N d'ordre :

# THÈSE

présentée à

**L'UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS**
**ÉCOLE DOCTORALE SPI**

pour obtenir le titre de

**DOCTEUR EN SCIENCE**

Spécialité

**INFORMATIQUE**

par

**M. Omar AIT-HELLAL**

Sujet de la thèse :

# CONTRÔLE DE FLUX DANS LES RÉSEAUX À HAUT DÉBIT

soutenue le 20 novembre 1998, devant le jury composé de

| | | | |
|---|---|---|---|
| M. | **Zhen** | **LIU** | Président |
| M. | **Serge** | **FDIDA** | Rapporteur |
| M. | **James** | **ROBERTS** | Rapporteur |
| M. | **Samir** | **TOHME** | Rapporteur |
| M. | **Eitan** | **ALTMAN** | Co-Directeur, Examinateur |
| M. | **Patrick** | **BROWN** | Examinateur |
| M. | **Philippe** | **NAIN** | Directeur, Examinateur |
| M. | **Keith** | **ROSS** | Examinateur |

*à Samia*
*à mes parents*
*à mes frères et sœurs.*

## Remerciements

Les mots me manqueraient certainement pour exprimer toute ma gratitude envers Eitan Altman. Je le remercie infiniment pour m'avoir suivi tout au long de cette thèse, pour m'avoir soutenu, pour les précieux conseils qu'il m'a prodigués, et pour sa générosité. Je lui suis hautement reconnaissant pour m'avoir conforté dans mes choix. Sa compétence et sa constante bonne humeur m'ont aidé à surmonter bien des difficultés. Je le remercie également pour la souplesse qu'il m'a accordée, ainsi que pour m'avoir tout le temps encouragé à prendre des initiatives quant aux sujets de recherche, tout en restant disponible à tout moment. Je lui doit plus que je ne saurais dire.

Je tiens à remercier François Baccelli, responsable du projet MISTRAL, pour m'avoir permis d'effectuer ma thèse au sein de l'INRIA (Institut National de Recherche en Informatique et Automatique). Il a tout le temps veillé à ce qu'il règne au sein du projet un climat propice au travail.

Je remercie Serge Fdida, James Roberts et Samir Tohme pour m'avoir fait l'honneur d'écrire un rapport sur ma thèse et d'évaluer mes travaux, ainsi que Zhen Liu, pour avoir accepté de présider le Jury. Merci également à Patrick Brown et Keith Ross pour m'avoir fait l'honneur de faire partie du Jury.

Je tiens particulièrement à remercier Philippe Nain, pour avoir accepter d'être mon directeur de thèse, ainsi que Alain Jean-Marie, pour sa disponibilité à répondre à toutes les questions même les plus triviales.

Je remercie également le Professeur Aurel Lazar, pour m'avoir accueilli dans son laboratoire (Center for Telecommunication Research, de l'Université de Columbia à New-York) pendant l'été 1997.

Merci à tous les membres du projet MISTRAL, chercheurs, thésards, stagiaires et visiteurs pour leur disponibilité, leur sympathie, et pour l'ambiance amicale qui a toujours régné au sein du projet. Je remercie particulièrement Ephie Deriche, assistante du projet MISTRAL pour sa disponibilité et sa bonne humeur.

Merci à mes parents, à Samia, merci à Slimane pour leur soutien moral pendant les moments très délicats.

Enfin, que tous ceux qui ont contribué de près ou de loin à cette thèse trouvent ici toute l'expression de ma gratitude et de ma sympathie.

Merci à Tous.

## Résumé

Cette thèse évalue les performances des mécanismes de contrôle de flux dans les réseaux à haut débit, et propose des améliorations. Deux approches ont été adoptées pour ce faire : les modèles fluides permettant d'avoir des résultats analytiques, et les simulations.

La thèse comporte deux parties : une première partie sur le protocole TCP, et une deuxième sur le service ABR dans les réseaux ATM. Dans la première partie, nous calculons les critères de performance pour TCP-Vegas et TCP-Reno, tels que le débit moyen et l'équité. Nous montrons en particulier que TCP-Vegas a le même comportement que TCP-Reno, lors de forte congestions. Nous évaluons également les améliorations que nous proposons pour Vegas.

