

tutoJRES Métrologie

Université Paris 5

3 Octobre 2008

Evolution des protocoles de transport réseau

Luc.Saccavini@inria.fr et groupe métrologie

Plan de l'exposé

- ◆ Les protocoles de transport UDP et TCP
- ◆ Les évolutions de TCP
- ◆ Les nouveaux protocoles de transport DCCP et SCTP
- ◆ (RTP/RTCP, RTSP)

Rappels sur IP

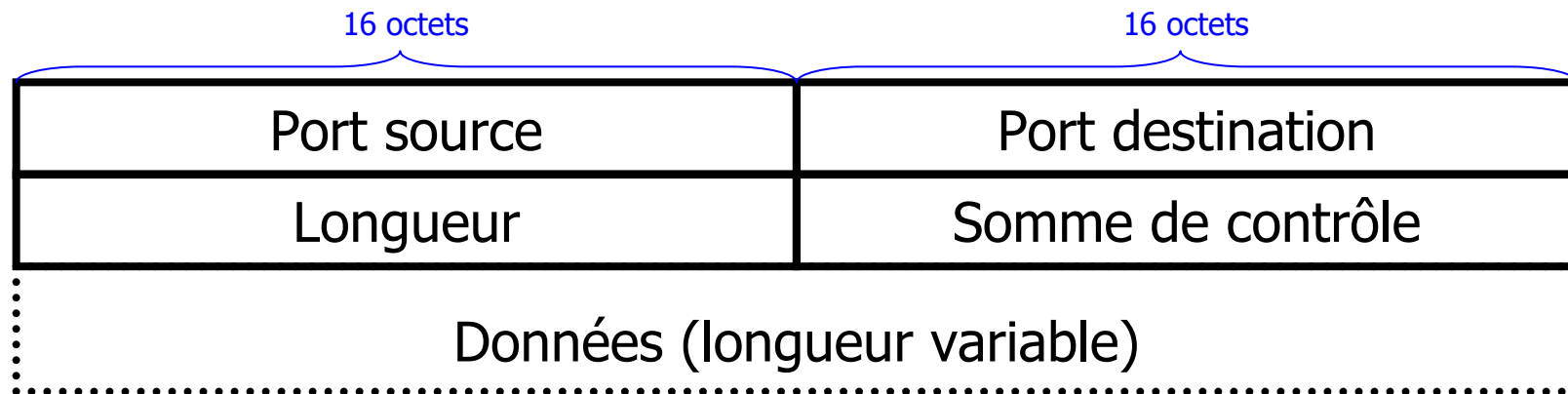
- ◆ IP : *Internet Protocol*
- ◆ IP est un protocole d'acheminement de paquets entre deux extrémités (interfaces de machines) du réseau
- ◆ Il est robuste : pas de session, routage par la destination
- ◆ Efficace : coût de 1,3% en IPv4 (2,7% en IPv6)
- ◆ Non fiable : la perte de paquets est même nécessaire pour éliminer les boucles (TTL<64)
- ◆ Ne prend pas en compte la notion de service (N° de port), de session...
- ◆ C'est le rôle de protocoles au dessus d'IP comme UDP, TCP, et de leurs évolutions DDCP, SCTP...

Le protocole UDP

- ◆ UDP : *User Datagram Protocol*
- ◆ Procure l'identification d'un flux de données (lié à un service) à partir du quadruplet
 - Adresse IP de la source (32bits IPv4, 128 bits en IPv6)
 - Port de la source (16bits)
 - Adresse IP de la destination
 - Port de la destination
- ◆ Très efficace (coût 0,55%), **MAIS** peut utiliser toute la bande passante (risque de congestion)
- ◆ N'assure pas la fiabilité de la transmission

Le protocole UDP

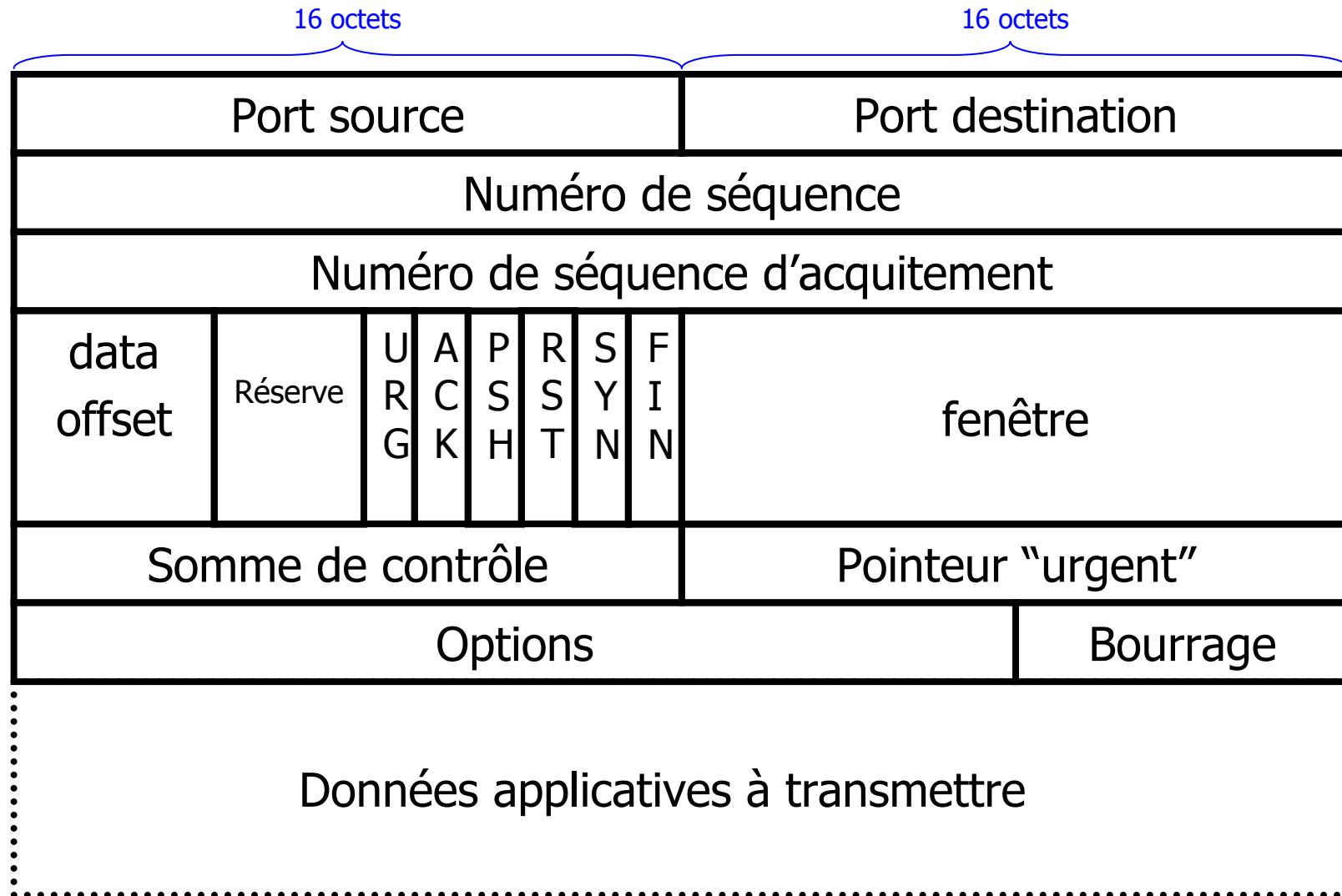
- ◆ Pas de notion de session, ni d'acquittement des paquets reçus
- ◆ Exemples d'utilisation
 - Echanges courts : DHCP, DNS, RIP, SNMP
 - Flux isochrones, à débit constant ou non : voix, vidéo
- ◆ L'entête du paquet UDP



Le protocole TCP

- ◆ TCP : *Transmission Control Protocol*
- ◆ Identifie le flux de données comme UDP
 - Adresses IP et ports de la source et de la destination
- ◆ Assure une transmission fiable
- ◆ Assure un contrôle de congestion
- ◆ Connexion point à point (unicast) en mode duplex

