
  $\Rightarrow$ chute du débit à la première congestion
- anticiper les congestions (*congestion avoidance*)
  au delà d'un seuil (la moitié de la fenêtre de congestion avant la dernière congestion), l'augmentation de débit ne se fait que de 1 lorsque tous les acquittements d'une fenêtre sont reçus
  $\Rightarrow$ augmentation ralentie

TCP – Fast retransmit

Idée : accélérer les retransmissions sans attendre l'expiration du timeout

Principe : au bout de 3 acquittements identiques, retransmettre le segment demandé sans attendre l'expiration du timeout



TCP

Problème : transporter des données *urgentes* (signaux, interruptions, conditions d'erreur, etc.)

Solution : *out of band data*

Implémentation :

- flag *URG* : signale des données urgentes
- *Urgent Pointer* : localise les données dans le flux

L'utilisateur est prévenu par un mécanisme dépendant du système d'exploitation (ex : signaux sous Unix)

Plan

Usurpation d'identité

L'association entre une adresse IP et une adresse MAC n'est par nature pas sécurisée

Il est possible de « forger » un datagramme IP avec une adresse IP source usurpée

$\Rightarrow$ attaques par injection de données malicieuses

Usurpation d'adresse IP (*IP Spoofing*)

Certains protocoles utilisent l'adresse IP source pour autoriser une communication (faiblesse d'authentification)

Exemples : rlogin, NFS, RPC, services « protégés » par *tcp-wrappers* ou équivalent, etc.

Attaque :

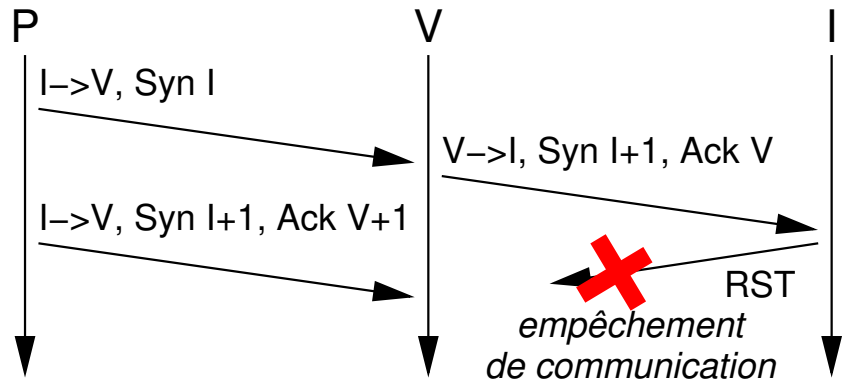
- forger l'adresse source d'un datagramme IP
- la réponse est envoyée à l'adresse usurpée $\Rightarrow$ souvent utilisé avec « vol de port »

Prévention :

- ne pas authentifier sur la simple adresse IP
- filtrer les adresses internes sur le port externe du routeur ou du pare-feu

TCP – Usurpation d'identité

Usurpation d'adresse IP avec TCP : un pirate essaye d'envoyer des données à une victime en usurpant l'adresse IP d'un innocent.



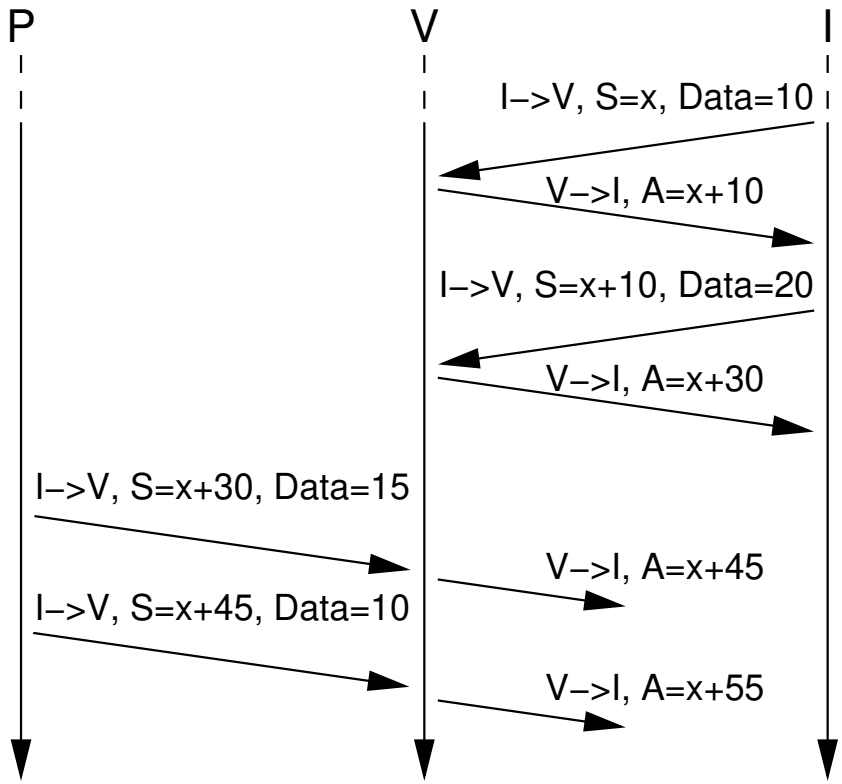
Difficultés :

