Contrôle de congestion dans le protocole TCP

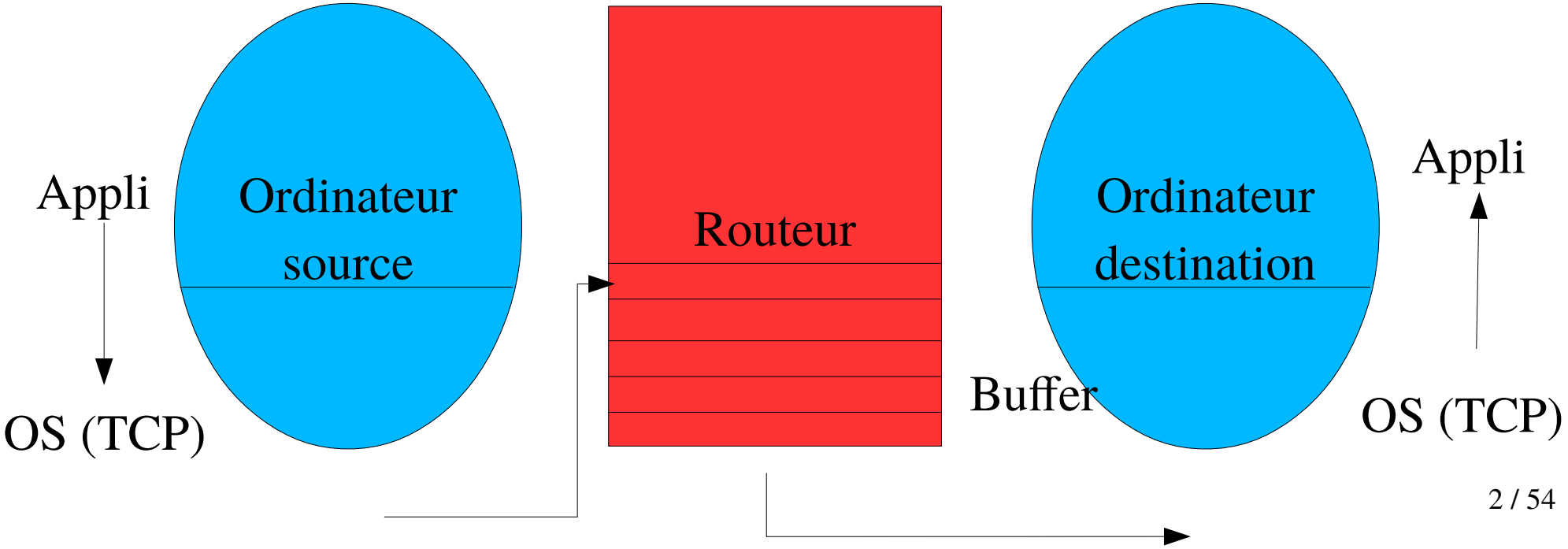
Eugen Dedu

Maître de conférences
Univ. de Franche-Comté, UFR STGI, M2 IMR
Montbéliard, France
sept. 2018

http://eugen.dedu.free.fr
eugen.dedu@univ-fcomte.fr

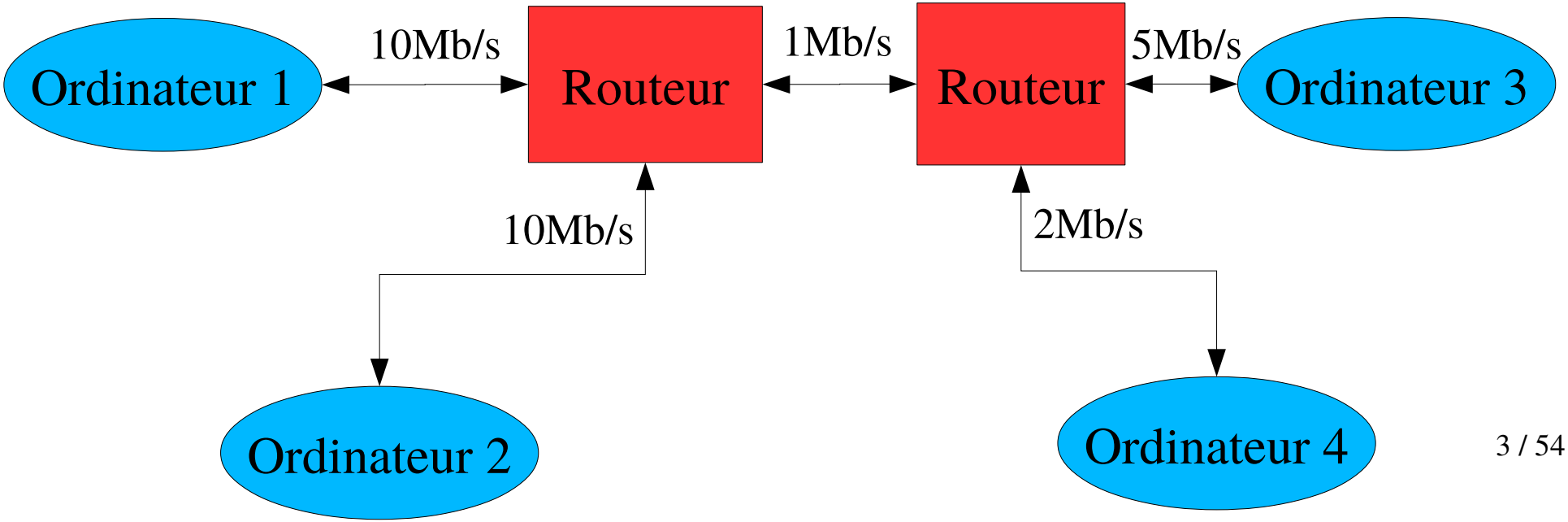
Introduction aux réseaux

- Transfert d'une donnée, e.g. un fichier par FTP
- Division en paquets par TCP de l'ordinateur source
- File d'attente (buffers) des routeurs



Introduction à la congestion

- Lien (bande passante, latence), analogie avec le train
- Ex : taux de transfert idéals entre différents ordi
- CC = adaptation à la bande passante disponible à chaque instant



Inconvénients de la congestion

- Congestion = routeur avec file d'attente pleine
- Si routeur avec file d'attente (quasi-)pleine :
  - rejet de paquet (car débordement mémoire routeur)
  - délais importants de transfert (car attente dans les files des routeurs)
- Causes de la perte d'un paquet :
  - problème matériel
  - problème d'environnement (souvent dans les réseaux sans fil)
  - mais surtout congestion d'un routeur
- En filaire, perte d'un paquet ~= congestion routeur

Netographie

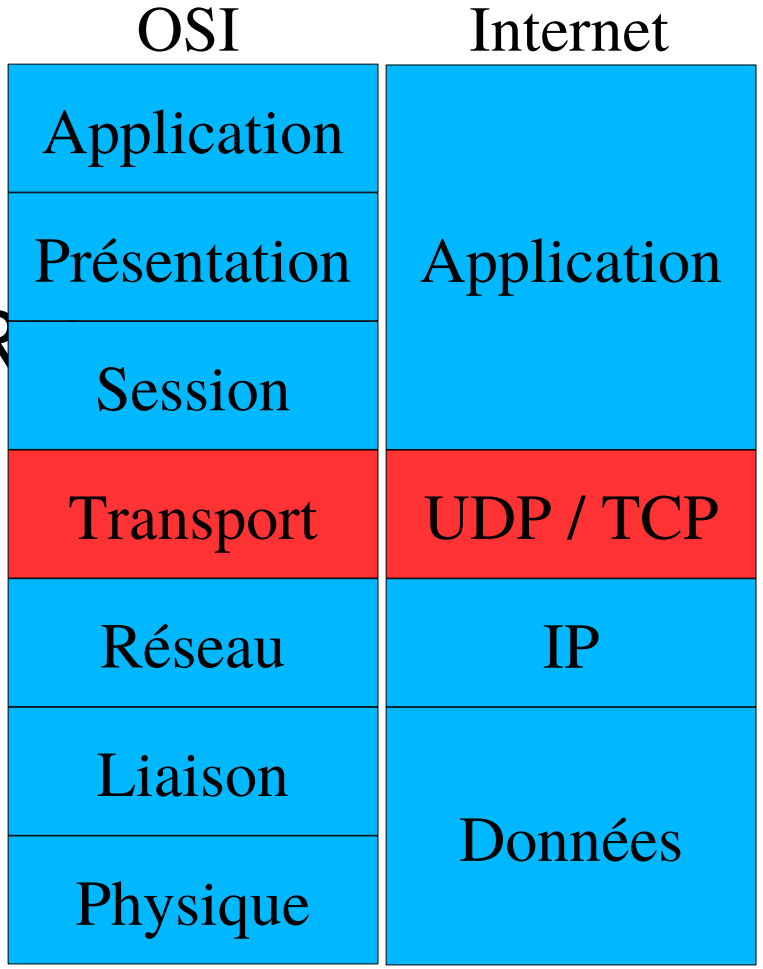
- TCPs classiques : “Practical Analysis of TCP Implementations: Tahoe, Reno, NewReno”, Bogdan Moraru et al., 2003
- Principe de TCP et CC : “Internetworking with TCP/IP”, Douglas Comer, 2000
  - ou tout autre livre sur TCP/IP
- Recherches sur Internet