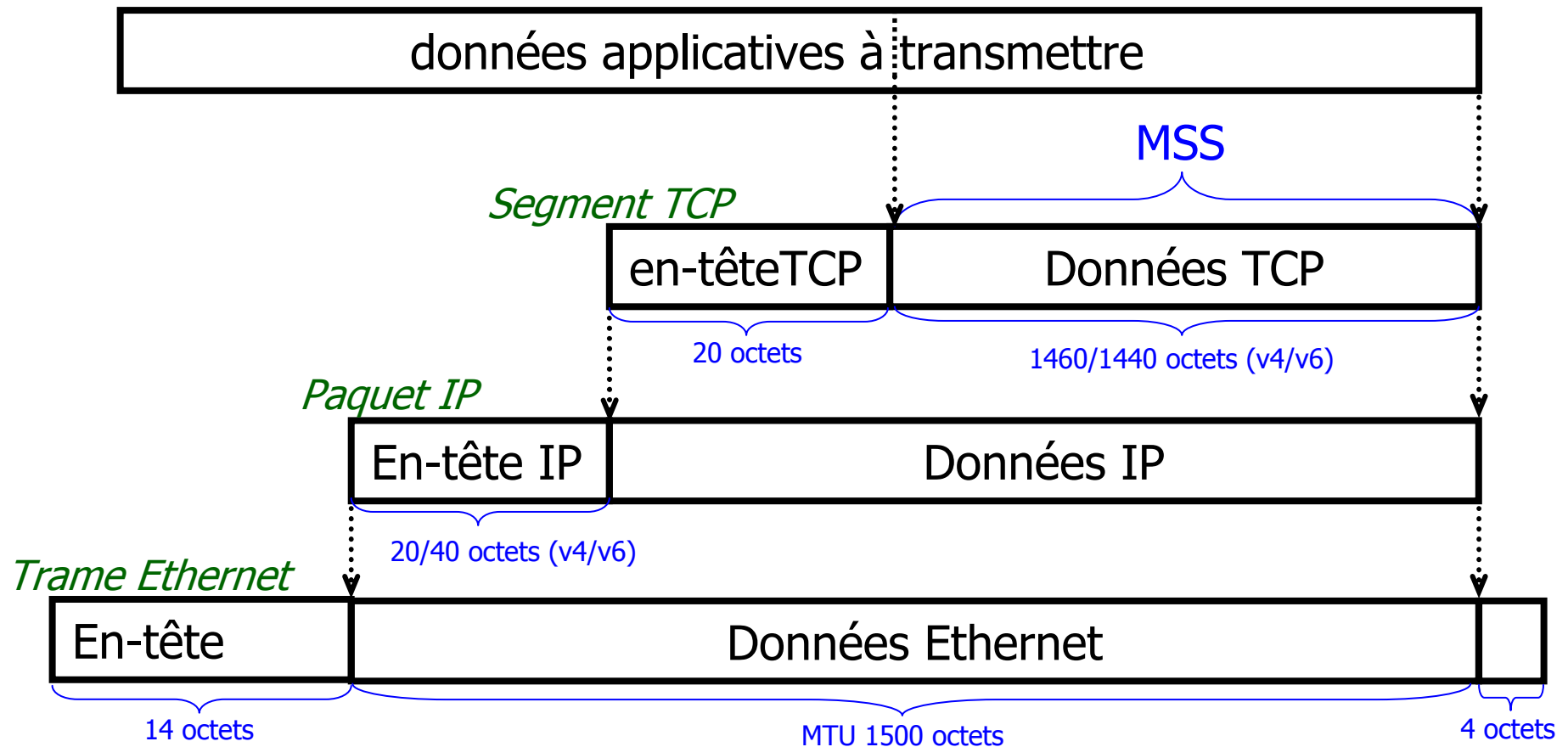
L'en tête TCP



Les options de TCP

- ◆ Ces options correspondent à des évolutions de TCP
- ◆ *Window Scale*
 - Elle permet de s'affranchir de la limitation de la taille de la fenêtre due à la longueur de 16bits du champ Fenêtre dans l'en-tête TCP
 - valeur maxi autorisée : 14 (=> fenêtre de réception jusqu'à 1Go)
- ◆ *TimeStamps*
 - permet un horodatage des segments, et un calcul plus précis du RTT
- ◆ *Selective Acknowledgments (SACK)*
 - permet d'informer l'émetteur qu'un segment est arrivé alors que des segments précédents ne le sont pas encore (désequencement, retard ou perte d'un segment)

L'encapsulation TCP dans Ethernet



Débit maxi utile 94,92Mb/s (IPv4) 93,62Mb/s (IPv6), avec MTU=1500
99,13Mb/s (IPv4) 98,91Mb/s (IPv6), avec MTU=9000

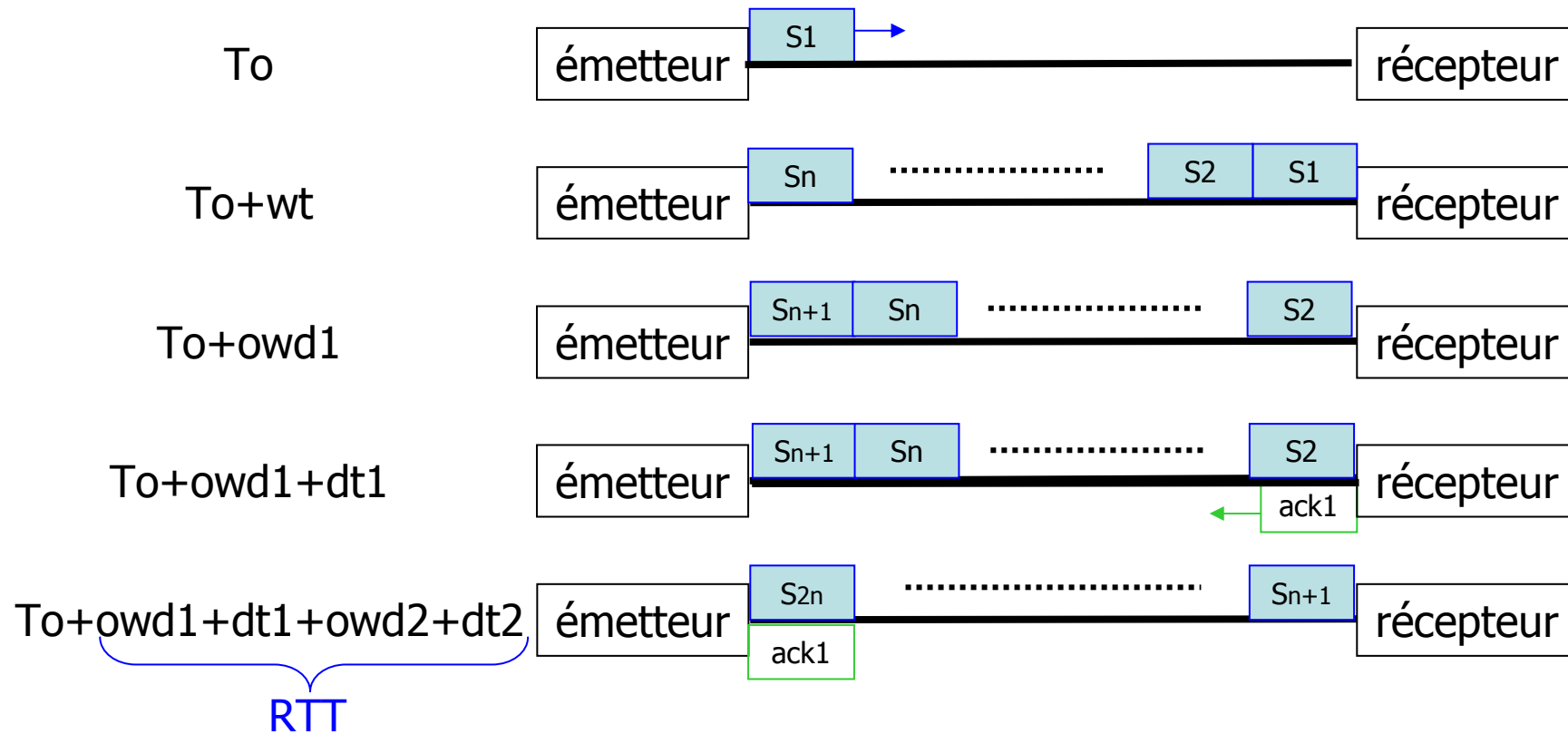
Contrôle de flux TCP

- ◆ Répond à plusieurs objectifs contradictoires
 - Utiliser toute la bande passante disponible
 - Sans provoquer de congestion (\Rightarrow perte de paquet)
 - De façon équitable (\Rightarrow répartition de la bande passante égalitaire entre toutes les sessions TCP partageant un même lien)
- ◆ Coté récepteur
 - Pilote l'émetteur en fonction de la demande de la couche applicative
 - Au moyen de la fenêtre de réception (FR)
- ◆ Coté émetteur
 - Cherche à émettre le maximum de segments autorisés (dans la FR)
 - Détecte la congestion par la perte de segments
 - Adapte la fenêtre d'émission (FE) en fonction de la situation