- deviner le numéro de séquence initial de la victime
- empêcher l'innocent de perturber la conversation

$\Rightarrow$ attaque de type « émission seulement »
(ex: mail toto@labas.fr < /etc/passwd)

TCP – Vol de session (*session hijacking*)

Une fois la connexion établie (y compris l'authentification éventuelle) entre le client et le serveur, le pirate usurpe l'adresse de l'innocent pour envoyer des données malicieuses à la victime



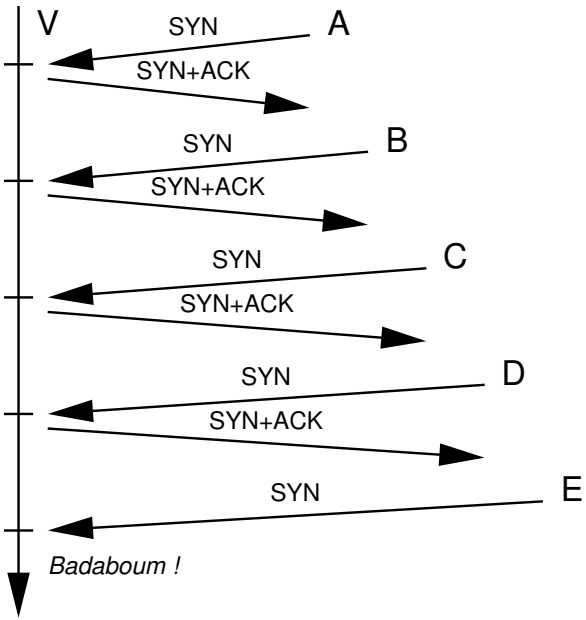
TCP – Vol de session (suite)

Difficultés :

- deviner le numéro de séquence de l'innocent
- empêcher l'innocent d'émettre des paquets (vol de port ou duperie de résolution d'adresse), ou gérer la tempête de demandes de réémissions

$\Rightarrow$ des logiciels rendent cette exploitation simple : T-Sight (Windows), Hunt (Linux), Juggernaut (Linux)

TCP – Syn flooding



Attaque de type Syn Flooding :