Plan

- Contexte
- TCP
  - principes
  - contrôles de congestion, algorithmes, options
  - qualité de service : RED, ECN
- Support des systèmes d'exploitation

Contexte : modèle OSI

- OSI : Open Systems Interconnection, 1982
- Divisé en couches

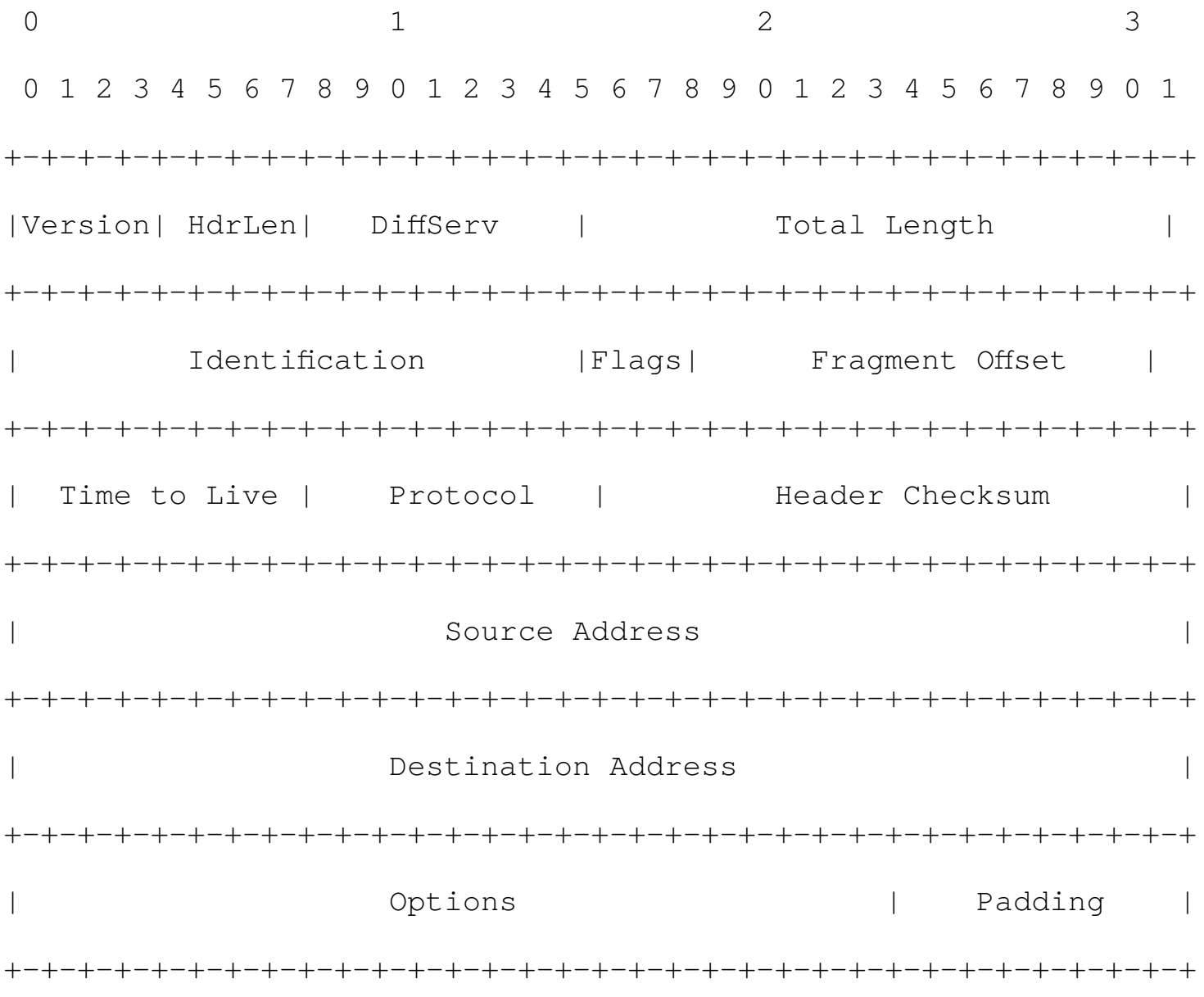


- Transfert de données avec CR
  - end-to-end
- UDP : 0 % de fiabilité
- TCP : 100 % de fiabilité

Couche réseau : protocole IP

- Couche réseau
- Assure le transfert des données entre deux machines sur des réseaux différents
- Routage

En-tête IP


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Version
HdrLen
DiffServ
Total Length
Identification
Flags
Fragment Offset
Time to Live
Protocol
Header Checksum
Source Address
Destination Address
Options
Padding

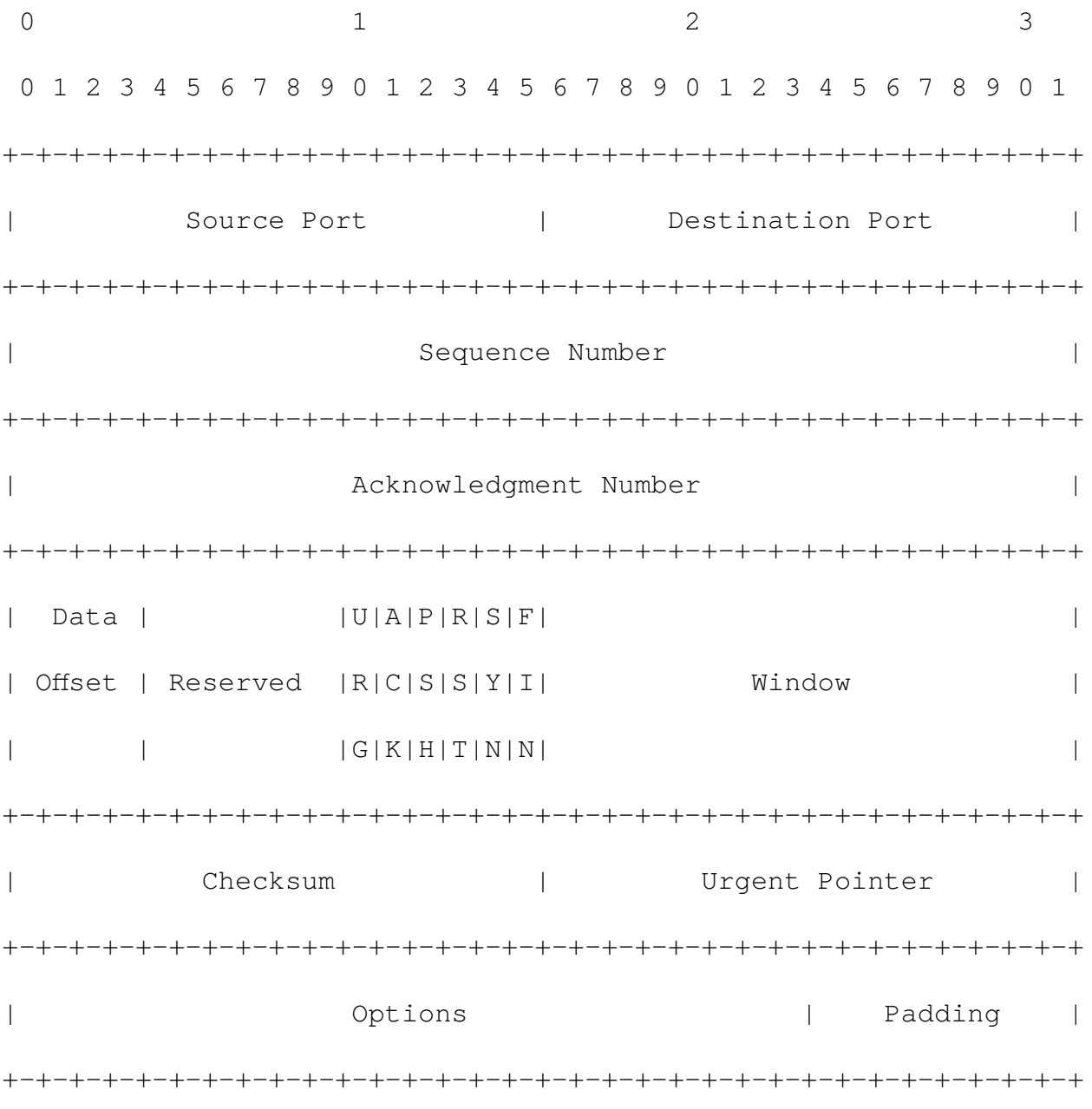
Contexte : UDP

- UDP = User Datagram Protocol
- Envoi de paquets :
  - dès qu'il est reçu de l'application
  - sans confirmation de réception => fiabilité 0 %
- Utile pour des services simples ou des données vidéo etc.
  - NFS, DNS, ...
- CBR = Constant Bit Rate
  - envoi régulier de paquets

