# Le contrôle de congestion dans l'Internet

Chaput Emmanuel

INP TOULOUSE ENSEEIHT

2023-2024



## Plan





## Le problème





## Qu'est-ce que la congestion ?





## Qu'est-ce que la congestion ?

**Qu'est-ce que la congestion ?**

Phénomène se produisant sur un équipement réseau soumis à une quantité de trafic pour laquelle il n'est pas dimensionné et entrainant une déterioration du service rendu.

- Quelles causes ?
  - Confluence de trafics
  - Débit d'entrée supérieur à capacité de sortie
  - Bursts de trafics sporadiques
  - Équipement sous-dimensionné
- Quelles conséquences ?
  - Accroissement du temps de traversée (délai)
  - Variation du temps de traversée (jigue)
  - Pertes



## Détection et réaction

Qui est concerné par la congestion ?

- Les équipements du cœur du réseau
  - Les routeurs dans le cas IP
  - Accroissement des files d'attente
  - Conséquence du trafic en transit
- Les équipements d'extrémité
  - Les applications qui soumettent le trafic
  - Pas de vision directe de l'état du réseau
  - Variations de la qualité de service perçue

Qui peut la détecter et la traiter ?

- Les équipements de cœur peuvent la détecter
  - Observation des files
  - Anticipation possible
- Les équipements d'extrémité peuvent y réagir
  - Modulation du trafic
  - Renégociation éventuelle de SLA

## Quelle gestion de la congestion ?





## Quelle gestion de la congestion ?

- Mécanismes préventifs
  - Dimensionnement des équipements
  - Admission/refus des communications (SLA)
  - Filtrage des trafics
  - ...
- Mécanismes réactifs
  - Adaptation des trafics soumis
  - Re routage
  - ...
- Mécanismes curatifs
  - Détruire les paquets (gestion *Drop Tail* des files d'attente)
  - ...



## Quelle gestion dans la pile IP ?





## Quelle gestion dans la pile IP ?

- Un message ICMP
  - *Source Quench* émis par un routeur saturé
  - La source réduit le débit (taille de fenêtre par exemple [24])
  - Consommateur de ressources !
  - Peu implanté
- Historiquement
  - Évolutions algorithmiques de TCP
  - TCP Tahoe, Reno, NewReno
  - C'était le principal "responsable"
- Prise en compte dans IP
  - Notification explicite (*Source Quench piggy-backé*)
  - Gestion active des files d'attente
  - ...
- Ingénierie de trafic
  - Fondée sur l'expérience du circuit virtuel
  - MPLS



## La gestion de la congestion par TCP





## Quel est le problème avec TCP ?

En Octobre 1986, Van Jacobson identifie un problème

- Un lien de 32 kbps reliant deux IMP ne fournit parfois que 40 bps
- Il suspecte le *comportement* de 4.3BSD
- Il analyse le problème et propose des solutions

**Le problème identifié**

Même en présence de connexions *a priori* à l'équilibre (un paquet n'entre dans le réseau que si un paquet en sort), le système n'est pas robuste face aux phénomènes de congestion, contrairement à la théorie.

Les axes proposés par Van Jacobson

- Atteindre l'équilibre
- Maintenir l'équilibre
- S'adapter aux évolutions du réseau

## Les mécanismes de TCP Tahoe

Papier fondateur de TCP [18]

- *Slow Start*
  - Accroissement progressif (rapide) du nombre de segments en transit
  - Bénéficier du *self clocking* assuré par le contrôle de flux implicite.
  - Pour atteindre un équilibre
- Estimation du temps d'aller-retour
  - Pour maintenir l'équilibre
  - Estimateur précédent [26] peu efficace
  - Besoin de prendre en compte les variations
- *Congestion Avoidance*
  - S'adapter aux évolutions du réseau
  - Pour maintenir l'équilibre

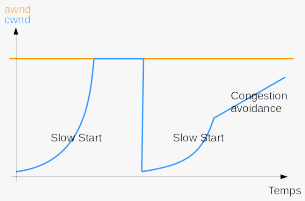


## Les mécanismes de TCP Tahoe

- Le contrôle de flux est assurée par une limite sur la taille de fenêtre de l'émetteur
  - Variable `awnd` (*Advertised Window*)
  - Valeur transmise par le récepteur
- Pour contrôler les congestions, ajoutons une nouvelle limite
  - Introduction d'une variable `cwnd`
  - Comment déterminer sa valeur ?
- À aucun moment, l'émetteur ne peut dépasser ces deux limites, si bien que sa taille de fenêtre utilisable est
  - `uwnd = min(awnd, cwnd)`



## Les mécanismes de TCP Tahoe




- Objectifs du slow start
  - Commencer prudemment
  - Atteindre au plus vite l'optimal
- Objectifs du congestion avoidance
  - Ne pas (re)provoquer une congestion



## Comment détecter une congestion ?

Comment une entité TCP peut détecter un phénomène de congestion ?