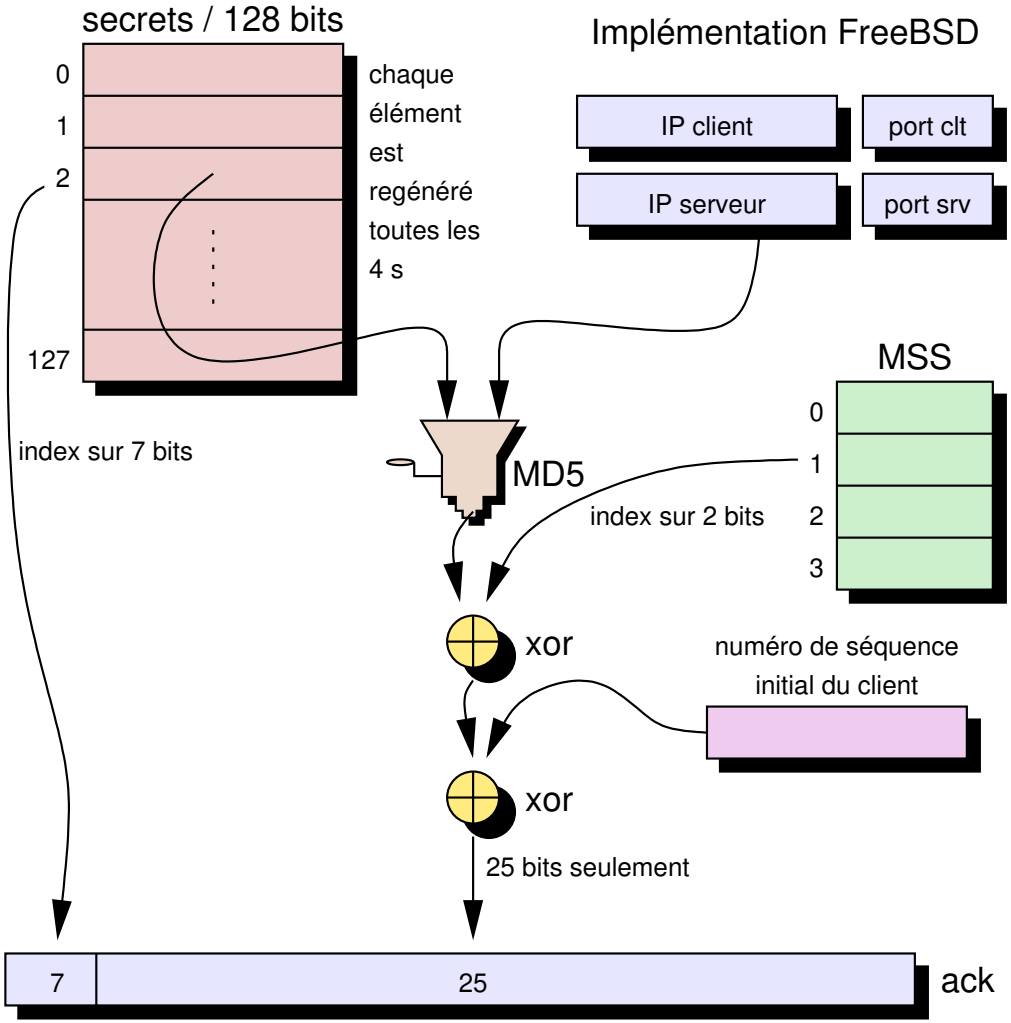
- sur réception d'un SYN, le serveur réserve un « état »
- le pirate envoie des SYN avec des adresses IP usurpées
- la victime répond avec des SYN+ACK aux adresses sources
- au bout d'un moment, la table des connexions entrantes de la victime est saturée

$\Rightarrow$ attaque de type « déni de service »

TCP – Syn flooding

Solution : supprimer l'état dans la mémoire du serveur TCP

$\Rightarrow$ l'état est dans le numéro d'acquittement initial



Plan

Filtrage au niveau IP

Filtrage simple (par exemple sur un routeur) à l'aide d'un ensemble de règles :

Interface d'entrée	Adresse IP source	Adresse IP destination	Action
re0	198.18.0.0/15	198.51.100.0/24	allow
re0	0.0.0.0/0	198.51.100.1/32	allow
re0	0.0.0.0/0	198.51.100.0/24	deny

Filtrage au niveau IP

Plusieurs approches possibles en fonction de l'implémentation :

- les règles sont examinées dans l'ordre avec arrêt sur le premier (ou le dernier) cas trouvé
- ou alors la spécification la plus précise est seule examinée
- action par défaut : « allow » ou « deny »
- interface = interface d'entrée ou de sortie
- certaines implémentations permettent d'inclure des éléments complémentaires du datagramme IP (flags, type de service, protocole, options)

Dans l'exemple précédent : arrêt sur la première règle qui correspond aux deux adresses.

Filtrage au niveau IP

Exemple (syntaxe IOS Cisco) :

```
access-list 123 permit ip 198.18.0.0 0.0.127.255
                           198.51.100.0 0.0.0.255
access-list 123 permit ip any 198.51.100.1
access-list 123 deny   ip any 198.51.100.0 0.0.0.255

interface GigaEthernet5/23
  ip access-group 123 in
```

Filtrage au niveau TCP/UDP

Ajout de triplets <protocole, port source, port destination> :

Interface d'entrée	Adresse IP source	Port src	Adresse IP destination	Port dst	Proto	Action
eth0	198.18.0.0/15	any	198.51.100.0/24	22	tcp	allow
eth0	0.0.0.0/0	any	198.51.100.1/32	80	tcp	allow
eth0	0.0.0.0/0	any	198.51.100.0/24	any	tcp	deny
eth0	0.0.0.0/0	any	198.51.100.2/24	53	udp	allow
eth0	0.0.0.0/0	any	198.51.100.2/24	any	udp	deny
eth0	0.0.0.0/0	-	198.51.100.0/24	-	icmp	deny

Filtrage au niveau TCP/UDP

Selon les implémentations : prise en compte des flags TCP

Exemple : autoriser n'importe quelle connexion sortante SSH vers l'extérieur, et accepter les paquets en retour