TCP

- TCP, Transmission Control Protocol, RFC 2581
- Contrôle d'erreurs :
  - fiabilité 100 %, non-duplication
  - ordonnancement
- Contrôle de flux
- Contrôle de congestion basé fenêtre
- Orienté connexion, full-duplex
- Orienté bit : l'application destinataire reçoit un flux ; TCP divise les données en paquets
- ...

En-tête TCP


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Reserved
U A P R S F
R C S S Y I
G K H T N N
Window
Checksum
Urgent Pointer
Options
Padding

Principe “end-to-end”

- Un hôte est impliqué dans 1 transfert, un routeur dans beaucoup de transferts
- Les deux hôtes d'extrémité sont responsables du débit de transfert des données :
  - à quel vitesse transmettre
  - quand transmettre
  - quand accélérer et décélérer le débit
- Le réseau ne leur fournit pas des informations explicites

Schéma multi-couche de transmission

- Couches : application, transport, réseau, mac
- Sur l'émetteur et le récepteur
- Files d'attente
- voir image tcp.png
- Algorithme de Nagle
  - for (i=0; i<1000000; i++) write(socket, &buffer[i], 1);
  - attend jusqu'à un certain temps s'il y a des paquets non accusés

Concurrence des flux

- Les flux TCP sont équitables : chacun des N flux prend 1/N de la bande passante
  - en réalité, la fraction est inversement proportionnelle au RTT du flux :

    débit ~ MSS / (RTT * sqrt(p)), p = probabilité d'erreur
- Les flux UDP ne sont pas équitables
  - TCP est défavorisé par rapport à UDP
- Évolution de la vitesse de transfert en concurrence de flux
- (Voir figures de ~/net*/ns2/hw1.pdf)

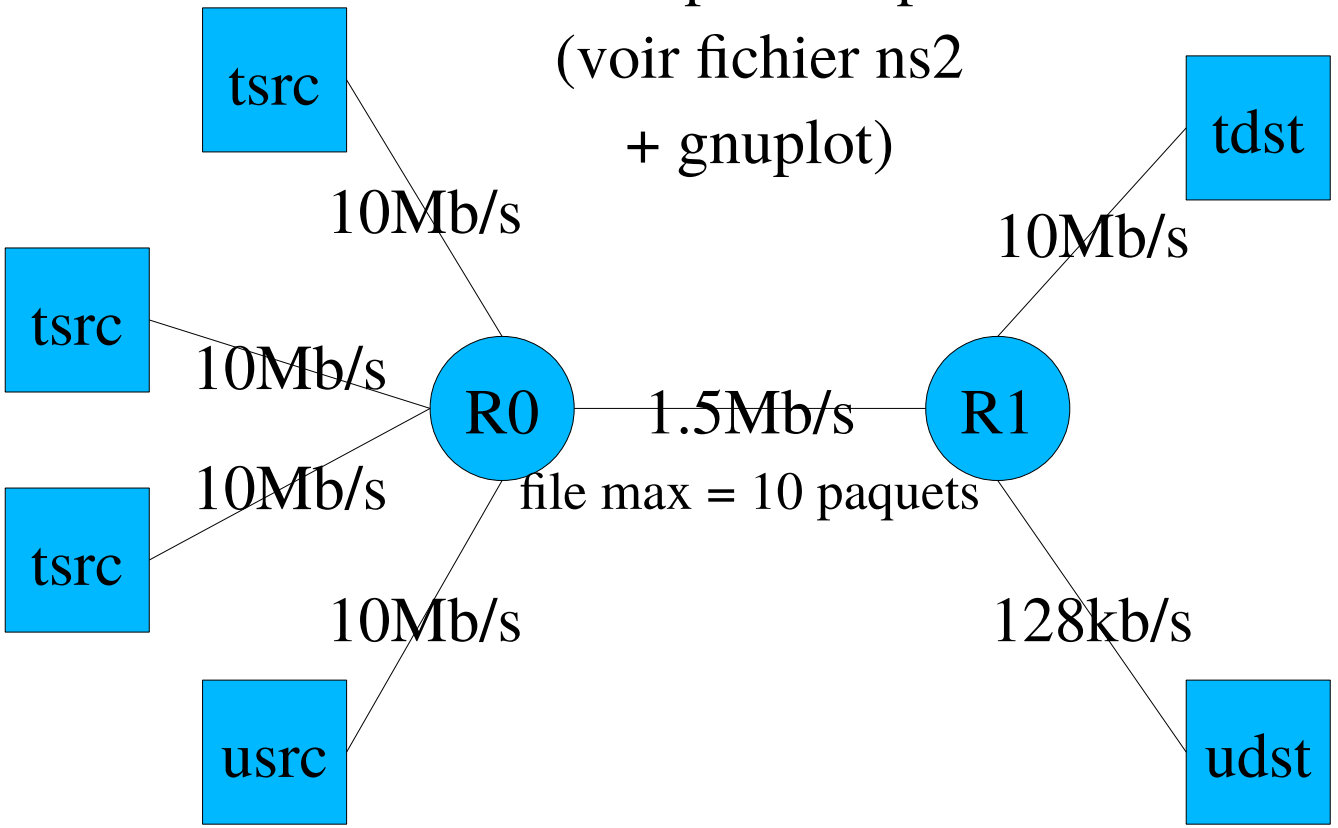
Effondrement du réseau (*Congestion collapse*)

RFC 2914

3 tcp + 1 udp

(voir fichier ns2

+ gnuplot)



UDP rate	UDP goodput	TCP goodput %	Total goodput %
10.5kb/s (0.7%)		0	95	95
27.0kb/s (1.8%)		1	94	95
39.0kb/s (2.6%)		2	93	95
79.5kb/s (5.3%)		4	90	94
132.0kb/s (8.8%)		8	87	95
157.5kb/s (10.5%)		8	85	93
196.5kb/s (13.1%)		8	82	90
262.5kb/s (17.5%)		8	78	86
394.5kb/s (26.3%)		8	70	78
789.0kb/s (52.6%)		8	45	53
876.0kb/s (58.4%)		8	40	48
985.5kb/s (65.7%)		8	34	42
1126.5kb/s (75.1%)		7	26	33
1314.0kb/s (87.6%)		8	13	21
1578.0kb/s (105.2%)		7	1	8
1972.5kb/s (131.5%)		6	0	6