- Principes déjà évoqués par ailleurs [20]
  - Besoin d'un mécanisme de signalisation sur les éléments actifs
  - Mise en œuvre d'une politique d'adaptation du trafic aux extrémités
- Choix faits dans TCP *Tahoe*
  - Une perte est la conséquence d'une congestion
  - Un déséquencement est la conséquence d'une perte
- Les pertes doivent donc être détectées le plus tôt possible
  - Il ne s'agit plus (uniquement) de retransmettre
  - Il faut également éviter/arrêter une congestion
  - Nécessite d'une bonne estimation du temps d'aller-retour
- Oui, mais . . .
  - . . . et en cas de perte suite à une erreur de transmission ?
- Effet de bord troublant
  - Pour un bon fonctionnement, TCP a "besoin de pertes" !



## L'estimation du temps d'aller-retour





## L'estimation du temps d'aller-retour

- Les timers de TCP fonctionnent mal [36]
  - Utilisation d'une moyenne mobile [26]
    - $RTT \leftarrow \alpha.RTT + (1 - \alpha).m$
    - $RTO = \beta.RTT$
    - $\alpha = 0.9, \beta = 2$ par exemple
  - Satisfaisant pour une charge faible ($< 0.3$)
  - En cas de montée de la charge, retransmissions supplémentaires !
- Prise en compte de la variation du délai [6]
  - $Err \leftarrow m - RTT$
  - $RTT \leftarrow RTT + \alpha.Err$
  - $MD \leftarrow MD + \beta.(|Err| - MD)$
  - $RTO \leftarrow RTT + \gamma.MD$
  - $\alpha = \frac{1}{8}, \beta = \frac{1}{4}, \gamma = 4$ par exemple (implantation en entiers)

## Le *Fast Retransmit*





## Le *Fast Retransmit*

- Principes de base
  - Le récepteur TCP envoie un accusé de réception ("dupliqué") à chaque segment hors séquence
  - Un accusé de réception dupliqué est donc le signe, pour l'émetteur
    - D'un segment hors séquence
    - D'une perte
  - Après plusieurs ACK dupliqués
    - Perte probable
    - Congestion
- Mise en œuvre
  - Introduction d'un compteur `ndup`
  - Réception d'un ACK nouveau
    - `ndup = 0`
  - Réception d'un ACK dupliqué
    - `ndup = ndup + 1`
  - Lorsque `ndup == 3`
    - Retransmission du premier segment non acquitté



## Le slow start



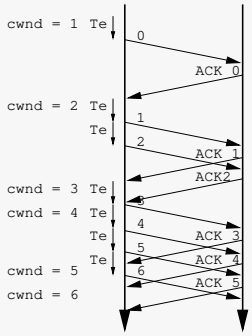


## Le Slow Start

- Principes
  - Commencer en émettant une fenêtre d'un seul segment
  - Doubler la taille à chaque nouvelle fenêtre
  - Respecter le contrôle de flux (`awnd`) !
- Mise en œuvre (sur l'émetteur)
  - Initialisation
    - `cwnd := 1`
  - À chaque ACK reçu
    - `cwnd := cwnd + 1`
  - A tout moment, fenêtre utilisable
    - `uwnd := min(awnd, cwnd)`



## Le Slow Start, un exemple






## La prévention de congestion

## Le mécanisme de *Congestion Avoidance*

- Accroissement additif de la taille de la fenêtre
- De la forme `cwnd := cwnd + 1/cwnd` (incrément de `cwnd` d'une unitée après `cwnd` ACKs reçus)
- En présence de congestion
  - Décroissance multiplicative de la taille de la fenêtre
  - `cwnd := cwnd / 2`
- Comportement qualifié de AIMD
  - *Additive Increase, Multiplicative Decrease*
- Habituellement implanté conjointement au *Slow Start*



## Implantation conjointe des mécanismes



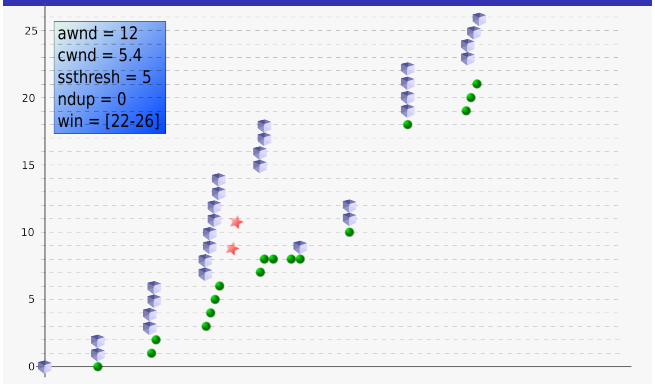


## Mise en œuvre conjointe dans TCP Tahoe

- Introduction d'un seuil `ssthresh`
  - *Slow Start Threshold*
  - Délimite la frontière entre *Slow Start* et *Congestion Avoidance*
- Initialisation
  - `cwnd := 1`
  - `ssthresh := awnd`
- Réception d'un nouvel ACK
  - `ndup = 0`
  - Si `cwnd < ssthresh` alors `cwnd = cwnd + 1`
  - Sinon `cwnd := cwnd + 1/cwnd`
- Réception d'un ACK dupliqué
  - `ndup := ndup + 1`
- Détection d'une perte (*TimeOut* ou `ndup ==3`)
  - Retransmission du premier segment non acquitté
  - `ssthresh := min(awnd, cwnd) / 2`
  - `cwnd := 1`