Contrôle de flux TCP (récepteur)

- ◆ À l'initialisation de la session TCP
 - le récepteur ouvre une Fenêtre de Réception (FR) autorisant l'émetteur à envoyer le nombre correspondant d'octets (valeur du champ *window*)
- ◆ À chaque segment TCP reçu, le récepteur
 - envoie à l'émetteur un accusé de réception (ACK) pour ce segment
 - autorise l'émetteur à envoyer FR octets
 - Stocke les données (et éventuellement les réordonne) pour qu'elles puissent être transmises à la couche applicative

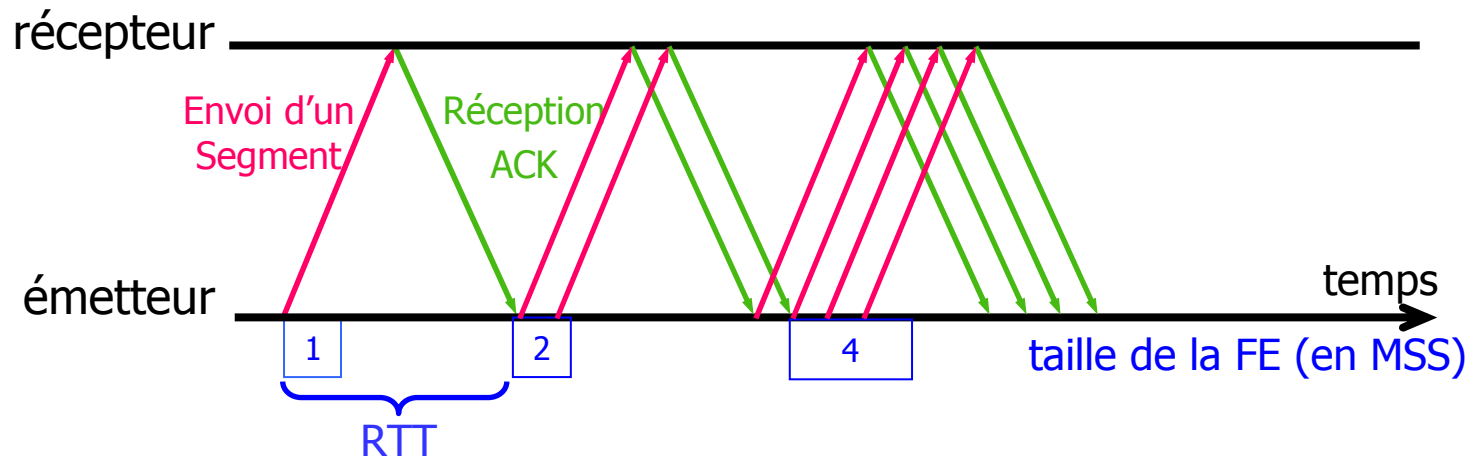
Contrôle de flux TCP (récepteur)



- ◆ Si on veut pouvoir utiliser toute la bande passante d'un lien, on doit donc avoir : **$FR > (\text{bande passante}) \times RTT$**

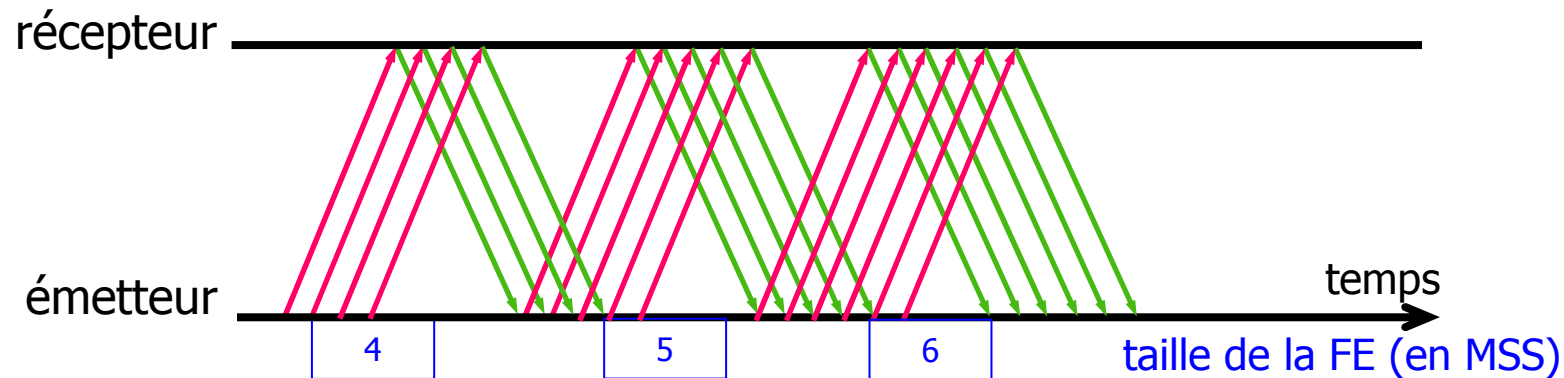
TCP : contrôle de flux (émetteur)

- ◆ Au démarrage : phase dite «*slow start*»
 - l'émetteur envoie un segment et attend le ACK correspondant
 - quand il reçoit le ACK, l'émetteur incrémente sa Fenêtre d'Emission (FE) de 1 MSS. Il envoie donc 2 segments et attend les ACK.
 - à chaque RTT, si tout va bien, la FE est doublée
 - la FE (et aussi le débit) croît donc exponentiellement jusqu'à un seuil prédéfini : le Seuil d'Evitement de Congestion (SEC)



TCP : contrôle de flux (émetteur)

- ◆ Quand le seuil SEC est atteint, l'émetteur passe en mode évitement de congestion
 - quand il reçoit le ACK, l'émetteur incrémente sa FE de $1/FE$
 - à chaque RTT, si tout va bien, la FE est augmentée de 1 MSS
 - en respectant toujours la condition : **FE < FR**



- ◆ En cas de congestion
 - l'émetteur ré-émet le paquet qui n'a pas été acquitté
 - divise par 2 le seuil SEC
 - repart en mode *slow-start* ($FE = 1$)

TCP : contrôle de flux (Tahoe)

- ◆ Le contrôle de flux d'une session TCP se caractérise donc
 - par un algorithme de gestion des FR et FE

```
pour chaque ACK {
    si (FE < SEC) alors { FE++ }      (Slow Start)
    sinon                    { FE += 1/FE } (évitemment de congestion)
}
pour chaque perte de segment TCP {
    SEC = max(FE/2, 2) (on mémorise le fait qu'il a eu congestion)
    FE = 1              (on repart en Slow Start)
}
En respectant les conditions FR > bande passante*RTT et FE < FR
```

- par un algorithme de détection de la congestion
 - réception de 3 ACK dupliqués (qui acquittent le même segment TCP)
 - pas d'accusé de réception d'un segment émis au bout de RTO secondes