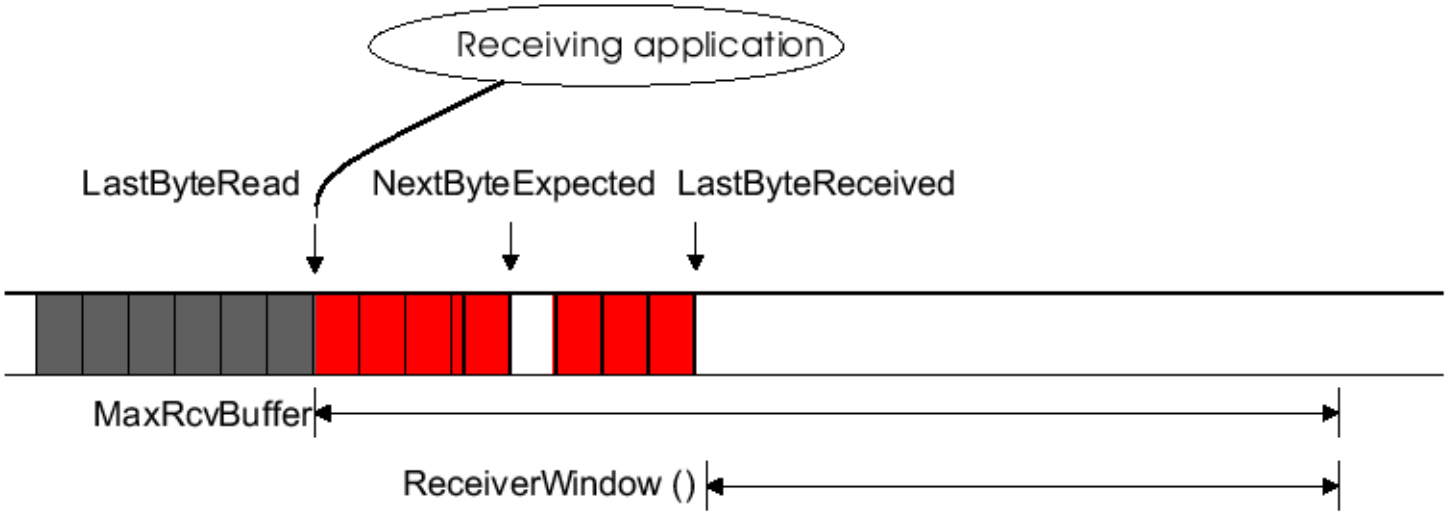
Q : Sur la totalité des paquets envoyés, combien en arrivent sur tdst et sur udst ?

TCP : définitions

- Contrôle de flux : par rapport au récepteur
  - l'émetteur adapte le nombre de paquets envoyés à la taille du buffer de réception
- Contrôle de congestion : par rapport au réseau
  - l'émetteur adapte le débit des données envoyées à la bande passante instantanée du réseau
  - en général, la perte d'un paquet est la seule information sur l'état du réseau, mais la variation du RTT ou l'ECN est parfois utilisé aussi
  - NB : ce n'est pas la taille des paquets, mais leur débit d'envoi qui change

TCP : définitions : fenêtres du récepteur

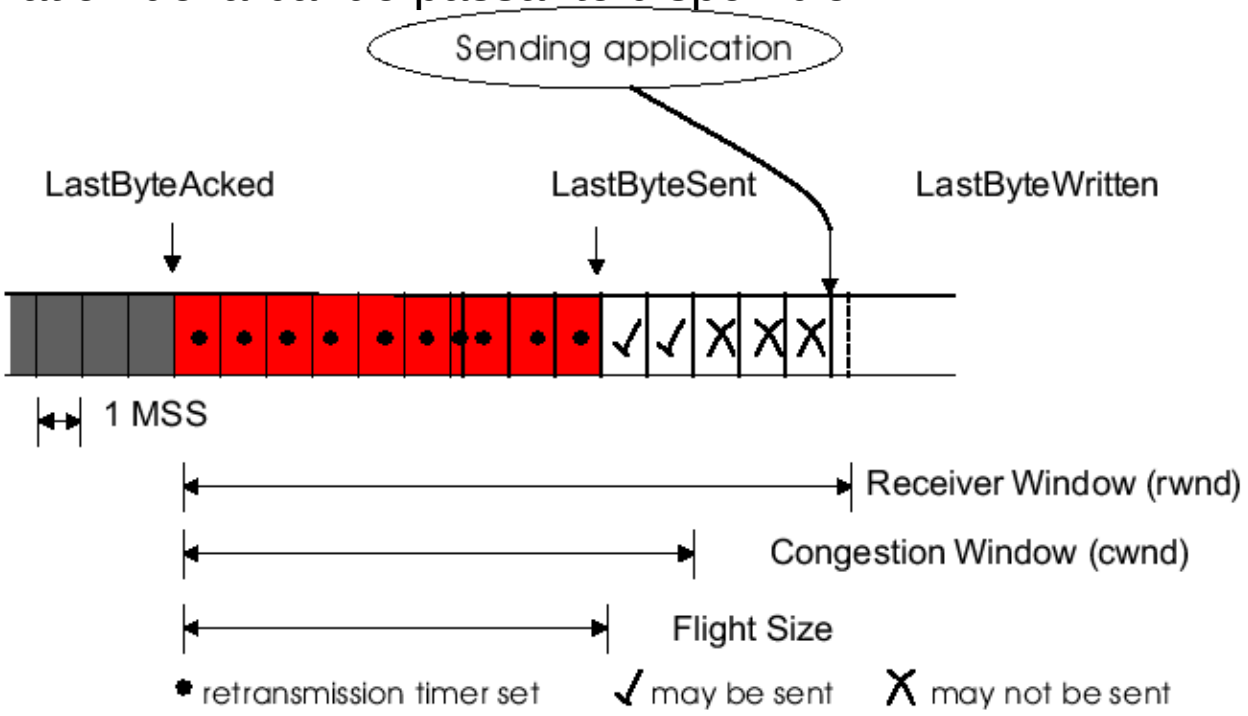
- Buffer de réception : espace de stockage des données (reçues ou non)
- Fenêtre de réception : nombre max de paquets que le destinataire peut recevoir à un certain moment (espace libre au destinataire)



source

TCP : définitions : fenêtres de l'émetteur

- Fenêtre d'émission : les données de l'application
- Fenêtre de congestion (cwnd)
  - sous-fenêtre mobile de la fenêtre d'émission
  - nombre max de paquets que l'émetteur peut envoyer sans recevoir aucun accusé
- Seuil de démarrage lent (ssthresh)
  - estimation de la bande passante disponible



source

TCP : définitions : accusés de réception

- Accusé de réception
  - récepteur -> émetteur
  - numéro du 1er octet attendu par le récepteur
- TCP classique : les accusés sont cumulatifs
- DupACK : un accusé identique au précédent
  - ex. : si paquet N arrive au récepteur avant N–1, son accusé est identique à l'accusé de N–2
- Delayed ACK : retarder les accusés
  - utile quand l'application répond au paquet : au lieu de 2 paquets (ack, data), on génère un seul paquet ack+data
  - le récepteur envoie après min (le 2ème pq reçu, 500 ms, paquet de données à envoyer) (sauf cas particuliers (après perte), voir TD) [RFC 2581]

TCP : définitions : horloges

- RTT (Round Trip Time)
  - temps entre l'envoi d'un paquet et la réception de son accusé
- RTO (Retransmission Timeout)
  - à chaque envoi d'un paquet de données, une horloge propre est lancée
  - si l'horloge expire, le paquet est retransmis
    - le RTO est affecté dynamiquement, en fonction du RTT [RFC 2988]
    - exemple : RTO = 4*RTT

TCP : mécanismes de CC

- Le CC est géré exclusivement par l'émetteur
  - le récepteur ne fait que renvoyer des accusés de réception
- Les algorithmes basiques de CC supportés par TCP sont [RFC 2581] :
  - slow start
  - congestion avoidance
  - fast retransmission
  - fast recovery

Principe AIMD de TCP

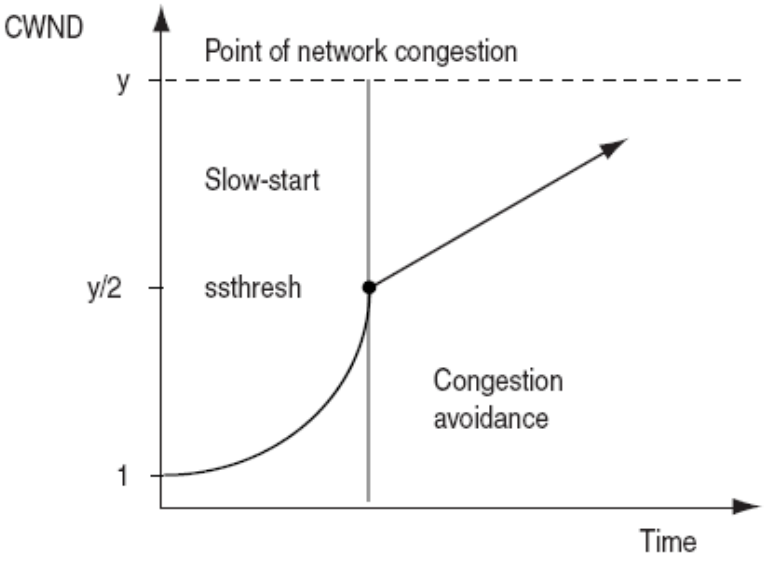
- Additive Increase, Multiplicative Decrease
- I : $w(t+RTT) = w(t) + a$, $a>0$
  - upon receiving one window of acks in an RTT
- D : $w(t+dt) = (1–b)w(t)$, $0<b<1$
  - upon a loss
- En TCP classique, $a=1$ et $b=0.5$