## TCP Tahoe : un exemple






## TCP Reno et la reprise rapide





## TCP Reno et le *Fast Recovery*

Fin Avril 1990, Van Jacobson fait le constat suivant

- Trois accusés de réception dupliqués sont le symptôme d'une congestion *modérée*
  - Le récepteur reçoit des segments
- Le retour en *Slow Start* est inutile
  - Application du *Multiple Decrease*
    - `cwnd := ssthresh` (et non plus `cwnd := 1`)
- On peut ajouter transitoirement à `cwnd` la valeur de `ndup`
  - `ndup` est le nombre de segments sortis du réseau
  - Rythmer les émissions sur les accusés de réception dupliqués

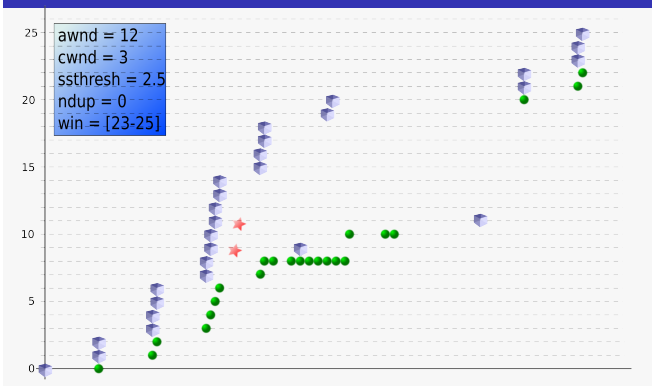
## TCP Reno et le *Fast Recovery*

Mise en œuvre du *Fast Recovery*

- Après trois accusés de réception dupliqués
  - Retransmission du dernier segment non acquitté (*Fast Retransmit*)
  - `ssthresh := min(awnd, cwnd) / 2`
  - `cwnd := ssthresh`
  - `uwnd := min(cwnd + ndup, awnd)`
- A la réception d'un nouvel accusé de réception
  - `ndup := 0` (comme d'habitude)
  - `uwnd := min(cwnd, awnd)` (fin du *Fast Recovery*)



## TCP Reno : un exemple






## TCP New Reno



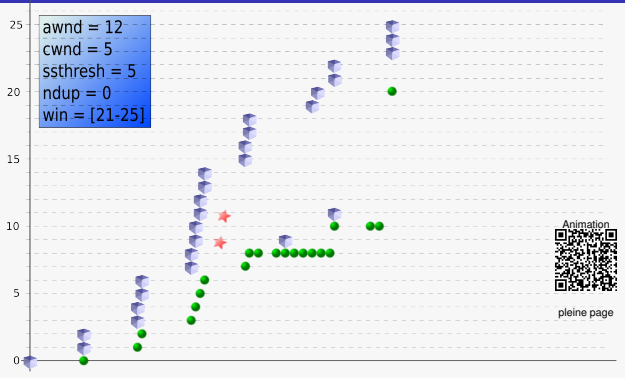


## TCP New Reno

- TCP Reno a un "petit défaut"
  - En cas de pertes multiples dans une fenêtre
  - En l'absence de l'option SACK
  - Risque de retomber en *Slow Start*
- Lors d'un *Fast Recovery*, TCP New Reno distingue deux types de nouvel ACK
  - Un ACK *total* accuse réception de tout ce qui a été émis avant la retransmission
  - Un ACK *partiel* n'accuse réception que d'une partie
- Un ACK partiel est le symptôme d'une (nouvelle) perte
  - Retransmission du premier segment non acquitté
  - `uwnd = min(awnd, cwnd + ndup - nback + 1)` où `nback` est le nombre de segments acquittés
  - Rester en *Fast Recovery*



## TCP NewReno : un exemple






## Les évolutions de TCP

## Les problèmes avec TCP





## Les problèmes avec TCP

On distingue traditionnellement trois grandes phases dans TCP :

- L'initialisation de la connexion
  - Établissement en trois phases, négociation des options
  - Premier aller-retour
- Le démarrage
  - Accélération depuis un *Send and Wait* vers un protocole à fenêtre
  - Croissance exponentielle de `cwnd`
- Le comportement long terme
  - Adaptation aux évolutions du réseau
  - Croissance linéaire de `cwnd`