Amélioration TCP : *Fast Recovery*

◆ Buts

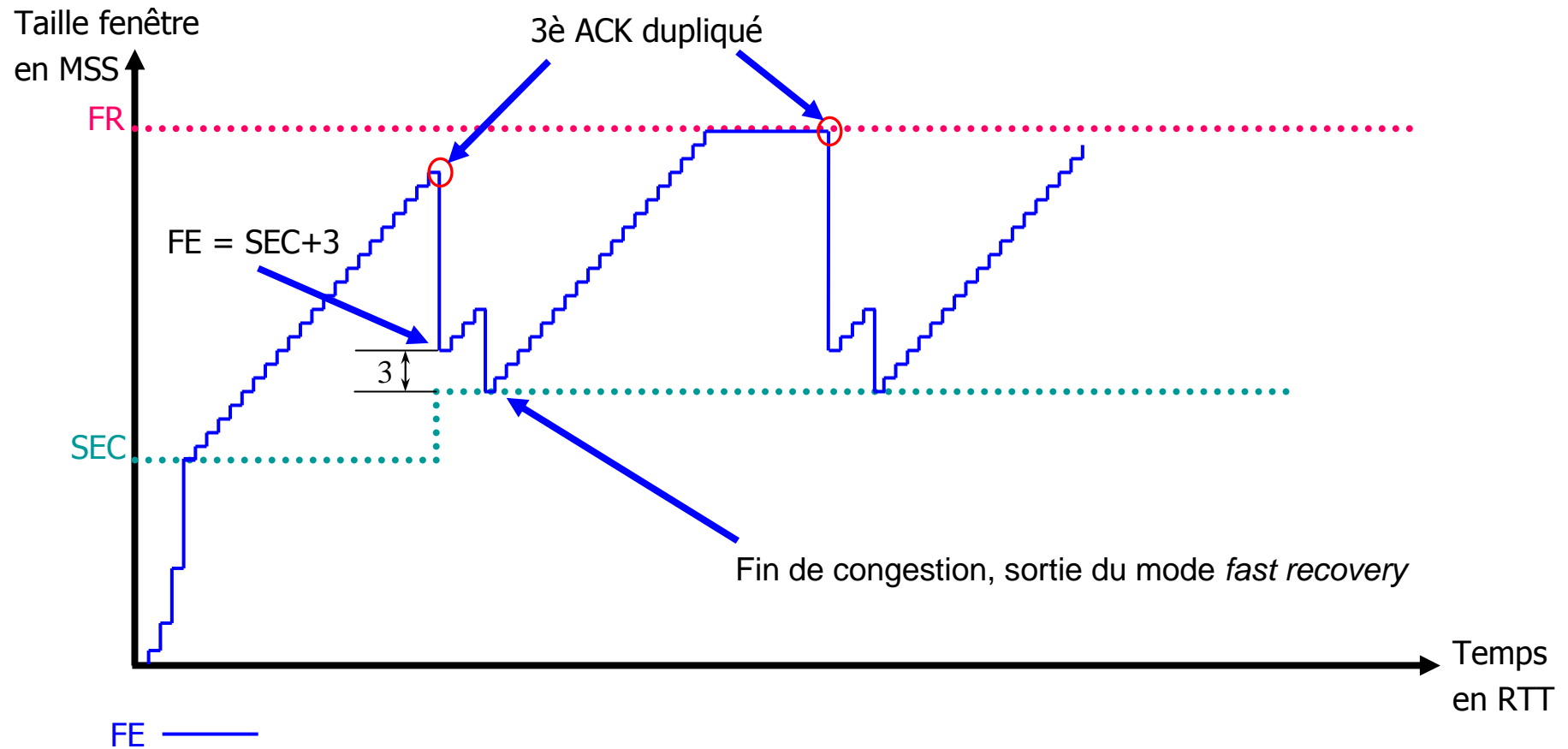
- ne pas retransmettre des segments bien arrivés (signalés par les ACK dupliqués)
- relancer la transmission de segments au plus vite

◆ Après 3 ACK dupliqués l'émetteur passe en *Fast recovery*

- $SEC = \max[FE/2, 2]$
- le segment perdu est immédiatement retransmis (*Fast Retransmit*)
- $FE = SEC + 3$ (pas de *slow start*, inflation temporaire de la FE)
- FE est incrémentée de 1 à chaque nouveau ACK dupliqué reçu
- transmission de nouveaux segments si possible (cf. FE et FR)
- au premier ACK correspondant aux nouveaux segments transmis, positionner FE à SEC et continuer en mode évitement de congestion

Amélioration TCP : Reno

◆ Reno = Tahoe + *fast recovery*



Amélioration TCP : Reno+SACK

◆ Buts

- résister aux pertes de paquets multiples
- tenir compte des SACK pour rester en mode *Fast Recovery*

◆ Après 3 ACK dupliqués l'émetteur passe en *Fast recovery*

- même mécanismes que vu précédemment avec en +
 - retransmission du dernier segment perdu indiqué par un SACK
 - émission d'un nouveau segment si S données reçues < FE
 - sortie du mode *Fast Recovery* quand tous les segments perdus sont acquités

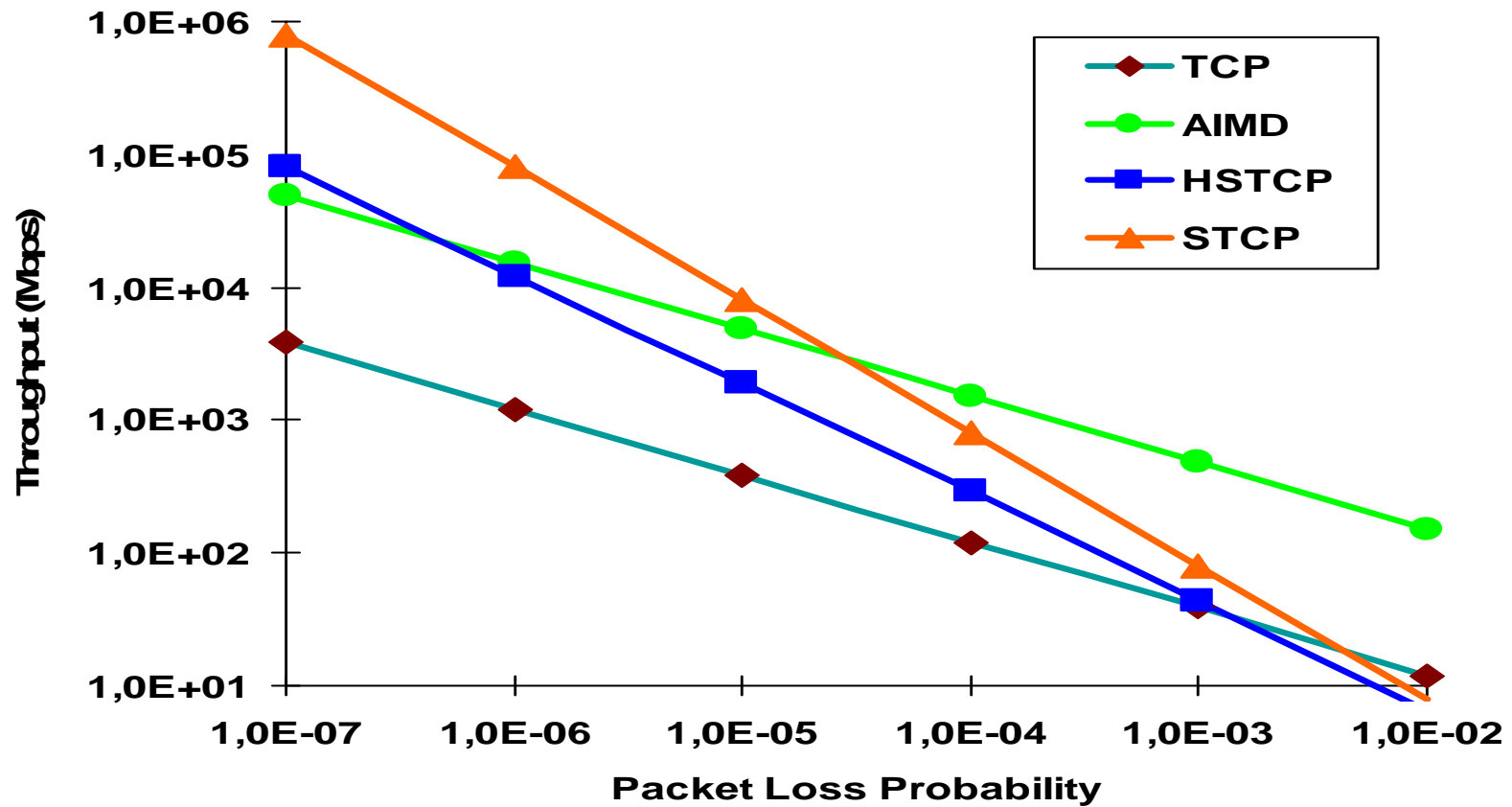
◆ Améliorations

- émission d'un segment perdu par RTT
- prolonge la phase de *Fast Recovery*, retarde/évite le passage en *slow start*

