

Compte rendu complet : Contrôle de congestion et gestion de buffer (AQM)

Introduction

Le **contrôle de congestion** est un ensemble de mécanismes destinés à réguler le trafic dans un réseau afin d'éviter la surcharge et de garantir un partage équitable des ressources. Il agit en ajustant le débit d'envoi des sources ou en gérant les files d'attente dans les routeurs.

Deux grandes familles d'approches existent :

- le **contrôle de congestion bout en bout**, réalisé par les hôtes (ex. TCP) ;
- le **contrôle de congestion assuré par le réseau**, mis en œuvre par les routeurs via des mécanismes AQM.
-

1 Contrôle de congestion bout en bout

Principe

Dans ce modèle, les extrémités de communication détectent la congestion sans aide explicite du réseau. Elles s'appuient sur :

- la **perte de paquets** (absence d'ACK ou retransmissions) ;
- l'**augmentation du délai moyen (RTT)** ;
- ou le **marquage explicite de congestion (ECN)**.

Mécanismes classiques (TCP)

- **Slow Start** : démarrage exponentiel de la fenêtre `cwnd` jusqu'à un seuil `ssthresh`.
- **Congestion Avoidance** : croissance linéaire (AIMD).
- **Fast Retransmit et Fast Recovery** : retransmission rapide et réduction douce de `cwnd` après perte.

AIMD : Additive Increase / Multiplicative Decrease

- **Augmentation additive** : $cwnd = cwnd + 1$ par RTT si tout va bien.
- **Diminution multiplicative** : $cwnd = cwnd / 2$ en cas de perte.

Ce comportement garantit une stabilité autour du point optimal et favorise une **équité** entre connexions concurrentes.

Exemples d'algorithmes TCP

- **TCP Reno** : implémente AIMD classique.
- **TCP Vegas** : ajustement de `cwnd` selon la différence entre débit attendu et observé (approche *delay-based*).
- **TCP CUBIC** : croissance cubique, adaptée aux réseaux haut débit.
- **TCP BBR** : estimation directe de la bande passante et du délai (approche *model-based*).

Détail de TCP CUBIC

Principe général : TCP CUBIC a été développé pour résoudre les limites de **TCP Reno**, dont la croissance linéaire de la fenêtre est trop lente sur les réseaux à grande capacité (haut débit et forte latence, dits *long fat networks*). CUBIC adopte une fonction **cubique du temps** pour modéliser l'évolution de la fenêtre de congestion (`cwnd`) après un événement de perte.

Équation principale : La taille de la fenêtre de congestion est donnée par :

$$W(t) = C \times (t - K)^3 + W_{\max}$$

où :

- $W(t)$: taille de la fenêtre de congestion à l'instant t ,
- C : constante de mise à l'échelle (généralement $C = 0.4$),
- t : temps écoulé depuis la dernière perte de paquet,
- K : temps nécessaire pour atteindre la valeur précédente W_{\max} ,
- W_{\max} : taille maximale de la fenêtre avant la dernière perte.

Comportement dynamique :

- Pour $t < K$: la croissance est rapide, car la fenêtre est inférieure au seuil précédent.
- Pour $t = K$: la fenêtre atteint la valeur maximale précédente W_{\max} .
- Pour $t > K$: la croissance devient plus douce, limitant les risques de nouvelles pertes.

Ainsi, la fonction cubique offre une phase d'**accélération** lorsque le débit est faible et une phase de **stabilisation** lorsque la congestion est proche.

Avantages :

- **Utilisation efficace de la bande passante** dans les réseaux à long délai (haut débit).
- **Meilleure équité** entre connexions concurrentes que TCP Reno.
- **Insensibilité au RTT** : CUBIC fait dépendre la croissance de la fenêtre du temps écoulé et non du RTT, ce qui le rend plus juste entre connexions longues et courtes.