- depuis le réseau interne vers l'extérieur : accepter tous les segments dont le port source = 22
- depuis l'extérieur vers le réseau interne : n'accepter que les segments TCP dont le port destination = 22 et les flags TCP sont soit ACK, soit RST

Filtrage délicat à réaliser à grande échelle (i.e. beaucoup de protocoles, de réseaux et/ou d'interfaces)

Filtrage au niveau TCP/UDP

Jusqu'ici : filtrage *sans état*
$\Rightarrow$ filtrage réalisé avec les seules informations du paquet

Mais :

- difficulté de créer des politiques de filtrage complexes
- vulnérabilité potentielle à des attaques avec des datagrammes forgés

D'où le filtrage *à états* :

- mémoriser un état pour des connexions TCP (voire des échanges sans connexion comme UDP ou ICMP)
  Exemple d'état pour TCP :
  <IP S, port S, IP D, port D, statut, date dernier paquet>
- nécessité d'expirer les états
  $\Rightarrow$ envoi de messages *keepalive* (application ou pare-feu)
- plus complexe pour le routeur/pare-feu

Réseaux privés

Raréfaction des adresses IPv4 $\Rightarrow$ réseaux *privés* (RFC 1918)

10.0.0.0/8
172.16.0.0/12
192.168.0.0/16

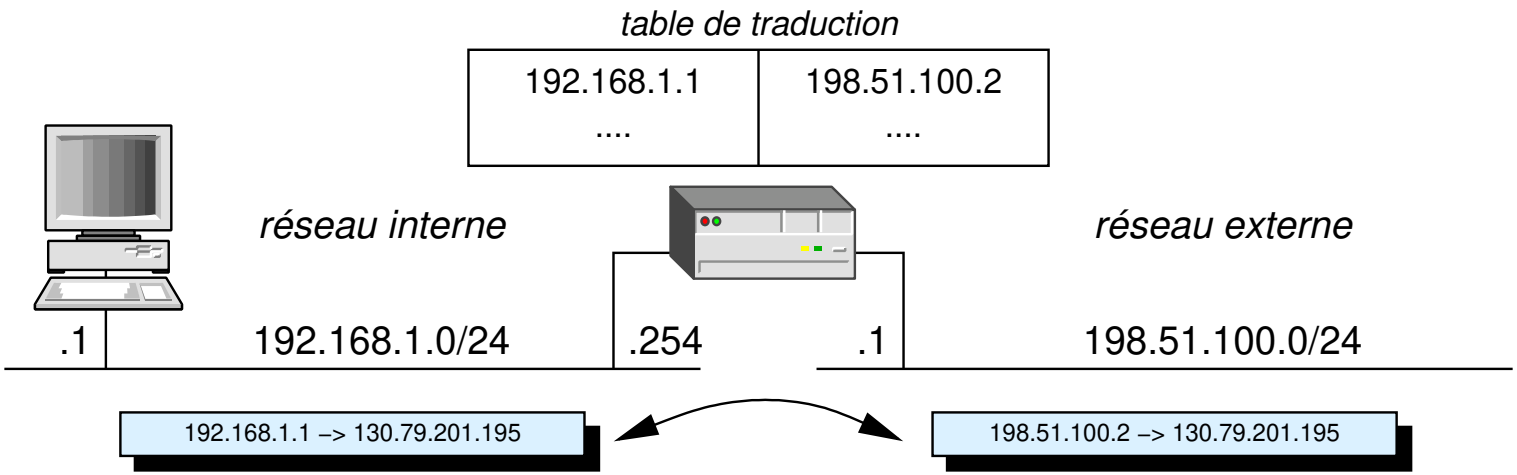
Utilisation privée :

- réseaux non connectés à Internet
  exemples : robots d'une chaîne de montage, distributeurs de billets, etc.
- réseaux non routés sur Internet
- réseaux à filtrer en bordure de réseau privé (pour ne pas polluer l'Internet avec ces messages)

$\Rightarrow$ de plus en plus utilisés avec de la traduction d'adresses

Network Address Translation (NAT)

Traduction d'adresse IP sur un routeur/pare-feu, à l'aide d'une table de traduction « 1 pour 1 » :



Rupture du paradigme « connectivité de bout en bout » :