## L'initialisation de la connexion

- Un temps d'aller-retour, c'est pénalisant
  - Surtout pour les connexions courtes (majoritaires)
  - Pas de données avec les SYN
  - Mal nécessaire ?
- Profiter d'informations d'une connexion précédente
  - Obtenir des paramètres plus pertinents (PMTU, RTT, ...)
  - S'authentifier par des *cookies*
  - ...
- Utiliser une seule connexion
  - Connexions persistantes (*Keep Alive*) de HTTP (par défaut dans HTTP 1.1)



## Le démarrage

- Le slow start peut être très pénalisant
  - Temps d'aller retour long
  - Connexions courtes
    - 90 % des connexions contiennent moins de 10 segments
- Envoyer des données plus tôt
  - T/TCP [2, 3], TCP Fast Open [27]
  - Problèmes de sécurité
- Commencer avec une taille de fenêtre plus grande
  - En 2002, la RFC 3390 préconise une taille de 3 segments (sur une MTU de 1500)
  - En 2013, la RFC 6928 (experimental) suggère de passer à 10



## Le comportement long terme

- TCP fonctionne relativement bien sur des réseaux à faible débit avec peu de pertes
- Il est bien moins performant sur les LFN
  - *Long Fat Network* (produit débit par RTT elevé)
  - Accroissement du débit par *Congestion Avoidance*
    - De nombreux RTT
    - Des RTT longs
- De nouvelles propositions
  - On cherche à être TCP *friendly* là où TCP New Reno est performant
  - Au delà, on ne se prive pas d'être plus aggressif
  - Par exemple TCP compound et TCP Cubic
- Il est fondé sur les pertes !
  - Est-ce bien raisonable ! ?



## Utilisation d'une connexion unique

## Utilisation d'une connexion unique

- Communications en série
  - Connexions persistantes (*Keep Alive*) de HTTP (par défaut dans HTTP 1.1)
  - Ne permet pas de profiter de tout le débit disponible
- Communications en parallèle
  - Exemple de SPDY (une proposition pour HTTP 2.0) [15]
  - Problème du *Head of Line Blocking*
- Refonte du protocole de transport
  - Par exemple QUIC [31] qui vise à optimiser l'utilisation de SPDY
  - Le protocole SCTP initialement prévu pour de la signalisation téléphonique [32]



## S'adapter aux performances des réseaux modernes





## TCP Cubic





## TCP CUBIC

Évolution de TCP BIC, elle propose un nouveau mécanisme d'évitement de congestion [16]

- En trois phases autour de la valeur de CWND suspecte
- Sans dépendance vis à vis du RTT

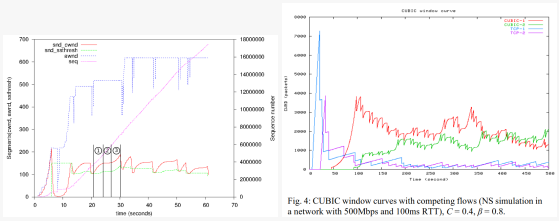


Fig. 4: CUBIC window curves with competing flows (NS simulation in a network with 500Mbps and 100ms RTT), $C = 0.4$, $\beta = 0.8$.

Figure de droite tirée de [16].



## Principes de TCP Cubic

- Supposons une congestion présumée avec une taille de fenêtre $W_{max}$
  - Détectée par une perte (timer ou `ndup = 3`)
- Diminution multiplicative pour résorber la congestion
  - $W_{min} = (1 - \beta).W_{max}$
- À la fin de la congestion, on repart
  - $W(t) = C.(t - K)^3 + W_{max}$ où
    - $t$ est le temps écoulé depuis la congestion
    - $C$ est un paramètre
    - $K$ est le temps nécessaire pour atteindre $W_{max}$ que l'on calcule par $W(0) = W_{min}$, ce qui donne $K = \sqrt[3]{\frac{\beta.W_{max}}{C}}$
- D'où la forme de la figure précédente
- Pas de dépendance vis à vis du RTT
- $C$ choisi pour rester TCP friendly lorsque nécessaire



## TCP Compound

## Principes TCP Compound

- Objectifs similaires à ceux de TCP Cubic
  - Être TCP friendly sur des réseaux saturés
  - Être plus agressif sur des réseaux sous utilisés
- Contrôle de congestion implanté dans Windows depuis Vista [33]
- Composé de deux approches
  - L'une fondée sur les pertes (à la TCP Reno)
  - L'autre sur les délais (à la TCP Vegas)
  - La première est toujours active (il est aussi agressif que TCP Reno)
  - La deuxième permet d'être plus agressif si le réseau semble sous utilisé
- L'idée de base est que la fenêtre *win* évolue de la façon suivante (la détection de congestion étant fondée sur le délai)
  - $win \leftarrow win + \alpha.win^k$ sans congestion
  - $win \leftarrow win.(1-\beta)$ en cas de congestion