Améliorations TCP : autres directions

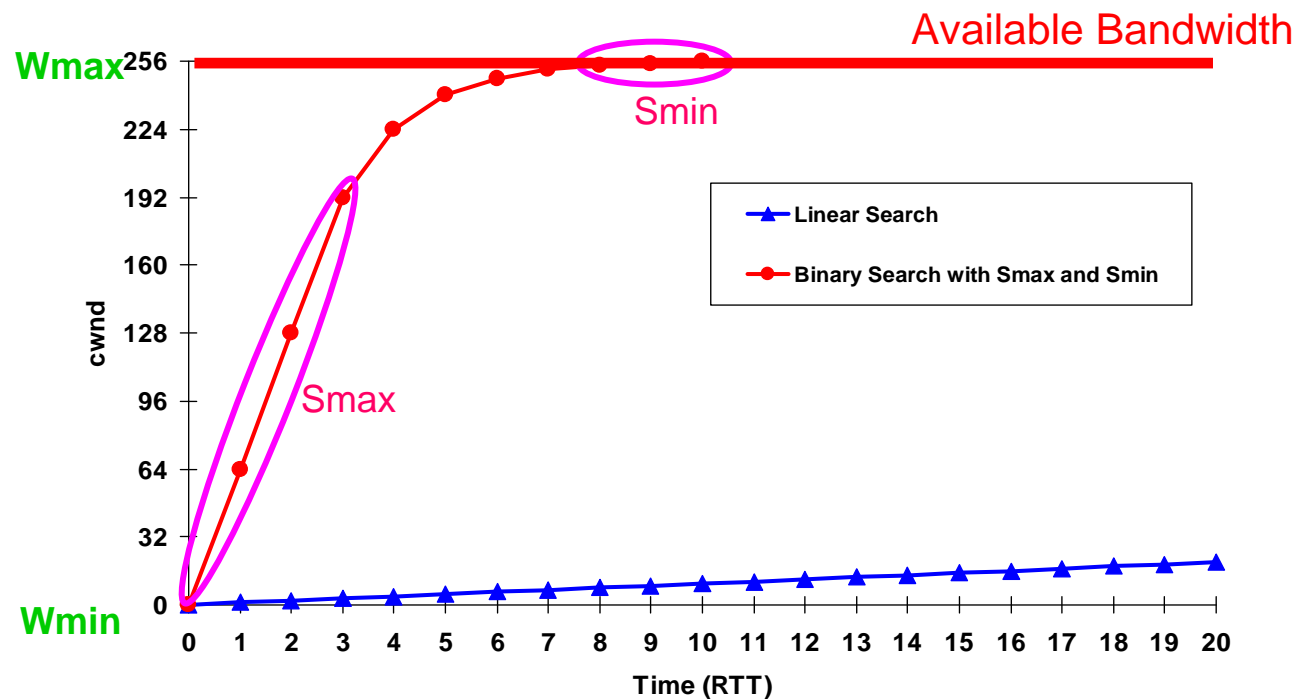
- ◆ Propositions modifiant la gestion de la fenêtre d'émission (FE)
- ◆ évitement de congestion
- ◆ RENO $FE = FE + 1$
- ◆ AIMD $FE = FE + 32$
- ◆ STCP $FE = FE + 0,01 * FE$
- ◆ HSTCP $FE = FE + \text{finc}(FE)$
 - $\text{finc}(FE)$ croît avec FE
- ◆ congestion
- ◆ $FE = FE * (1 - 1/2)$
- ◆ $FE = FE * (1 - 1/8)$
- ◆ $FE = FE * (1 - 1/8)$
- ◆ $FE = FE * (1 - \text{fdec}(FE))$
 - $\text{fdec}(FE)$ décroît avec FE
- ◆ Double objectif recherché par ces propositions
 - permettre à TCP d'atteindre des très hauts débits (10Gb/s) rapidement
 - bien prendre en compte les congestions (être *TCP Friendly*)

Améliorations TCP : résultats

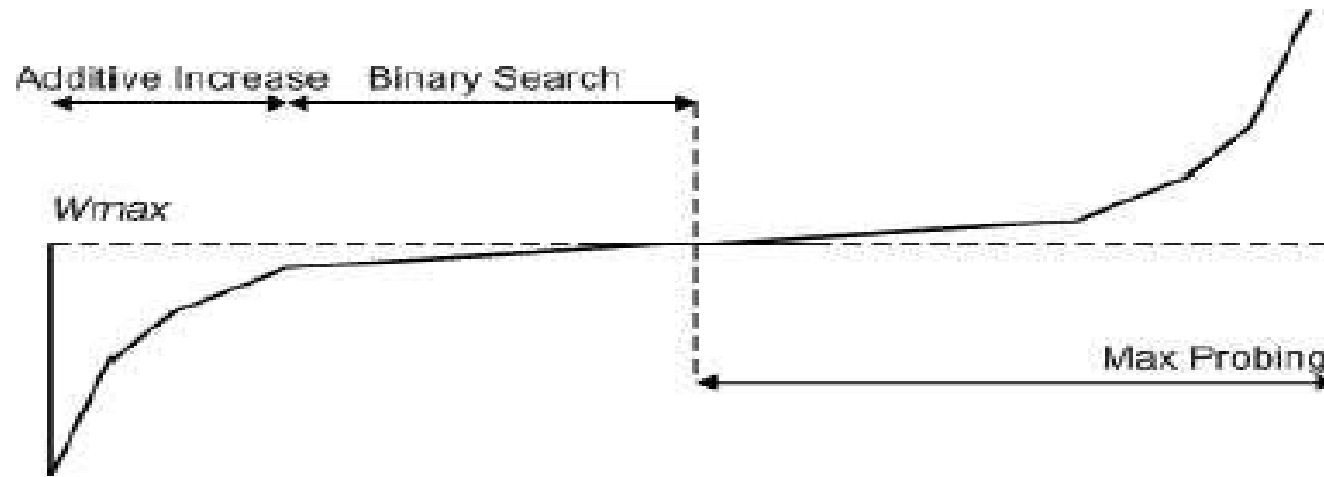


Améliorations TCP : BIC

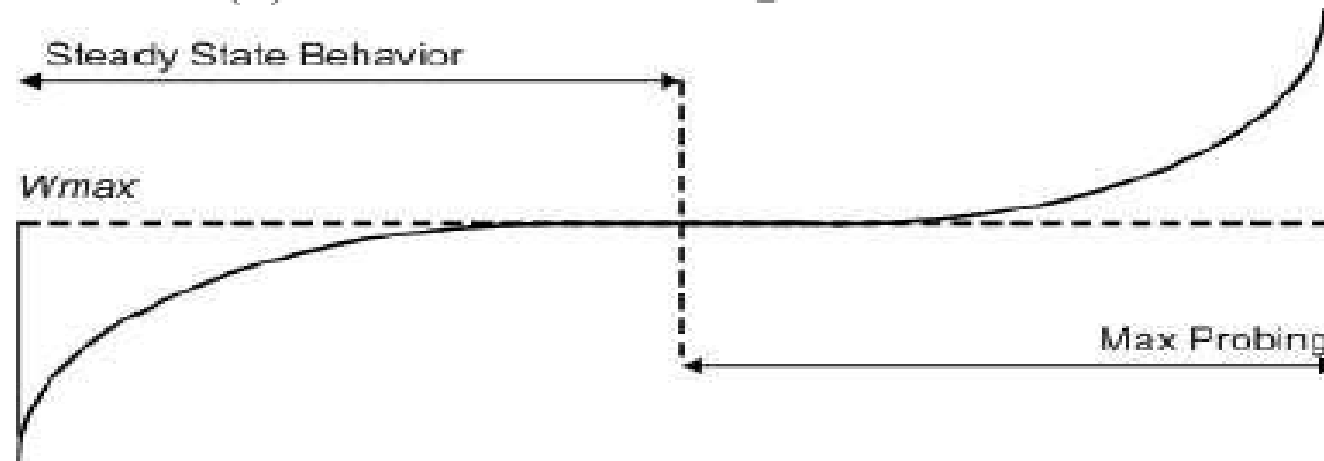
- ◆ BIC (*Binary Increase Congestion*) mix entre STCP et HSTCP
- ◆ évitement de congestion congestion
- ◆ BIC $FE = FE + f(FE, \text{historique})$ $FE = FE * (1 - 1/8)$