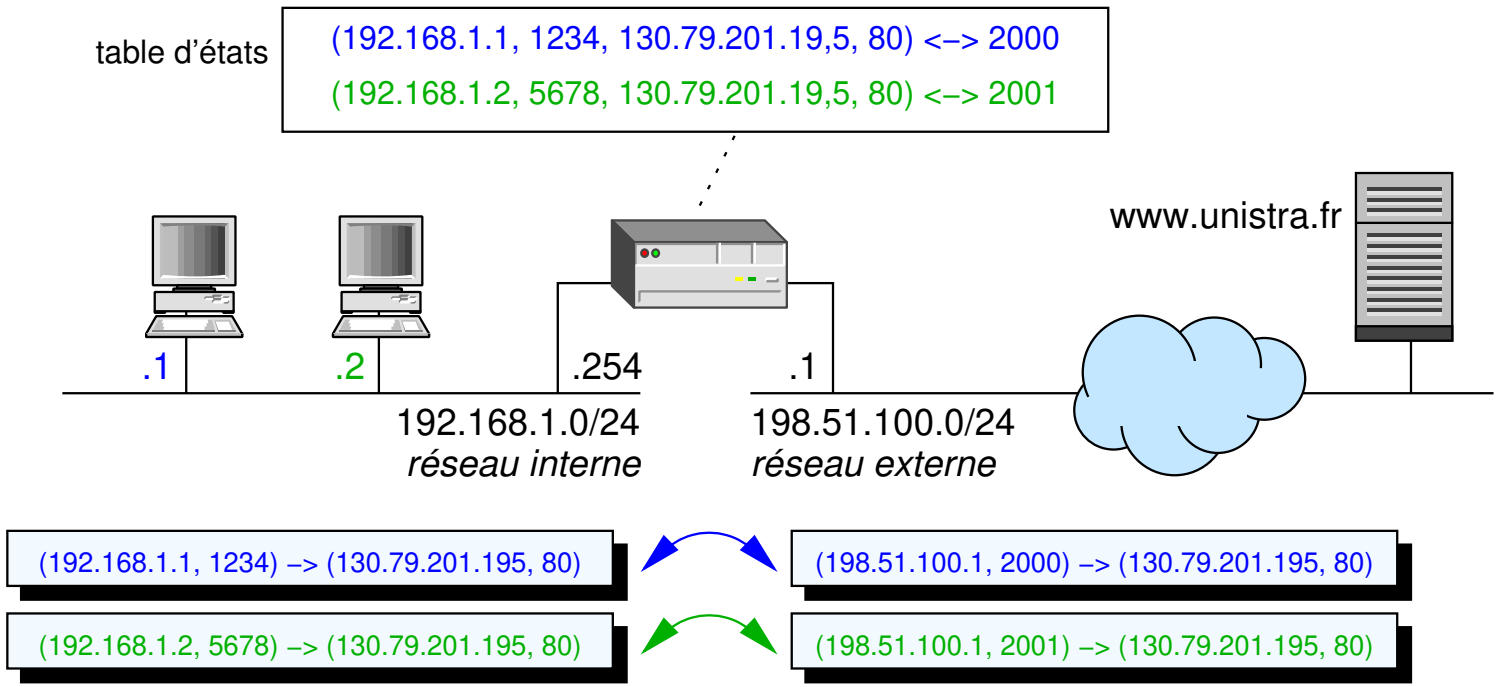
- protocoles supérieurs citant des adresses IP (ex: FTP)
- chiffrement IPSec (utilise les adresses IP)
- etc.

Network Address Translation (NAT)

- NAT statique : table de correspondance statique entre les adresses internes et les adresses externes
  $\Rightarrow$ peu économe en adresses IP
- NAT dynamique : la première connexion sortante réserve la première adresse IP externe disponible
  $\Rightarrow$ peu utilisée en pratique

Network Address Translation (NAT)

Traduction <adresse, port> (NAPT) pour connexions sortantes :



- modèle quasi-universel pour le résidentiel et le GSM
- traduction possible en entrée (table statique de redirection)
- double-NAT, triple-NAT, carrier-grade NAT, etc.

Network Address Translation (NAT) – Bilan

Avantages :

- économie d'adresses IP
- refus des connexions entrantes $\Rightarrow$ semblant de sécurité

Inconvénients :

- faux sentiment de sécurité
- rupture du paradigme « connectivité de bout en bout »
  - protocoles supérieurs citant des adresses IP ou des ports (H.323, FTP, DNS, etc.)
  - protocoles où le serveur ouvre une nouvelle connexion vers le client (ex: FTP, rlogin)
- ICMP
- implémentations défaillantes (fragmentation, extensions « récentes » de protocoles, etc.)
- IPv6