## Fonctionnement de TCP Compound

- Nouvelle expression de la taille de fenêtre utilisable
  - $uwnd = min(awnd, cwnd + dwnd)$
- Où *dwnd* évolue en fonction du délai
  - En tenant compte de *cwnd* qui perdure
  - *dwnd* n'est actif que si $win > W_{low}$
- Pour cela, on évalue l'occupation du réseau via la différence entre les débits espéré et constaté
  - $diff = (\frac{win}{baseRTT} - \frac{win}{RTT}).baseRTT$
- On agit en fonction d'un seuil $\gamma$ permettant d'anticiper une congestion
  - Si $diff < \gamma$, alors *dwnd* incrémenté de $max(\alpha.win^k - 1, 0)$
  - Si $diff \geq \gamma$, alors *dwnd* décrémenté de $\zeta.diff$
  - En cas de perte (fast retransmit) *dwnd* reçoit $(1-\beta).win - cwnd/2$



## Fonctionnement de TCP Compound

- La borne *dwnd* adopte un comportement AIMD
  - *Additive Increase Multiple Decrease*
  - Fondé sur une observation du réseau
  - Sans dépendance au RTT
- Cela permet une plus grande agressivité
  - Lorsque l'état du réseau le permet
- Permet de respecter TCP New Reno et d'être au moins aussi efficace
  - *dwnd* n'est jamais négatif
  - $\zeta$ permet d'être plus ou moins TCP friendly
- Proposition de paramètrage fondé sur une analyse
  - $k = 0.75, \alpha = 0.8, \beta = 0.5$
  - $\gamma$ défini dynamiquement
  - $\zeta$ non décrit



## En finir avec la dépendance aux pertes





## Le problème





## Le problème

- Le comportement long terme de TCP Reno est instable
  - Algorithme *Additive Increase Multiple Decrease*
- Il suppose que les caractéristiques du réseau sont stables
  - Temps d'aller retour
  - Débit disponible
- Il est sensible à la taille des files d'attente
  - Trop courtes, il va ralentir inutilement
  - Trop longues, il va les remplir et subir un long délai
- Le seul outil pour moduler le débit est la perte
  - Perte = congestion
  - Est-ce encore toujours vrai ?
  - Est-il nécessaire d'en arriver là ?

## Quelle gestion des files d'attente ?

3 Les évolutions de TCP
- En finir avec la dépendance aux pertes
  - Le problème
  - Quelle gestion des files d'attente ?
  - TCP BBR



## Quelle gestion des files d'attente ?

- Une file d'attente d'un routeur peut être
  - Vide : c'est peut-être le signe que le lien n'est pas utilisé à pleine capacité
  - Partiellement occupée : du retard est induit, et la situation risque d'être instable car le lien est peut-être trop sollicité
  - Pleine : des paquets sont perdus, la situation n'est pas bonne !
- TCP Reno vise une occupation partielle
  - Il accélère (doucement) pour ne pas sous-utiliser les liens
  - Il ralentit (fortement) lorsqu'une perte lui montre qu'une file est pleine
- TCP CUBIC fait finalement pareil
  - Il ralentit moins fort et accélère plus fort pour plus de réactivité
  - La limite reste matérialisée par une perte

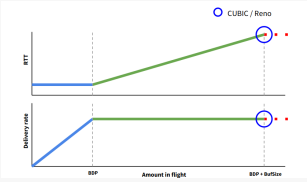


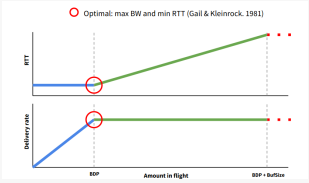
## Quelle gestion des files d'attentes ?

- Le point d'équilibre idéal est le tout début du remplissage de la file [5]
  - Débit maximal et délai minimal
  - Le lien est utilisé à pleine capacité
  - Il y a parfois quelques paquets dans la file
  - La file n'introduit aucun retard
  - Pas de risque de perte
- Comment déterminer cet état ?
  - Risque de sous utiliser le lien
  - Pas de perte pour détecter un excés
- Idée : observer le RTT
  - Une valeur qui croît est signe d'une file qui se remplit
  - Idée déjà présente dans TCP Vegas [4]



## Quelle gestion des files d'attentes ?




- Le point de fonctionnement de Cubic et Reno
- Identifié par des pertes

- Le point de fonctionnement idéal
- Comment l'identifier ?