TFRC (TCP Friendly Rate Control)

Principe général : Le protocole **TFRC** (*TCP Friendly Rate Control*) a été conçu pour les applications multimédias (voix, vidéo en continu) nécessitant un débit stable, sans à-coups, tout en restant **équitable vis-à-vis des flux TCP**. Contrairement à TCP classique, TFRC ne repose pas sur une fenêtre de congestion (`cwnd`), mais sur un **contrôle**

du débit d'envoi en bits par seconde, calculé à l'aide d'une équation modélisant le comportement de TCP Reno.

Objectif principal : Fournir un **contrôle de congestion doux et continu**, adapté aux applications temps réel, tout en conservant la « **TCP-friendliness** », c'est-à-dire la même moyenne de débit qu'un flux TCP soumis aux mêmes conditions (RTT et pertes).

Formule de débit TFRC : Le débit moyen admissible T (en octets par seconde) est donné par l'équation :

$$T = \frac{s}{RTT \times \sqrt{\frac{2bp}{3}} + t_{RTO} \times \left(3\sqrt{\frac{3bp}{8}} \times p \times (1 + 32p^2) \right)}$$

où :

- s : taille moyenne du segment (en octets) ;
- RTT : temps aller-retour moyen (Round Trip Time) ;
- p : taux moyen de perte de paquets ;
- b : nombre moyen d'ACK par segment (souvent $b = 1$) ;
- t_{RTO} : valeur moyenne du timeout (souvent $t_{RTO} = 4 \times RTT$).

Cette équation est dérivée du modèle de comportement de **TCP Reno**, mais lissée pour fournir un débit plus stable.

Comportement et fonctionnement :

- TFRC mesure périodiquement le taux de perte de paquets (p) et le RTT.
- À chaque intervalle, le débit d'envoi T est recalculé selon l'équation précédente.
- Les variations sont graduelles : TFRC **augmente ou réduit lentement le débit** pour éviter les oscillations brusques typiques de TCP Reno.

Avantages :

- **Débit plus lisse et régulier**, idéal pour les flux audio/vidéo interactifs.
- **Compatibilité TCP-friendly** : n'écrase pas les flux TCP concurrents.
- **Approche analytique** : le débit est calculé à partir d'une formule stable, non de réactions instantanées à des pertes.

Limites :

- Réactivité plus faible : l'ajustement du débit est lent en cas de variation rapide du réseau.
- Dépendance aux mesures précises du taux de perte et du RTT.
- Moins efficace dans les environnements avec pertes non liées à la congestion (ex. réseaux sans fil).

TCP BBR (Bottleneck Bandwidth and Round-trip Propagation Time)

Principe général : **TCP BBR**, développé par Google en 2016, introduit une approche radicalement différente du contrôle de congestion classique. Alors que les algorithmes tels que Reno, CUBIC ou TFRC réagissent à la **perte de paquets** ou à l'**augmentation du délai**, BBR cherche à **modéliser directement la capacité du lien** et le **délai de propagation** pour envoyer au rythme optimal.

Il s'agit d'une approche **model-based** : BBR construit un modèle du réseau (goulot d'étranglement + RTT minimal) et adapte en continu son débit pour atteindre une utilisation maximale sans provoquer de file d'attente.

Principes fondamentaux : BBR repose sur deux grandeurs mesurées en permanence :

- **BtlBw** : la *bottleneck bandwidth*, c'est-à-dire la bande passante maximale observée sur le trajet.
- **RTprop** : le *round-trip propagation time*, soit le délai de propagation minimal (sans file d'attente).

Le débit idéal d'envoi est alors :

$$\text{Taux}_{\text{BBR}} = \text{BtlBw} \times \text{RTprop}$$

BBR ajuste sa *cwnd* (congestion window) et son *pacing rate* pour que la quantité de données en vol soit proche de ce produit.

Les quatre phases de BBR : BBR opère par cycles appelés *gain cycles*, chacun comportant plusieurs états :

1. **Startup** : phase d'apprentissage rapide pour estimer la bande passante maximale. BBR double le débit jusqu'à ce qu'une saturation soit détectée (comme le *slow start* de TCP, mais plus précis).
2. **Drain** : une fois la bande passante estimée, BBR réduit temporairement le débit pour vider les files d'attente formées pendant *Startup*.
3. **Probe Bandwidth** : BBR fait varier périodiquement son débit (par petits cycles d'augmentation et de diminution) pour détecter d'éventuels changements de bande passante disponible.
4. **Probe RTT** : toutes les 10 secondes environ, BBR abaisse son débit pour mesurer le RTT minimal (RTprop) et mettre à jour son modèle.