Initialisation

- ssthresh = valeur arbitraire
- cwnd = 1 (en réalité il est plus grand, par ex. 10 pour linux depuis 2010)
- Slow start

TCP : slow start et congestion avoidance

- Si cwnd <= ssthresh, on est en slow start, sinon en congestion avoidance
- SS :
  - but : retrouver rapidement la bande passante disponible
  - cwnd++ à chaque non dupack reçu (cwnd *= 2 à chaque RTT), RFC 2581
    - (croissance exponentielle)
- CA :
  - but : augmenter le débit en testant gentiment la bande passante disponible
  - cwnd++ à chaque RTT (après cwnd ACKs reçus)
    - (croissance linéaire)
- Si perte par timeout (=> pertes successives) :
  - ssthresh = cwnd / 2
  - entre en slow start avec cwnd=1
- Si perte par 3 dupack (=> perte isolée) :
  - ssthresh = cwnd / 2
  - entre en fast retransmission



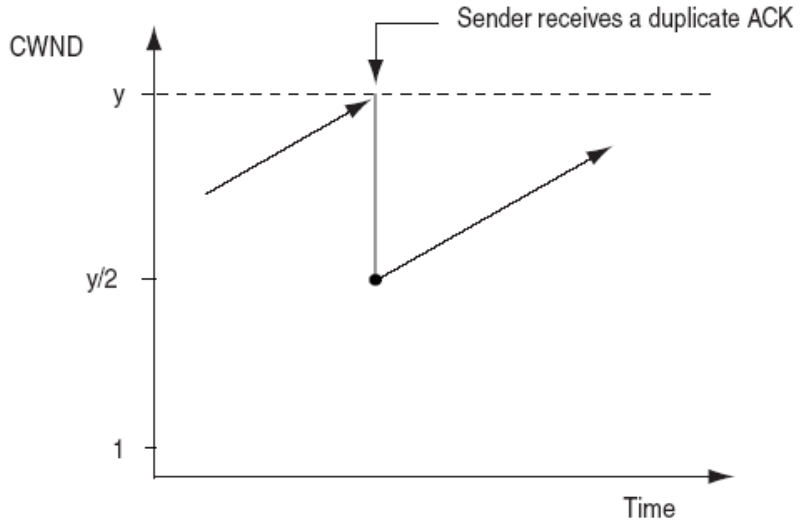
TCP : fast retransmission

- Étape très courte, arrive ici après une perte par 3 dupack
- Fast retransmission = on n'attend plus le timeout, mais :
  - on retransmet tout de suite le paquet
  - on entre en fast recovery (sauf Tahoe : slow start avec cwnd=1)

TCP : fast recovery

- Étape temporaire, arrive ici depuis fast retransmission
- But : attendre cwnd/2 ACKs, ensuite renvoyer un nouveau paquet pour chaque ACK reçu
- ssthresh = cwnd / 2
- cwnd = ssthresh + 3 (“gonflement” de cwnd)
  - => envoi **éventuel** de nouveaux paquets
  - 3, car 3 paquets accusés
- Pour chaque dupack, cwnd++
  - => envoi **éventuel** d'un nouveau paquet
- Réception d'un non dupack

  (“dégonflement” de cwnd) :
  - cwnd = ssthresh
  - retour au congestion avoidance



TCP : évolution des CC

- 1974–1980 Protocoles TCP et UDP, les deux sans CC
- 1988 TCP Tahoe, 1er CC, slow start + cong. avoidance + fast ret.
- 1990 TCP Reno, Tahoe + fast recovery, se remet plus rapidement lors d'une perte
- 1994 TCP Vegas, basé sur l'historique du RTT (état des routeurs)
- 1999 TCP NewReno, TCP Reno + adaptation de frec, gère mieux plusieurs pertes
- Options (entre l'émetteur et le récepteur) :
  - 1992 Window scaling et timestamps, gère de grandes fenêtres de réception, resp. mesure les RTTs
  - 1996 SACK, DSACK, gère mieux les pertes en spécifiant exactement les paquets reçus
- 1994 ECN, les routeurs donnent des informations sur la congestion
- 1998–2002 cwnd initial augmente de 1 à 2–4 segments (~4 ko)
- 1999/2003 Algorithme ABC, modification d'un CC, des octets à la place de paquets
- 2000 TFRC, CC basé équation
- CCs réseaux sans fil :
  - 2001 TCP Westwood+, basé sur l'historique du RTT, meilleure utilisation si pertes aléatoires
- CCs pour grands cwnd (réseaux « rapides ») :
  - 2003 High-speed TCP, suivi de 2004 BIC, 2005 CUBIC, 2006 Compound TCP
- 2006 Protocole DCCP, entre UDP et TCP, sans retransmission, choix du CC
- 2011 bufferbloat, CoDel
- 2013 cwnd initial augmente de 2–4 à 10 segments
- 2013 Algorithme PRR, se remet plus rapidement lors d'une perte dans les flux courts (Web)
- 2013 Data Center TCP
- 2017 BBR (Bottleneck Bandwidth and Round-trip propagation time)
- Beaucoup d'autres...

Versions TCP courantes

- linux :
  - avant 2004 : BIC avec SACK (et timestamps ?)
  - depuis 2004 : window scaling par défaut
  - depuis 2006 : CUBIC et ABC par défaut
  - depuis 2012 : ajout de PRR
- windows :
  - xp (2001) : ???, pas de window scaling
  - vista (2007) : CTCP, window scaling par défaut
  - 7 (2009), 8 (2012), 10 (2015) : ??
- OS X :
  - 10.10 (2014) : utilise CUBIC il paraît, window scaling, ...
- Android : basé sur linux
- Google : BBR actuellement

TCP : Tahoe

Utilise SS + CA + FRet

cwnd <-- 1 (MSS)

ssthresh <-- Init_Ssthresh;

State <-- Slow Start;

ACK received in Slow Start:

    cwnd <-- cwnd +1 (MSS); // "exponential"

    If cwnd > ssthresh Then

        State <-- Congestion Avoidance;

ACK received in Congestion Avoidance:

    If (#ack received = cwnd) Then

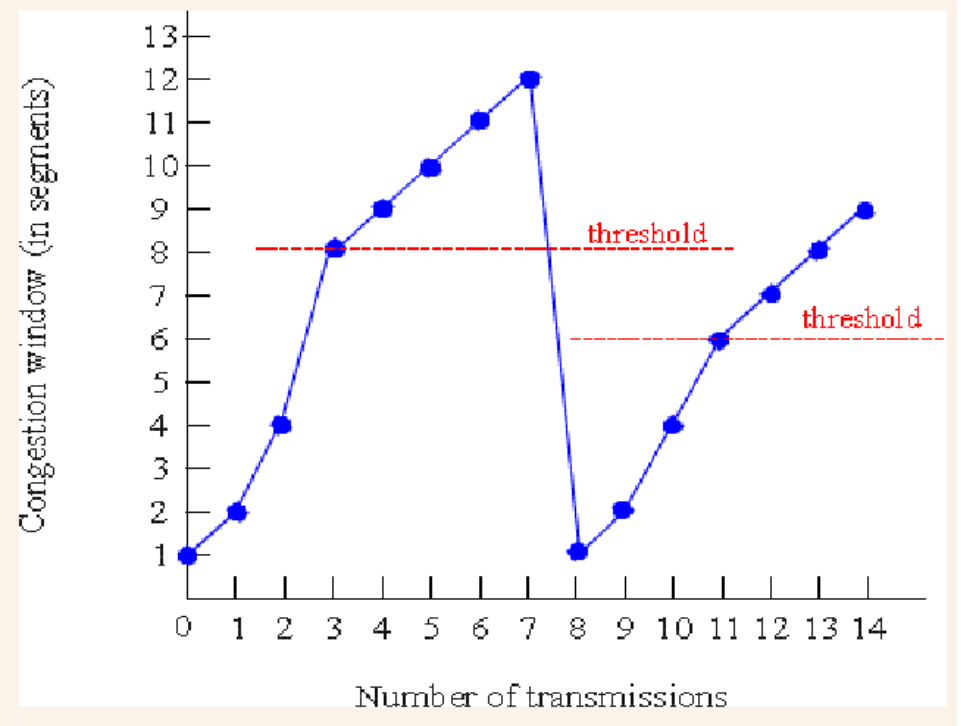
        cwnd <-- cwnd +1 (MSS); // "linear" increase of cwnd