*Figures tirées de [5]*

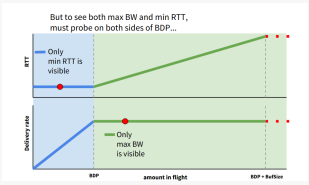


## TCP BBR

3 Les évolutions de TCP
- En finir avec la dépendance aux pertes
  - Le problème
  - Quelle gestion des files d'attente ?
  - TCP BBR



## TCP BBR : le point de fonctionnement idéal




- Débit maximal
- Délai minimal
- Il est nécessaire de mesurer les deux

*Figure tirée de [5]*

## TCP BBR : les principes

- Ne plus se fonder sur les pertes, mais sur l'état du réseau
  - Maintenir les files d'attente dans un état optimal
  - Maximiser le débit
  - Minimiser le délai
- Mesure permanente du RTT
  - Conservation de la valeur minimale sur une fenêtre de quelques dizaines de secondes
- Mesure permanente du débit maximal
  - Mesuré sur le récepteur sur une fenêtre de 6 à 10 RTT
- Transmission des données
  - Espacées pour ne pas dépasser le débit max
  - Les accusés de réception des données non limitées par l'application sont utilisés pour mettre à jour les paramètres



## TCP BBR : tentative d'accélération

- Périodiquement on augmente la taille de la fenêtre
  - Sur un RTT, le volume transmis de 25 % (du BDP)
- Si la mesure des paramètres montre un accroissement du débit
  - Cela signifie que les liens étaient sous utilisés
  - On était en dessous du point de fonctionnement optimal
  - On reste à ce nouvau débit
- Si la mesure des paramètres montre un accroissement du délai
  - Cela signifie qu'on est en train de remplir une file d'attente
  - On était au delà du point de fonctionnement optimal
  - On baisse la taille de fenêtre sur un RTT (pour revenir à l'optimal)
  - On revient aux paramètres précédents

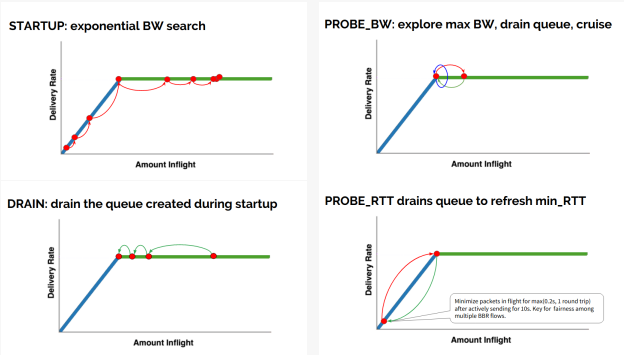


## TCP BBR : fonctionnement général

- Démarrage
  - Croissance exponentielle de la taille de fenêtre
  - Un peu à la *slow start* de Reno
  - On en sort lorsque le débit est en plateau
- Drainage
  - On ralentit les émissions de sorte à évacuer le trop plein du réseau
  - La phase de démarrage à probablement induit de la bufferisation
  - On utilise ensuite les paramètres mesurés
- Mesure du débit
  - On teste périodiquement le débit
  - Tentatives d'accélération (voir ci dessus)
- Mesure du temps d'aller-retour
  - De temps en temps, on baisse le débit pour ré-évaluer le RTT



## TCP BBR : fonctionnement général




*Figures tirées de [5]*



## TCP BBR v2 : correction des erreurs de jeunesse

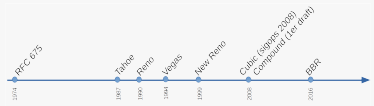
- BBR n'est pas très *friendly* avec Reno ou Cubic
  - Mesure du débit moins aggressive dans v2
- BBR ne s'intéresse pas aux pertes
  - Chute de performance dans certaines situations
  - Prise en compte dans la v2 pour une plus grande réactivité
- BBR ne tient pas compte de l'ECN
  - Prise en compte dans la v2 pour une plus grande réactivité
- Faibles performances dans certaines situations
  - En cas d'agrégation de paquets et/ou ACKs
  - Le retard dans les ACKs empêche d'émettre
  - On estime le degré d'agrégation de sorte à émettre un peu plus
- Fortes variations de débit lors des mesures de RTT
  - Mesurée dans v2 de façon moins invasive



## Quel déploiement ?

## Quel déploiement ?

Des analyses récurentes [25, 23, 22, 34, 35], mais il est difficile de se faire une idée précise.




D'après une étude menée sur les 20 000 sites référencés par Alexa (`https://www.alexa.com/topsites`)

- CUBIC est présent sur 36 % des sites
- BBR est désormais second avec 22 % (pour 40 % du trafic et ce sans Netflix ni Akamai)
- Une variante développée par Akamai sur 6 % des sites

D'après [23].



## Et UDP ?





## Et UDP ?

- Mécanismes implantés dans TCP uniquement
  - Protocole historiquement le plus utilisé pour de gros volumes
  - UDP cantonné au plan de gestion (DNS, ...) ou aux réseaux locaux (NFS, ...)
- Utilisation croissante d'UDP pour des données
  - Vidéo, voix, ...
  - Besoin de mécanismes équivalents ("*TCP Friendly*")
- En l'absence de tels mécanismes
  - Risque d'iniquité [13] [9]
  - Baisse de performance de TCP [7]
  - Besoin de civisme (*cf* l'expérience de TCP Vegas)

**Problème avec UDP**

Les mécanismes implantés dans TCP sont fondés sur la notion de connexion, absente d'UDP !