Améliorations TCP : CUBIC

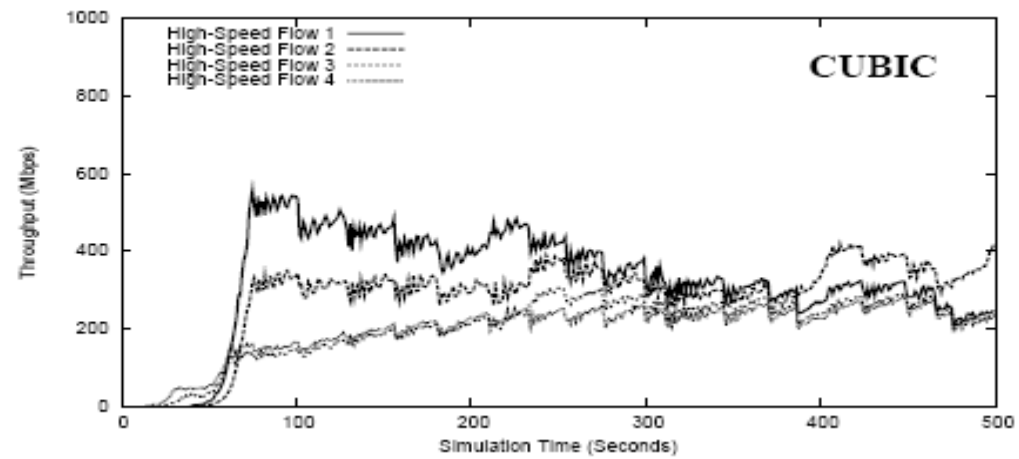
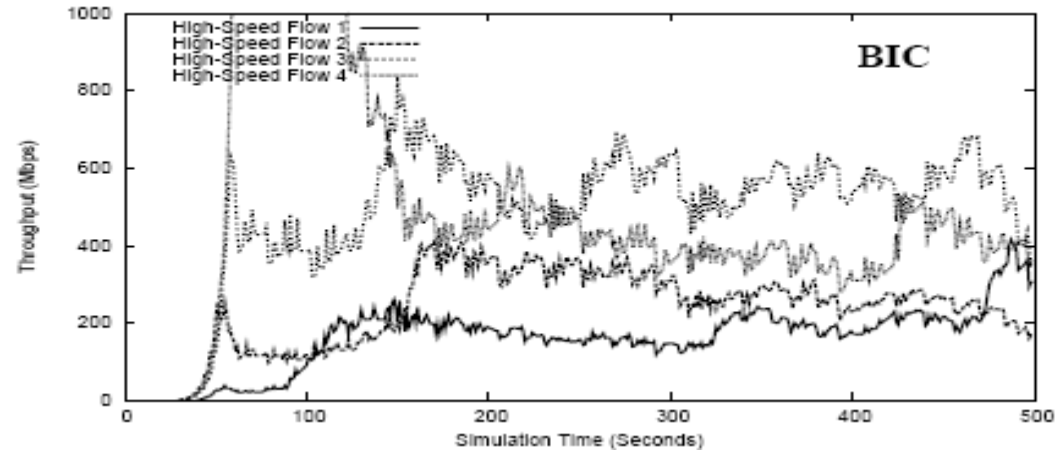


(a) BIC-TCP window growth function.



(b) CUBIC window growth function.

Améliorations TCP : CUBIC



TCP : analyse d'une session

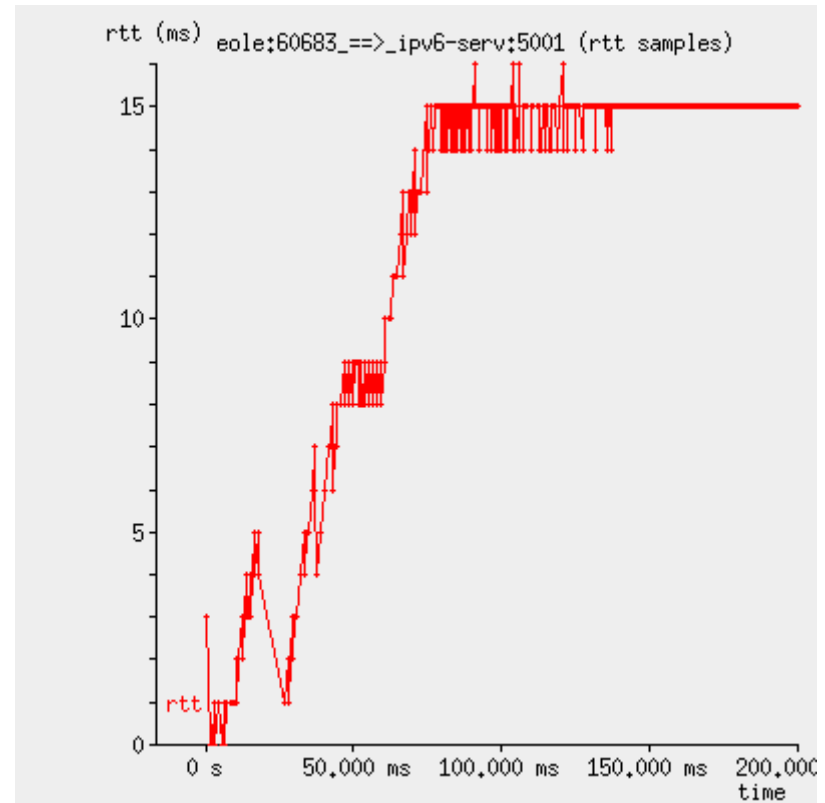
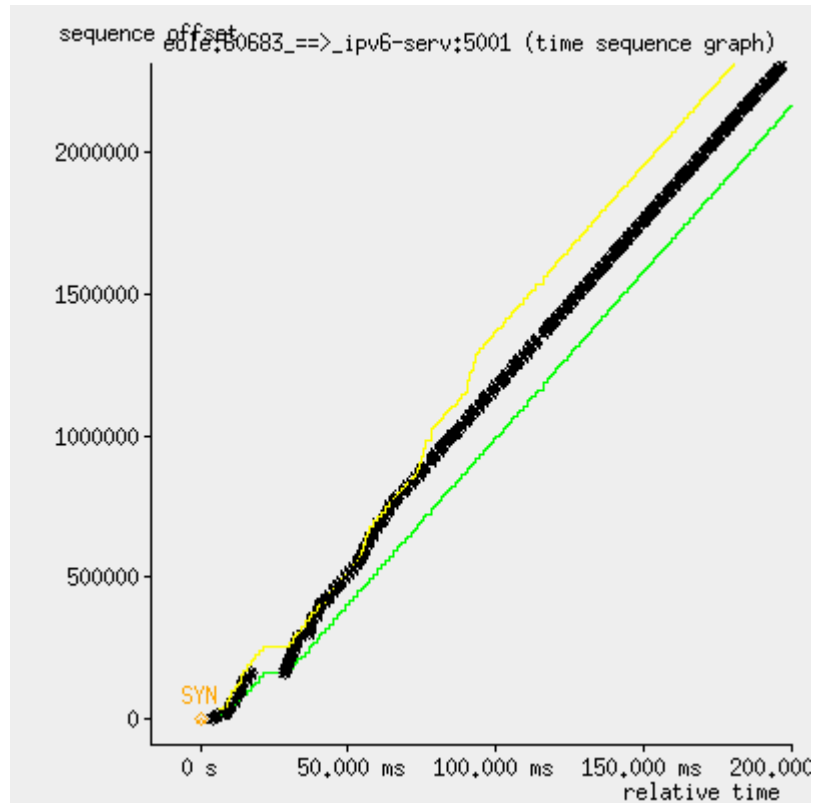
◆ La boîte à outils disponible

- capture des paquets : `tcpdump`, wireshark
- analyse la session TCP : wireshark, tcptrace+xplot, `TcpPlot`
- simulation et tests : iperf, iproute2+netem (Linux), `NS2`