Timeout expiration OR 3rd DupACK received:

    Retransmit (lost packet);

    ssthresh <-- cwnd/2; cwnd <-- 1(MSS);

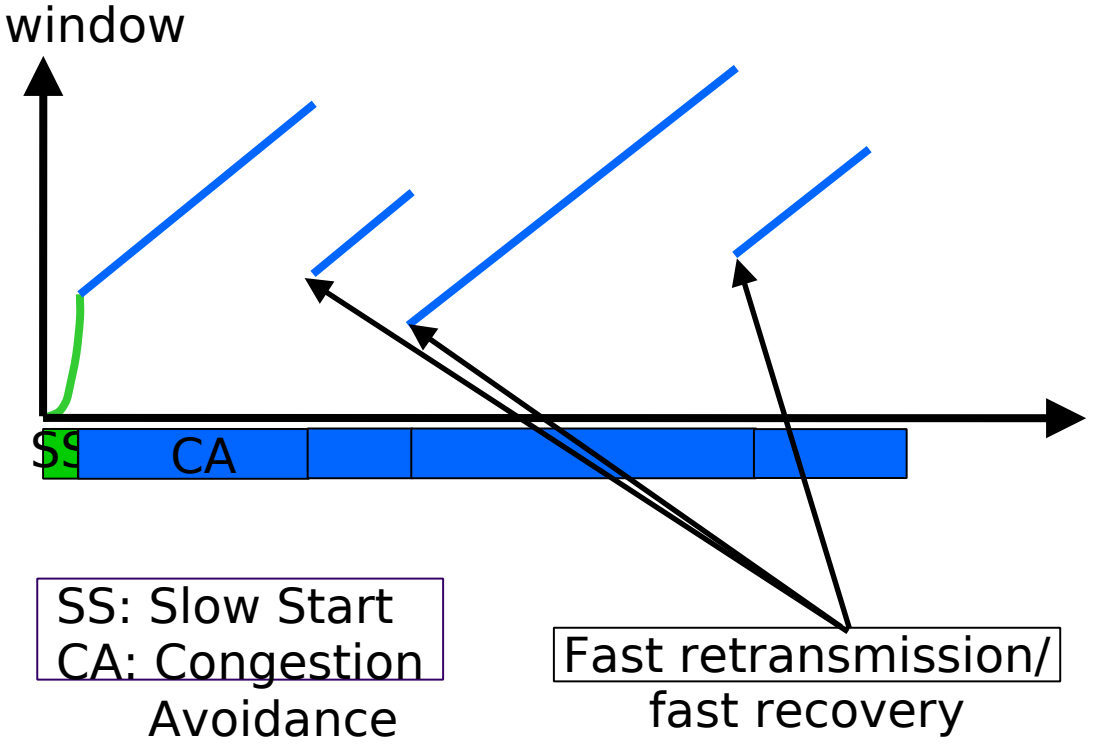
    State <-- Slow Start;

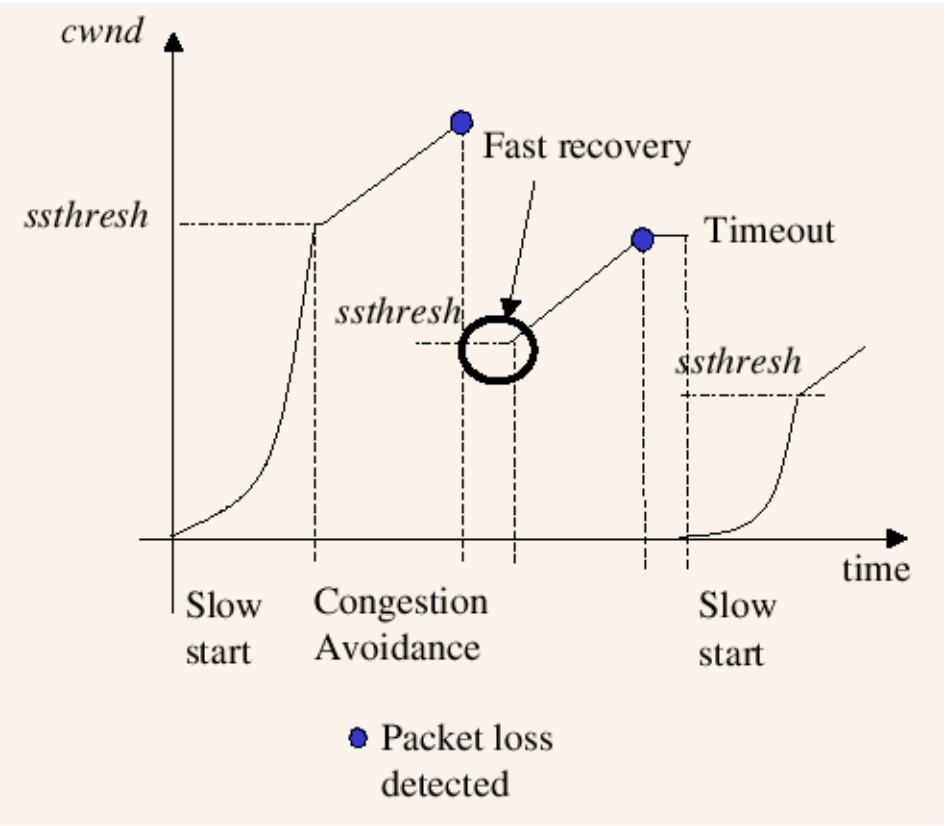


TCP : Reno

- Reno = Tahoe + fast recovery
- Récupération plus rapide après 3 dupacks

source



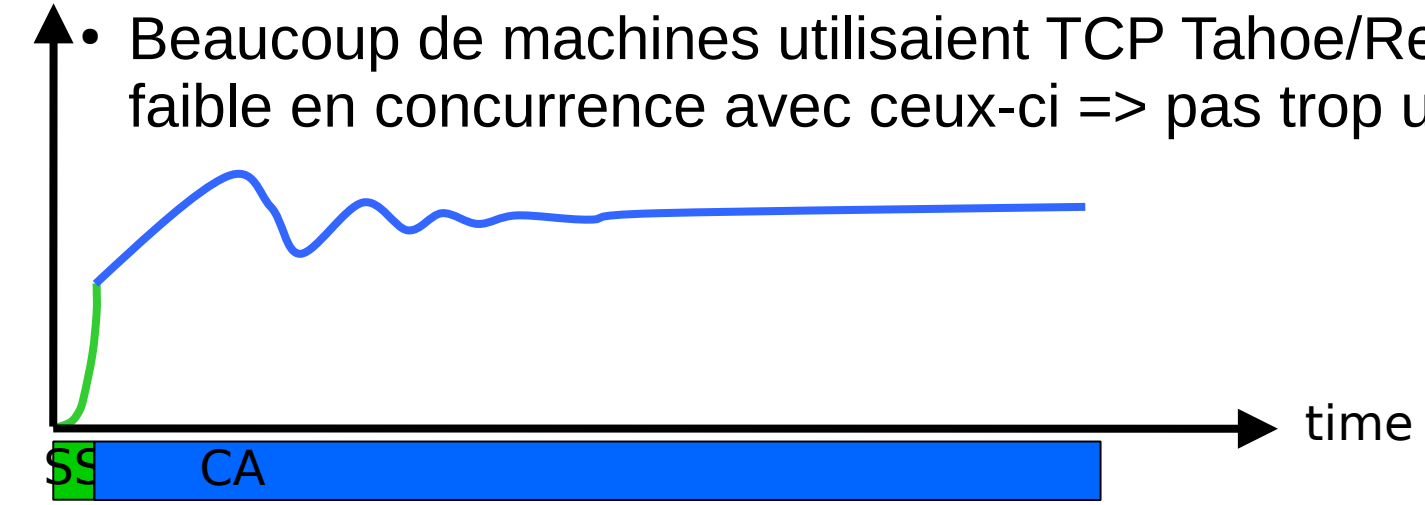


TCP : Newreno

- Reno + légère modification de fast recovery
- Meilleur si plusieurs pertes non contigus dans un même “vol” de paquets
  - Reno : cwnd peut décroître plusieurs fois dans un même RTT, alors que c'est un seul événement de congestion !
- Lors de la réception d'un accusé partiel (qui accuse une partie des paquets envoyés), envoie le paquet suivant perdu tout de suite
- Reste en fast recovery jusqu'à la réception des accusés de **tous** les paquets perdus dans le “vol” de paquets initial
- => Meilleur, car il coupe cwnd seulement après la 1ère perte (une seule fois donc)