## Le protocole DCCP





## La mise en œuvre dans UDP

- Nécessité d'une mise en place au niveau applicatif
  - Difficulté de standardisation
  - Sous forme de librairie
- Définition du protocole DCCP par l'IETF [17] [8]
  - Service équivalent à UDP
  - Mécanismes protocolaires [21]
    - Établissement de connexion, accusés de réception
    - Prise en compte de l'ECN, du PMTUD, ...
  - Algorithmes de contrôle de congestion
    - TCP-*like Congestion Control* [10]
    - TCP *Friendly Rate Control* [11] (variations de débit moins abruptes, applications à taille de segment fixe)



## La prise en compte dans IP

## La gestion active des files d'attente





## La gestion active des files d'attente

- Gestion de base des files d'attente des routeurs IP
  - *Drop Tail*
  - Inéquitable
  - Synchronisation entre flots
  - Réaction lorsque les files d'attente sont pleines (les bursts entrainent alors des pertes)
- *Active Queue Management* (AQM) [1]
  - Agir de façon anticipée sur les files d'attentes
  - Par exemple en détruisant des paquets avant que la file soit pleine
  - Accroître l'équité
  - Maintenir les files peu remplies



## Random Early Detection





## *Random Early Detection*

- Un algorithme d'AQM pour le *Best Effort* [14] [1]
  - Désynchroniser les TCP
  - Accroître l'équité
  - Efficacité non prouvée
- Estimation de la taille de la file d'attente
  - Par exemple moyenne mobile
- Décision de détruire/marquer un paquet
  - Choix aléatoire
  - Probabilité fonction linéaire du taux de remplissage entre $t_{min}$ et $t_{max}$



## La notification explicite de congestion





## La notification explicite de congestion

- *Explicit Congestion Notification* ou ECN [28] [29]
  - Fournir une signalisation explicite de congestion [19]
  - Permettre à TCP (et autres) de ne pas voir le réseau comme une boite noire [12]
- Principe
  - Notification vers le destinataire des paquets (comme l'EFCI d''atm ou le FECN de Frame Relay)
  - À charge du destinataire d'en informer l'émetteur (par du contrôle piggybacké par exemple)
  - Mise en place d'une contre mesure par l'émetteur
- Mise en œuvre
  - Utilisation des deux bits de poids faible de l'ex champ TOS d'IPv4
  - Ou du *Traffic Class* d'IPv6
  - Utilisation de deux bits *Reserved* de TCP
  - Même comportement de TCP qu'en cas de perte
- Le message ICMP *Source Quench* est une forme de BECN

## Et pourquoi pas un circuit virtuel ?

5. La prise en compte dans IP
   - La gestion active des files d'attente
   - La notification explicite de congestion
   - Et pourquoi pas un circuit virtuel ?



## Et pourquoi pas un circuit virtuel ?

- Lors de la mise en place d'un circuit virtuel
  - Négociation d'un contrat de trafic
  - Réservation de ressources
  - Ingénierie de trafic
  - Congestion résiduelle limitée
- Incompatible avec IP
  - Routage en mode paquet
  - Pas d'état dans les routeurs



## Et pourquoi pas un circuit virtuel ?

"La" solution : *Multiprotocol Label Switching* (MPLS) [30]

- Paquets marqués d'un *label* à l'entrée d'un réseau
- Commutation en fonction du label
- Mise en place de chemins (LSP) éventuellement routés par IP
- Granularité variable grâce à l'empilement de labels

MPLS apporte à IP certaines propriétés du mode circuit virtuel, notamment la possibilité de faire de l'ingénierie de trafic. C'est un complément (à plus large échelle) des outils de contrôle de congestion.




[1] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang.
RFC 2309 : Recommendations on queue management and congestion avoidance in the internet.
Technical report, IETF, April 1998.
Category : Informational.

[2] R. Braden.
Extending TCP for Transactions – Concepts.
Technical Report 1379, November 1992.
Obsoleted by RFC 6247, updated by RFC 1644.

[3] R. Braden.
T/TCP – TCP Extensions for Transactions Functional Specification.
Technical Report 1644, July 1994.
Obsoleted by RFC 6247.

[4] Lawrence S. Brakmo and Larry L. Peterson.





TCP vegas : End to end congestion avoidance on a global internet.
*IEEE Journal on Selected Areas in Communications*, 13(8) :1465–1480, 1995.

[5] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson.
Bbr congestion control.
In *Presentation in ICCRG at IETF 97th meeting*, 2016.

[6] Stephen William Edge.
An adaptive timeout algorithm for retransmission across a packet switching network.
*SIGCOMM Comput. Commun. Rev.*, 14(2) :248–255, 1984.