Nous évaluons ensuite les performances des algorithmes de contrôle de congestion pour le service ABR. Nous étudions l'impact du nombre de commutateurs traversés sur les tailles maximales des files d'attentes dans ces derniers, pour les algorithmes basés sur la croissance linéaire et la décroissance exponentielle du débit, ainsi que pour l'algorithme EPRCA. En utilisant la théorie de stabilité des solutions aux équations différentielles à retard, nous calculons les domaines de stabilité pour certains algorithmes, en particulier les algorithmes ERICA+ et ERAQLES. Nous proposons une méthode simple pour évaluer la convergence et l'équité des algorithmes ABR basés sur le débit explicite.

En ce qui concerne les améliorations, nous proposons des solutions pour des problèmes que nous avons constatés dans TCP, avec des illustartions par des exemples de simulations. Nous proposons également des améliorations pour TCP-Vegas dans la façon de croître la fenêtre de congestion, aussi bien dans la phase de slow start que dans la phase de congestion. Nous proposons une méthode permettant de recouvrir de la perte d'un segment retransmis. Enfin, nous proposons des améliorations pour l'algorithme de contrôle de congestion dans l'ABR, ERICA+ permettant d'avoir un algorithme stable pour tout délai de propagation.



**Mots clés :** TCP, ABR, contrôle de flux, approximations fluides, équité, stabilité.

## Abstract

In this thesis, we evaluate the flow control mechanisms in high speed networks, and we propose some enhancements. Two approaches are adopted for performance evaluation : fluid models, which allow us to obtain analytic formulas, and simulations.

The thesis contains two parts : the first part concerns TCP protocol and the second one deals with the ABR service in ATM networks. In the first part, we compute some performance criterions, such as the average throughput and fairness. We show in particular that under heavy congestion TCP-Vegas behaves like Reno. We evaluate also the enhancements that we propose for TCP-Vegas.

Then, we evaluate performances of ABR congestion control algorithms. We study the effect of the number of switches crossed on the maximum buffer sizes in the latter, for the algorithms based on the linear increase and the exponential decrease of the rate, as well as for EPRCA algorithm. By using the stability theory of solutions to delayed differential equations, we compute the stability domains for some algorithms, in particular ERICA+ and ERAQLES. We propose a simple method for evaluating convergence and fairness of explicit rate based algorithms.

Concerning the enhancements, we propose solutions for the problems that we have observed in TCP, and we give some illustrations by using simple simulation examples. We propose also enhancements for TCP-Vegas, by changing the increase of the window in the slow start phase as well as in the congestion phase. We propose a simple method which enables to recover from the loss of a retransmission. Finally, we propose enhancements for ERICA+ (ABR congestion control algorithm), which guarantee a stable algorithm for any round trip propagation delay.



**Key words :** TCP, ABR, flow control, fluid approximations, fairness, stability.

# Table des matières

# Table des figures

# Liste des tableaux

# Chapitre 1

# Introduction

Les applications utilisant le protocole TCP ont une part de trafic non négligeable sur Internet. Le protocole TCP a ainsi toutes les chances de rester prépondérant pour de très longues années, même dans les réseaux de demain (ATM par exemple). Par conséquent, l'intérêt de l'étude de TCP sous toutes ses formes et considérations est primordiale pour améliorer les communications dans les réseaux ; et dans Internet en particulier.

Avec l'avènement de la vidéo et de la téléphonie sur Internet, utilisant typiquement UDP, bien qu'ils ne soient pas parfaitement au point (essentiellement à cause des congestions), le premier trafic qui serait victime des congestions permanentes est le trafic TCP. En effet, on verra au long de la première partie de cette thèse, réservée à l'évaluation de TCP, que le trafic généré par TCP est un trafic à faible priorité devant les autres, notamment le trafic UDP. Ceci est dû au fait que les transmissions dans TCP sont orchestrées par la réception d'acquittements.