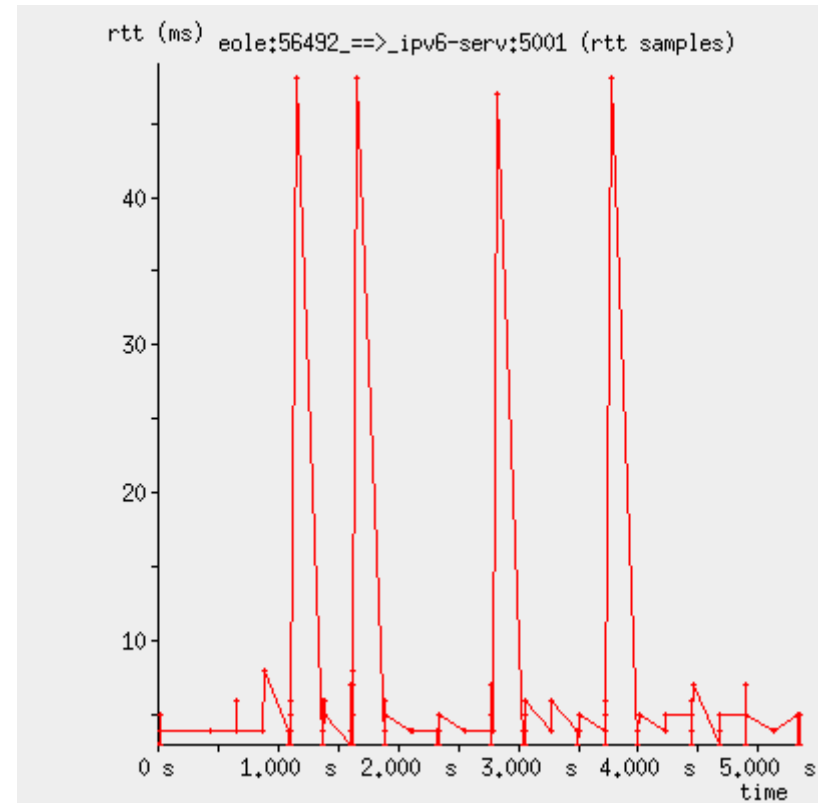
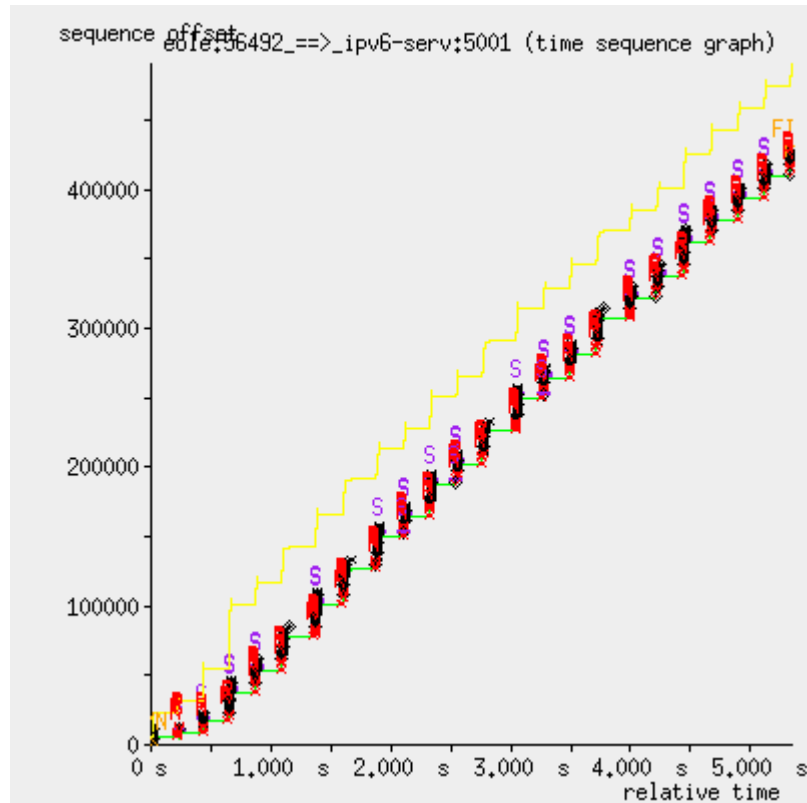
◆ Méthode d'analyse proposée

- sur l'émetteur : capture et analyse d'une session
 - `wireshark` : analyse fine, perte de segments, fermeture de la FR....
 - `tcptrace+xplot` : analyse graphique des paramètres (RTT, FE, FR, pertes de segments...) de la session

Analyse d'une bonne session TCP (1)



Analyse d'une mauvaise session TCP (1)



Le protocole DCCP

- ◆ DCCP : *Datagram Congestion Control Protocol*
- ◆ Assure ou permet
 - contrôle de congestion (avec choix de l'algorithme possible)
 - CCID=2 *TCP like* (similaire à AIMD, à privilégier)
 - CCID=3 *TCP-Friendly Rate Control* (TFRC), meilleur lissage du débit
 - connexion bi directionnelle ou unidirectionnelle (pas de ACK)
 - ouverture/fermeture de la connexion
 - négociation d'options
- ◆ Mais
 - perte de paquets non récupérée, pas de multicast
- ◆ Applications cibles
 - flux de données importants (isochrones, à débit constants ou non)

DCCP : format du paquet (1)

- ◆ Paquet = en-tête générique+en-tête additionnelle+options+data
- ◆ En-tête générique de 16 octets, toujours présent ($X = 1$)

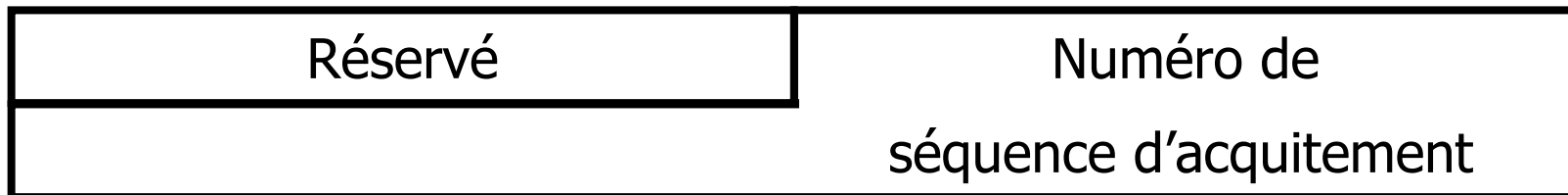
Port source				Port destination			
Data offset		CCval	CsCov	Somme de contrôle			
Res	Type	X	Réservé			Numéro de séquence	

- ◆ Si $X = 0$ (en-tête réduit à 12 octets)

Port source				Port destination			
Data offset		CCval	CsCov	Somme de contrôle			
Res	Type	X	Numéro de séquence				

DCCP : format du paquet (2)

- ◆ L'en-tête aditionnelle (dépend du type de paquet)
 - Exemple : tous types sauf DCCP-Request DCCP-Data (X=1)



- Si $X = 0$ (en-tête aditionnelle réduite à 4 octets)



- ◆ Options
 - données complémentaires (dépend du type de paquet)
 - ex : négociations de fonctionnalités, contrôle de congestion, ACK

DCCP : conclusion

◆ Pour

- Design intéressant (intermédiaire entre UDP et TCP)
- peut simplifier l'écriture d'applications (prise en charge congestion)
- introduit moins de variation de délai que TCP

◆ Mais

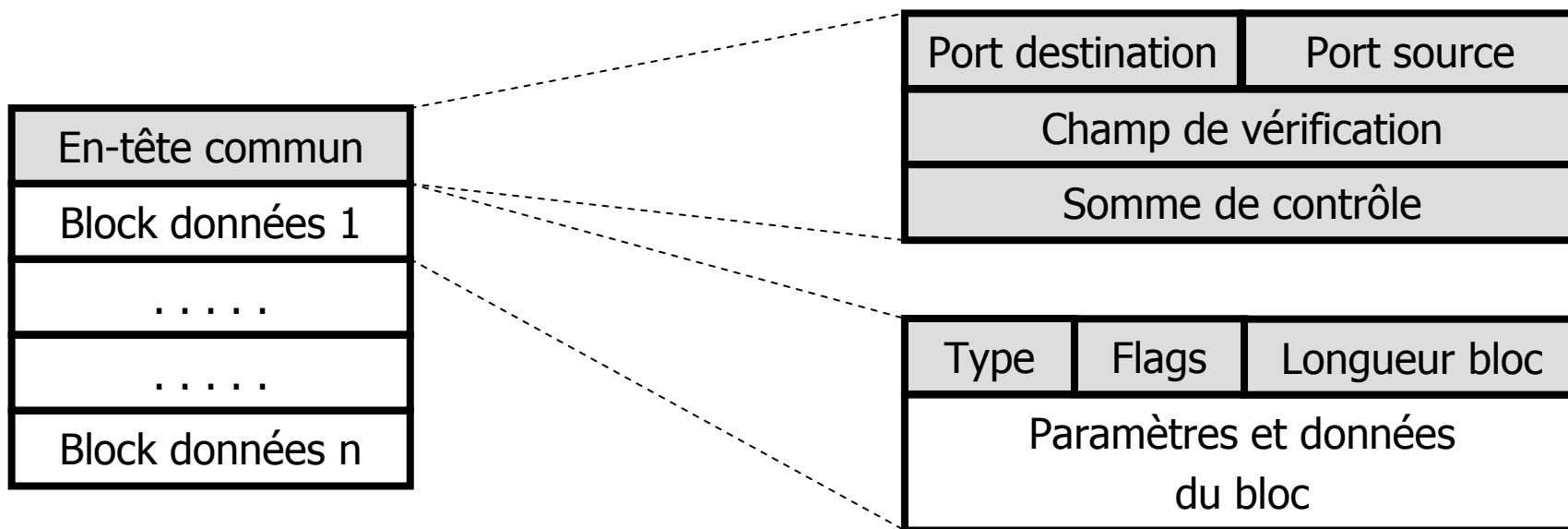
- implémentation complexe (en-tête variable, options, négociations)
- peu déployé à ce jour bien que disponible (Linux, BSD)