[7] S. Floyd.
Congestion Control Principles.
Technical Report 2914, September 2000.

[8] S. Floyd, M. Handley, and E. Kohler.





Problem Statement for the Datagram Congestion Control Protocol (DCCP).
Technical Report 4336, March 2006.

[9] S. Floyd and J. Kempf.
IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet.
Technical Report 3714, March 2004.

[10] S. Floyd and E. Kohler.
Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2 : TCP-like Congestion Control.
Technical Report 4341, March 2006.

[11] S. Floyd, E. Kohler, and J. Padhye.
Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3 : TCP-Friendly Rate Control (TFRC).
Technical Report 4342, March 2006.
Updated by RFC 5348.

[12] Sally Floyd.

Tcp and explicit congestion notification.
*SIGCOMM Comput. Commun. Rev.*, 24(5) :8–23, 1994.

[13] Sally Floyd, Mark Handley, and Eddie Kohler.
Problem statement for dccp.
Internet draft, IETF, August 2005.

[14] Sally Floyd and Van Jacobson.
Random early detection gateways for congestion avoidance.
*IEEE/ACM Trans. Netw.*, 1(4) :397–413, 1993.

[15] GOOGLE, http ://www.chromium.org/spdy.
SPDY *web page*.

[16] Sangtae Ha, Injong Rhee, and Lisong Xu.
Cubic : A new tcp-friendly high-speed tcp variant.
*SIGOPS Oper. Syst. Rev.*, 42(5) :64–74, July 2008.

[17] IETF, http ://www.ietf.org/html.charters/dccp-charter.html.
*Datagram Congestion Control Protocol (dccp) Working Group Charter*.



[18] Van Jacobson and Michael J. Karels.
Congestion avoidance and control.
*Computer Communication Review*, 18(4) :314–329, 1988.

[19] R. Jain, K. K. Ramakrishnan, K. K. Ramakrishnan, and Raj Jain.
A binary feedback scheme for congestion avoidance in computer networks.
*ACM Transactions on Computer Systems*, 8 :158–181, 1990.

[20] Raj Jain, K. K. Ramakrishnan, and Dah ming Chiu.
Congestion avoidance in computer networks with a connectionless network layer.
Technical report, Innovations in Internetworking, Artech House, 1987.

[21] E. Kohler, M. Handley, and S. Floyd.
Datagram Congestion Control Protocol (DCCP).
Technical Report 4340, March 2006.

[22] Alberto Medina, Mark Allman, and Sally Floyd.
Measuring the evolution of transport protocols in the internet.



*SIGCOMM Comput. Commun. Rev.*, 35(2) :37–52, April 2005.

[23] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong.
The great internet tcp congestion control census.
*Proc. ACM Meas. Anal. Comput. Syst.*, 3(3), December 2019.

[24] John Nagle.
RFC 896 : Congestion control in ip/tcp internetworks.
Technical report, IETF, January 1984.

[25] Jitendra Pahdye and Sally Floyd.
On inferring tcp behavior.
In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, page 287–298, New York, NY, USA, 2001. Association for Computing Machinery.

[26] J. Postel.
RFC 793 : Transmission control protocol.
Technical report, IETF, 1981.



[27] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan.
Tcp fast open.
In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 21 :1–21 :12, New York, NY, USA, 2011. ACM.

[28] K. Ramakrishnan and S. Floyd.
A Proposal to add Explicit Congestion Notification (ECN) to IP.
Technical Report 2481, January 1999.
Obsoleted by RFC 3168.

[29] K. Ramakrishnan, S. Floyd, and D. Black.
The addition of explicit congestion notification (ECN) to IP.
RFC 3168, Internet Engineering Task Force, September 2001.

[30] E. Rosen, A. Viswanathan, and R. Callon.
RFC 3031 : Multiprotocol label switching architecture.
Standards track, IETF, January 2001.

[31] Jim Roskind.



Quic quick udp internet connections multiplexed stream transport over udp.
Technical report, GOOGLE, 2013.

[32] R. Stewart.
Stream Control Transmission Protocol.
Technical Report 4960, September 2007.
Updated by RFCs 6096, 6335, 7053.

[33] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan.
A compound tcp approach for high-speed and long distance networks.
Technical Report MSR-TR-2005-86, Microsoft Research, July 2005.

[34] Peng Yang, Wen Luo, Lisong Xu, Jitender Deogun, and Ying Lu.
Tcp congestion avoidance algorithm identification (caai).
2011.

[35] Peng Yang and Lisong Xu.



A survey of deployment information of delay-based tcp congestion avoidance algorithm for transmitting multimedia data.
In *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, pages 18–23, Dec 2011.

[36] L Zhang.
Why tcp timers don't work well.
In *SIGCOMM '86 : Proceedings of the ACM SIGCOMM conference on Communications architectures & protocols*, pages 397–405, New York, NY, USA, 1986. ACM.