Modèle de contrôle : BBR ne réagit pas aux pertes ; il réagit uniquement aux changements observés de **bande passante** ou de **RTT**. Les pertes ne servent qu'à signaler un problème grave (ex. file saturée ou routeur congestionné), pas à piloter le débit.

Caractéristiques et comportement :

- **Basé sur la mesure directe** : estime en continu la capacité réelle du lien.
- **Contrôle par rythme (*pacing*)** : les paquets sont émis à un intervalle régulier selon le débit cible.
- **Indépendant du RTT** : contrairement à CUBIC ou Reno, BBR n'avantage pas les connexions à RTT court.
- **Peu de pertes** : comme il évite la surcharge des files d'attente, les pertes sont rares.

Avantages :

- Utilisation quasi complète de la bande passante disponible.
- Réduction drastique de la latence et du phénomène de *bufferbloat*.
- Débit stable, même sur les liens à longue distance ou à grande capacité.
- Meilleure performance sur les réseaux modernes (fibre, 5G, datacenters).

Limites et critiques :

- Peut être **trop agressif** face à des flux TCP classiques : BBR envoie davantage de données avant la détection de congestion.
- Sensible aux erreurs de mesure du RTT minimal ou de la bande passante.
- Première version (BBR v1) parfois injuste ; BBR v2 améliore l'équité inter-flux.
-

2 Gestion de buffer et politiques de rejet (AQM)

Introduction

Les routeurs disposent de **buffers (files d'attente)** pour stocker temporairement les paquets avant leur transmission. Cependant, lorsque le trafic augmente, ces buffers peuvent se remplir, entraînant une congestion et des pertes de paquets.

La **gestion de buffer** vise à répondre à deux questions essentielles :

- **Quand** faut-il rejeter un paquet ?
- **Quel paquet** faut-il rejeter ?

Ces politiques de rejet interagissent directement avec le **contrôle de congestion TCP** et la **gestion de trafic globale**. Elles influencent fortement la stabilité, la latence et l'équité du réseau.

—

1. Politique *Tail Drop* (TD)

Principe : La politique de rejet la plus simple, appelée **Tail Drop**, consiste à accepter tous les paquets tant que le buffer n'est pas plein, puis à rejeter tout paquet supplémentaire.

Caractéristiques :

- Facile à implémenter et intuitive.
- Tous les paquets sont traités de la même manière.
- Mais entraîne une **injustice** envers le trafic en rafales : lorsqu'une rafale arrive dans un buffer déjà presque plein, tous les paquets de la rafale sont perdus simultanément.

Inconvénients :

- **Synchronisation globale** : plusieurs connexions TCP peuvent détecter la perte au même moment, réduisant toutes leur débit, puis augmentant à nouveau simultanément — ce qui crée des oscillations de congestion.
- **Perte par rafale** : injuste envers les flux intermittents ou interactifs.
-

2. RED : Random Early Detection

Principe : L'algorithme **RED** (Random Early Detection) a été introduit pour remédier aux défauts de *Tail Drop*. Il consiste à **rejeter ou marquer aléatoirement des paquets avant que le buffer ne soit plein**. L'idée est de signaler précocement la congestion aux sources TCP afin qu'elles réduisent leur débit de manière graduelle.

Mécanisme de fonctionnement : RED calcule la **taille moyenne de la file** (avg) à l'aide d'une moyenne mobile exponentielle :

$$avg = (1 - w_q) \times avg + w_q \times q$$

où :

- q : taille instantanée de la file ;
- w_q : poids de la moyenne (typiquement entre 0.001 et 0.004).