Le protocole SCTP

- ◆ *SCTP : Stream Control Transmission Protocol*
- ◆ Association entre deux hôtes
 - support de la multi-domiciliation aux deux extrémités
 - plusieurs flux par plusieurs chemins (liés aux interfaces)
 - contrôle de congestion par flux
 - meilleure sécurité contre : inondation de SYN, homme du milieu
 - haute disponibilité du canal de communication entre les hôtes
- ◆ Plusieurs modes de transmission
 - flots d'octets (comme TCP)
 - ordre partiel (par flux)
 - sans ordre garanti (comme UDP)

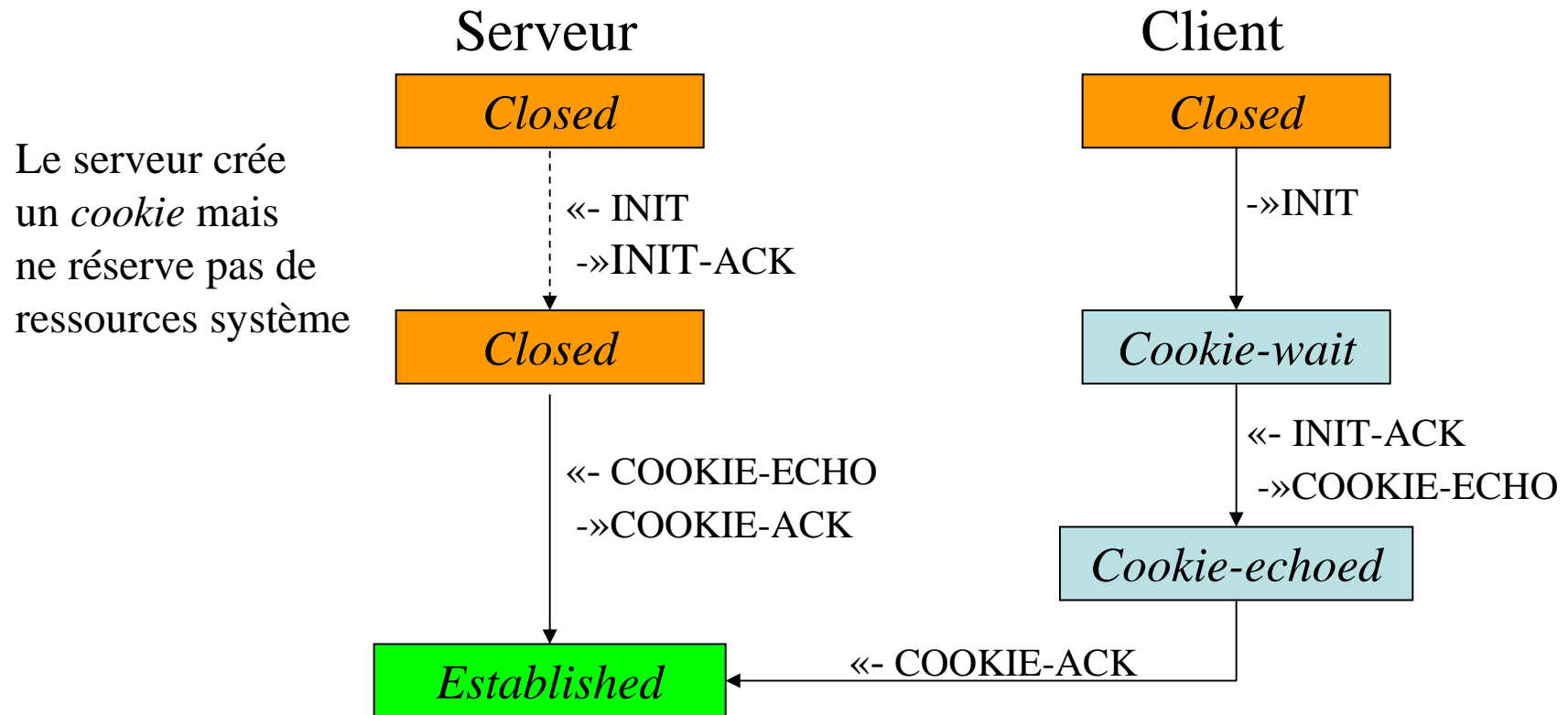
SCTP : format du paquet (1)

- ◆ Paquet = en-tête générique+blocs de données



SCTP : ouverture de session

◆ Association en 4 étapes



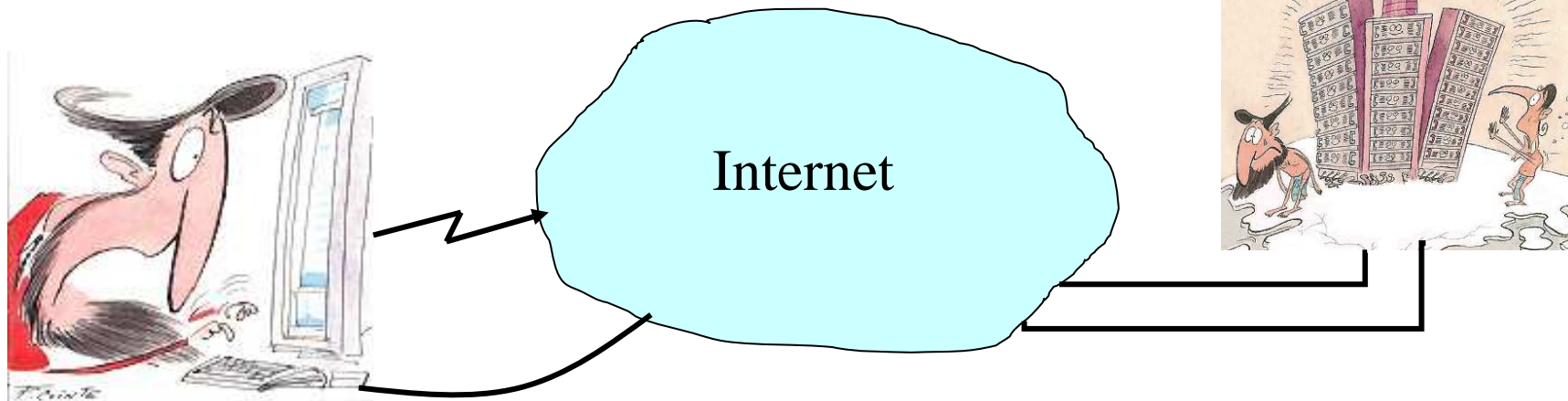
-le client doit rester visible du serveur
-la consommation de ressources équilibrée

==>

Meilleure protection contre
le déni de service

SCTP : la multi domiciliation

- ◆ Plusieurs interfaces par hôte peuvent participer à l'association
 - le *Init-Ack* contient toutes les @IP participant à l'association
 - un des chemins est choisi comme chemin primaire (données+contrôle)
 - le/les chemins de secours peuvent être utilisés pour les retransmissions
 - surveillance des chemins par « chien de garde »



SCTP : le multi-flux

- ◆ Une application (vidéo) = de plusieurs flux de données
 - vidéo, son_droit, son_gauche, sous_titres, infos de contrôle....
 - ces flux sont pris en charge directement par SCTP
 - TSN : N° du paquet SCTP
 - SI : identifiant de flux
 - SSN : N° du bloc dans le flux
 - PPI : Protocole associé aux données

Type	Flags	Longueur bloc
TSN		
SI		SSN
PPI		
Données du bloc		

SCTP : conclusion

◆ Pour

- haute disponibilité native du canal de communication (multi-domiciliation)
- le multi-flux de données applicatives
- prise en compte de la mobilité (modif des @IP d'une association)

◆ Mais

- implémentation complexe (en-tête variable, options, négociations)
- peu déployé à ce jour bien que disponible (Linux, BSD)
- coexistence avec IPsec ?

Autres protocoles de transport

- ◆ RTP (*Real-time Transmission Protocol*)
 - caractéristiques de transmission temps réel (latence, IP délai, ou variation de délai)
 - peut utiliser un transport UDP, DCCP, TCP
 - support du multicast
- ◆ RTCP (*Real-time Transmission Control Protocol*)
 - associé à RTP
 - suivi QoS, contrôle flux, contrôle des participants
- ◆ RTSP (*Real Time Streaming Protocol*)
 - spécifique à la transmission de flux audio et vidéo
 - utilise un transport UDP, TCP

Conclusion

- ◆ UDP
 - protocole efficace, mais potentiellement agressif sur le réseau
- ◆ TCP
 - Ok, pour les dernières versions (tenir les OS à jour)
- ◆ DCCP
 - À préférer à UDP car contrôle de congestion
- ◆ SCTP
 - À préférer à TCP pour les nouvelles applications