TCP : Vegas

- But : réduire le débit **avant** qu'une perte apparaisse
- L'historique du RTT donne des informations sur l'évaluation de la taille des buffers des routeurs
- cwnd régulé par le temps d'arrivée des accusés et par le RTT courant vs. moyen
  - intervalles 0..alfa..bêta..
  - (TCP classique régulé uniquement par la réception des accusés de réception)
- Beaucoup de machines utilisaient TCP Tahoe/Reno, et il est plus faible en concurrence avec ceux-ci => pas trop utilisé



TCP : option window scaling

- RFC 1323
- Option permettant au récepteur de spécifier des valeurs plus grandes à sa fenêtre de réception
- Le champ Window est sur 16 bits => 2^16 = 64ko de fenêtre de réception => limitation du débit, combien ?
- Pendant le paquet syn+ack, spécifie un nombre
- La valeur réelle est la champ * 2^nombre

TCP : option timestamps

- RFC 1323
- Permet, entre autres, d'obtenir avec précision le RTT d'un paquet
  - en cas de paquets reçus en désordre
  - lors de retransmission (à quel paquet se réfère un ack ?)
  - sur plusieurs paquets, permet de savoir si la congestion est sur le chemin ascendant ou descend
- Option de TCP, 10 octets
- Émetteur et récepteur :
  - insèrent le timestamp dans chaque paquet au moment de l'envoi
  - renvoient le timestamp de l'autre extrémité

TCP : option SACK

- “Selective ACKs”
- Implémenté comme option TCP dans Reno par ex.
  - négocié lors de l'initiation de la connexion
- Récepteur précise les paquets reçus :
  - début et fin de chaque bloc de segments contigus reçus
- Connaissance du numéro du ou des paquets à **ne pas** retransmettre
- DSACK (Duplicate SACK) : paquet reçu plusieurs fois

TCP : algorithme ABC

- ABC : Appropriate Byte Counting (1999, 2003, rfc 3465)
- TCP classique :
  - en SS ou CA : augmentation à chaque accusé reçu
  - mais si un accusé est perdu et le prochain accusé accuse deux paquets, l'augmentation est faite une seule fois !!
  - aussi, si le récepteur utilise des accusés retardés, l'émetteur augmente le transfert plus lentement
- ABC : utiliser le nombre d'octets accusés au lieu du nombre d'accusés reçus

Réseaux sans fil : caractéristiques

- Filaire :
  - 99.9... % : perte due à une congestion d'un routeur
  - le reste : perte due à un problème matériel
- Non filaire :
  - beaucoup d'interférences, donc des pertes
  - lors d'une perte, TCP considère qu'il s'agit d'une congestion, donc il réduit le débit, le contraire de ce qu'il doit faire

TCP : Westwood+

- Mémorise les RTTs
  - estime la bande passante disponible
- Lors d'une perte, se remet à la valeur précédente de cwnd
  - (TCP classique réduisait par 2 le cwnd)
- Mieux adapté que les autres aux réseaux sans-fil

Réseaux à grand cwnd : caractéristiques

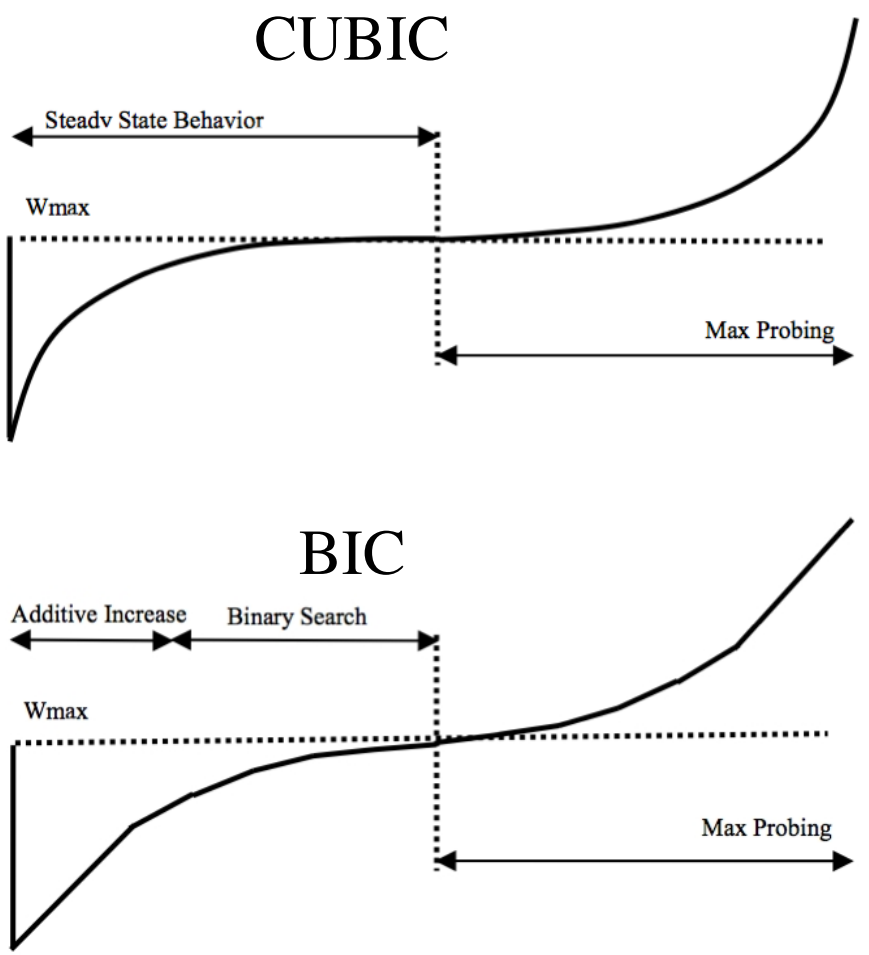
- Grand cwnd = grand bande passante * délai
- Ex. : 10 Gb/s et RTT de 100ms
- L'augmentation linéaire de cwnd est trop faible et la baisse 0.5 est trop importante
- Protocoles nouveaux : HighSpeed TCP, BIC, ...
- HighSpeed TCP, constante low_window :
  - si cwnd <= low_window, TCP classique
  - si cwnd > low_window, l'augmentation est plus grande et la baisse plus petite, en fonction du nb pertes, cwnd etc.
    - 1 => f(cwnd), 0.5 => f(cwnd)

TCP : BIC

- Adapté aux réseaux à grand cwnd
- Fonction d'augmentation de cwnd modifiée pour cwnd > certaine valeur (e.g. 100 Mb/s)
- Lors d'une perte, l'actuelle cwnd devient max et la nouvelle cwnd devient min
- Recherche binaire : cwnd suivante est (min+max)/2 jusqu'à perte d'un paquet
- Si max–min trop grand, augmentation linéaire jusqu'à une certaine différence

TCP : CUBIC

- Similaire à BIC, mais utilise une fonction cubique (polynôme degré 3), qui a une partie concave et une convexe
- La portion concave entraîne l'augmentation rapide de la cwnd, ensuite la portion convexe permet de stabiliser le réseau



"CUBIC: A New TCP-Friendly High-Speed TCP Variant"
Injong Rhee and Lisong Xu, PFLDnet 2005

TCP : Compound TCP (CTCP)

- Adapté aux réseaux à grand cwnd
- Utilise la somme de deux fenêtres :
  - cwnd classique de Reno
  - dwnd, une fenêtre basée sur le délai des paquets (dérivée de Vegas)
- SS non changé

TCP : algorithme PRR (Proportional Rate Reduction)

- S'applique à une version TCP, en lui modifiant l'étape fast recovery, rfc6937 (2013)
- En fast recovery, l'émetteur attendait cwnd/2–3 ACKs avant d'envoyer un paquet, ensuite envoyait 1 paquet par ACK reçu
- Mais certains CC ne réduisent pas cwnd par 2 (e.g. CUBIC réduit moins), et PRR utilise en algorithme pour faire baisser cwnd jusqu'à cette valeur
- => Récupère plus rapidement, ce qui est important surtout pour des flux courts (Web)