Décision de rejet :

- Si $avg < min_{th}$: aucun paquet n'est rejeté.
- Si $avg > max_{th}$: tous les paquets sont rejetés.
- Si $min_{th} \leq avg \leq max_{th}$: le paquet est rejeté avec une probabilité :

$$p = p_{max} \times \frac{avg - min_{th}}{max_{th} - min_{th}}$$

Avantages de RED :

- Préviend la congestion avant le débordement.
- Évite la synchronisation globale entre flux TCP.
- Améliore le multiplexage des connexions.

Inconvénients :

- Sensible au réglage des paramètres (min_{th} , max_{th} , p_{max} , w_q).
- Difficulté à maintenir une stabilité optimale dans tous les contextes.
-

3. RED + ECN : Notification Explicite de Congestion

Principe : Au lieu de **rejeter** les paquets pour signaler la congestion, l'idée d'**ECN (Explicit Congestion Notification)** est de **marquer** les paquets. Ce marquage évite les pertes inutiles tout en informant la source qu'il faut réduire son débit.

Fonctionnement :

- Lorsqu'un routeur RED détecte une congestion, il **marque** le paquet au lieu de le rejeter.
- Ce marquage utilise deux bits dans l'en-tête IP (CE = Congestion Experienced).
- Le récepteur détecte ce marquage et renvoie une notification (**ECN-Echo**) à l'émetteur via le drapeau **ECE** dans l'en-tête TCP.
- L'émetteur, à son tour, réduit sa fenêtre de congestion et signale la prise en compte avec le drapeau **CWR** (Congestion Window Reduced).

Avantages :

- Réduction du taux de perte de paquets.
- Maintien d'une faible latence.
- Meilleure performance sur les liens où les pertes ne signifient pas nécessairement congestion (ex. sans fil).

Inconvénients :

- Nécessite une compatibilité ECN entre routeurs et hôtes.
- Implémentation plus complexe que RED pur.

Référence : Le fonctionnement ECN est normalisé dans la **RFC 3168**.

—

4. Garanties et performances de RED

Objectifs : RED vise à atteindre une **stabilité du buffer** : la taille moyenne doit osciller modérément autour d'une valeur cible, sans grandes variations. Une mauvaise configuration (valeurs de seuils inadéquates) conduit à des oscillations excessives, voire à une instabilité complète du système.

Évaluation :

- **Comportement stationnaire** : débit moyen, utilisation du buffer, taux de perte.
- **Comportement dynamique** : stabilité instantanée, variation de la taille du buffer dans le temps.

Des tests empiriques montrent que RED, bien paramétré, permet une meilleure fluidité et une réduction des oscillations de congestion, mais un mauvais calibrage (p_{max} trop élevé, seuils mal choisis) peut rendre le système instable.

—

5. Variantes de RED

GRED (Gentle RED) :

- Variante de RED où la probabilité de rejet continue d'augmenter linéairement même au-delà de max_{th} , jusqu'à atteindre 1.
- Rend la transition entre les états « toléré » et « congestion sévère » plus progressive.
- Améliore la **stabilité** et évite les transitions brutales.

WRED (Weighted RED) :

- Version **pondérée** de RED, utilisée dans les réseaux différenciés (DiffServ).
- La probabilité de rejet dépend de la classe de service du paquet (priorité, type de flux, etc.).
- Permet une **gestion différenciée de la qualité de service (QoS)**.

Autres variantes :

- **ARED (Adaptive RED)** : ajuste dynamiquement les seuils selon la charge du réseau.
 - **BLUE, CoDel, PIE** : alternatives à RED cherchant à mieux contrôler le délai que la taille de la file.
-

6. Bilan

La gestion de buffer joue un rôle crucial dans le contrôle de congestion :

- **Tail Drop** : simple mais brutal et injuste.

- **RED** : signalisation préventive de congestion via pertes aléatoires.
- **RED** + **ECN** : signalisation sans perte, plus efficace.
- **ARED** / **WRED** / **GRED** : améliorations de stabilité et de QoS.

Une bonne gestion de file d'attente permet de maintenir un compromis entre :

Efficacité – Stabilité – Équité – Faible latence.