Réseaux à bande passante asymétrique

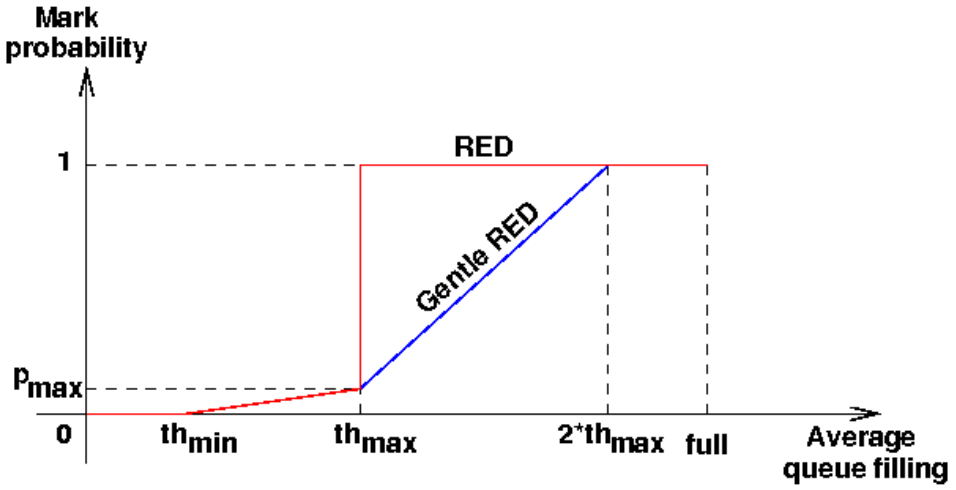
- Ex. : ADSL
- TCP n'est pas adapté

AQM, gestion des files d'attente des routeurs

- La file d'attente des routeurs peut se remplir
- “Statiques”, e.g. DropTail : le paquet est rejeté ssi la file est pleine
  - très rapide
  - le plus utilisé
  - génère rafale de pertes (et de retransmissions aussi)
  - parfois le(s) même flux est pénalisé
  - synchronisation globale : les flux baissent et augmentent en même temps
  - avoir des files d'attente pleines augmente les délais
- AQM (*Active Queue Management*) : mesure régulièrement l'occupation de la file
  - RED : Random Early Detection
  - ECN : Explicit Congestion Notification
  - CoDel : Controlled Delay

AQM : RED

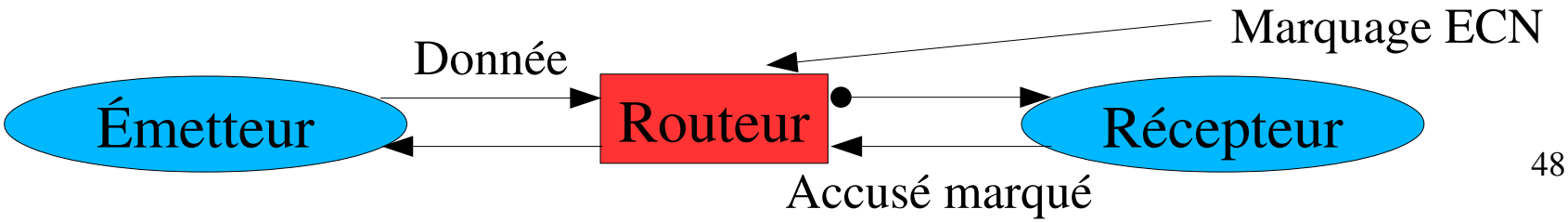
- But : avertir l'émetteur que la file est trop remplie
  - rejeter quelques paquets **avant** que la file ne soit pleine
  - les émetteurs réduisent leur débit, comme si perte
- Taille = moyenne sur les N dernières sec.
- Deux seuils, th_{min} et th_{max}
  - si taille<th_{min}, le paquet passe
  - si th_{min}<=taille<th_{max}, le paquet est rejeté/marqué avec une probabilité dépendante de taille
  - si taille>=th_{max}, le paquet est rejeté



- Ex de paramètres (Floyd, Jacobson) : full = 100, th_{max} = 15, th_{min} = 5, maxp = 1/50, wq = 0.002
- Variante : gentle red

AQM : ECN

- But : avertir l'émetteur **sans** perte de paquet
- Si un flux est compatible ECN, le routeur marque, sinon il rejette (rfc 3168, page 9, par. 3 et 4 pas clair, voir autres articles)
- Activé seulement si lors de l'initialisation, les deux machines ont spécifié (dans l'en-tête TCP) être capables ECN
  - pas utilisé avec UDP
- Pendant le transfert d'un flux compatible ECN :
  - si congestion, le routeur marque le paquet (modifie 2 bits de l'en-tête IP) (les accusés ne sont jamais marqués)
  - le récepteur : si paquet marqué reçu, il marque chaque accusé jusqu'à réception de CWR
  - l'émetteur : si accusé marqué reçu, il répond par CWR (Congestion Window Reduced)



AQM : CoDel

- Implémenté en 2012 dans le noyau Linux
- En train d'être standardisé (RFC)
- Réduit le délai dans la file d'attente (plusieurs ordres de magnitude !) par rapport à RED et autres
- Pas de paramètre à configurer

Algorithme :

list of intervals for reduction: 100/sqrt(i), i>0
interval = 100 ms
for each interval (all its packets) do
  if the lowest queueing delay > 5 ms
    drop last packet
    reduce interval
  else
    interval = 100 ms

DCTCP (Data Center TCP)

- Voir https://tools.ietf.org/html/draft-ietf-tcpm-dctcp-02 (2010/2016)
- Problème des data centers : lire Introduction (dctcp.txt)
- Idée : modifier ECN pour donner pas seulement l'info si congestionné ou non, mais aussi le degré de congestion, et l'émetteur réduit le débit plus ou moins, en fonction de ce degré
- Tout le réseau doit être ECN, l'émetteur et le récepteur doivent être modifiés (déployable, car c'est dans un data center)
- Implémentation dans linux: [git commit]
  - exemple : latence passe de 4 ms (CUBIC) à 0.04 ms (DCTCP)

BBR (Bottleneck Bandwidth and Round-trip propagation time)

- Voir https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00 (2017)
- Congestion = plus de données injectées dans le réseau que le réseau n'accepte
- De nos jours, la perte de paquet n'est plus équivalent à congestion :
  - grands buffers : la congestion apparaît avant la perte (cf. bufferbloat)
  - petits buffers : perte apparaît pour des flux temporaires
- Idée : utilise la perte mais aussi la différence de RTT (comme Vegas)
- Différent des autres TCP (plus de paramètres)
  - centré sur deux variables : RTprop (RTT "idéal" : queues=0, pas de delayed ack etc.) et BtlBw (Bottleneck Bandwidth)
  - un chemin est assimilé à un lien avec un certain RTT et le débit du goulot d'étranglement
- "BBR is being deployed on Google.com and YouTube video servers" (02/2017)
- "Even so [youtube does video adaptation], BBR reduces median RTT by 53% on average globally and by more than 80% in the developing world"

Support des systèmes d'exploitation

- linux : /proc/sys/net/ipv4/
  - tcp_available_congestion_control : cubic, reno, highspeed, vegas, westwood etc.
  - tcp_congestion_control : cubic
  - tcp_ecn, tcp_timestamps, tcp_sack, tcp_dsack
  - plus d'info : man tcp et man 7 ip ou https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/ip-sysctl.txt
- linux : /etc/rc.local
  - echo 1 >/proc/sys/net/ipv4/tcp_ecn
- Windows XP : http://support.microsoft.com/kb/314053
- Code source linux de TCP : https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/net/ipv4/tcp_input.c

Choix du CC sur les machines

- Tout ce que nous avons vu du CC s'applique à l'émetteur des données
- Quels sont les émetteurs de données sur Internet ?
  - serveurs
  - machines personnelles : cas où elles émettent :
    - méls avec pièces jointes
    - sites Web à maintenir
    - stockage de données dans le cloud, réseaux sociaux
    - sur ma machine, 7% du trafic est sortant et 93% est entrant (cf. ifconfig, exemple non représentatif)

Conclusions

- TCP adapte le débit des données envoyées à la bande passante disponible
- Les flux TCP sont équitables
- TCP n'est pas adapté à tous les types de réseau et tous les types d'applications, de nouvelles versions apparaissent
- Plusieurs algorithmes de CC en TCP existent, pouvant être groupés suivant plusieurs critères en :
  - adaptés aux réseaux sans fil, aux LFN, aux data centers, ...
  - utilisent la perte et/ou le délai des paquets comme information sur la congestion
  - plus différents algorithmes/options pour les améliorer
- Quatre phases lors d'un transfert TCP : slow start, congestion avoidance, fast retransmit, fast recovery
- RED + ECN permet d'avertir l'émetteur du début d'une congestion sans perte de paquet