L'amélioration du contrôle de flux dans TCP devient alors une nécessité, pour minimiser les retransmissions qui peuvent accentuer encore plus les congestions dans le réseau. Ainsi, plusieurs versions de TCP ont été proposées. Dans le chapitre 3, nous nous intéressons aux plus importantes. Nous présentons ces différentes versions ainsi que le lien chronologique qui les lie les unes aux autres. Nous illustrons par des exemples simples de simulations quelques problèmes décrits dans [41, 48], ainsi que de nouveaux problèmes que nous avons identifiés [6]. Nous commentons les différentes versions, et proposons également quelques améliorations soit pour pallier à des dysfonctionnements, ou pour améliorer le rendement de TCP.

Pour comprendre la "philosophie" du mécanisme de contrôle de flux dans les versions TCP-Reno et TCP-Vegas, nous présentons dans le chapitre 4 un modèle simple pour une approche analytique [7]. Cette approche nous permettra de mieux cerner les insuffisances de chaque version, et les conditions dans lesquelles elles permettent de meilleures performances. Nous calculons pour le modèle considéré les différents paramètres de performance, à savoir le débit moyen, l'évolution de la taille de la fenêtre et les délais d'aller retour moyens. Nous proposons également quelques améliorations pour TCP Vegas qui permettent de minimiser les probabilités de perte.

Quand on parle de TCP, on fait toujours référence à Internet, bien que celui ci restera présent pour de très longues années sur les réseaux de demain, à savoir les réseaux ATM. On sait également que TCP est conçu pour transporter du trafic communément appelé *Best Effort*. Ainsi, dans les réseaux ATM, TCP sera certainement au dessus des classes conçues spécialement pour transporter ce type de trafics. Car la congestion dans ces réseaux est régie par le réseau lui même et non par le protocole de transport, contrairement à Internet. Par conséquent, TCP se verrait obligé de cohabiter avec les services tels que UBR, UBR+ ou ABR dans les réseaux ATM. Le service ABR offre plus de facilités quant à la limitation des pertes et à la prévention des congestions. Ces limitations de pertes dépendent essentiellement des algorithmes de contrôle de congestion implémentés dans les commutateurs.

Plusieurs algorithmes ont été proposés à cet effet. La deuxième partie de cette thèse est réservée à l'évaluation d'un certain nombre d'entre eux. Des méthodes de calcul direct ou d'analyse de stabilité sont utilisées. Nous définissons également des critères de performances connus, tels que l'équité et la convergence, et nous proposons des moyens pour les vérifier.

Dans le chapitre 6, les algorithmes basés sur l'indication de congestion sont évalués [8, 11]. L'objectif de ce chapitre est double : le première est de comprendre le phénomène du goulot d'étranglement. Ainsi, nous montrons que ce n'est pas nécessairement le commutateur le plus lent qui est "responsable" de la congestion. Le deuxième objectif est de calculer des bornes sur les tailles de file d'attente, dans un modèle se rapprochant plus de la réalité, où plusieurs commutateurs en tandem sont considérés. Nous examinons ainsi, la validité de l'approche qui consiste à simplifier le réseau en un seul nœud, correspondant au nœud ayant la plus petite bande passante.

Les algorithmes basés sur le débit explicite sont très difficiles à analyser par des calculs directs. Aussi, d'autres méthodes doivent être utilisées à ces fins. Les méthodes stochastiques [9, 15], ne permettent pas l'analyse de stabilité des algorithmes basés sur le débit explicite, quand ceux-ci sont un peu complexes (utilisation de la taille la file d'attente dans ERICA+ par exemple). D'autre part, les algorithmes les plus intéressants utilisent la taille de la file d'attente comme paramètre de contrôle du débit. Ce qui fait que la taille de la file d'attente est bornée bien qu'elle peut être très large. Le critère de performance à évaluer est alors les possibles oscillations de celle-ci.

Dans le chapitre 7 nous utilisons la méthode de stabilité des solutions aux équations différentielles à retard, pour évaluer la stabilité d'un certain nombre d'algorithmes [5]. Nous considérons essentiellement les algorithmes basés sur le débit explicite et/ou la taille de la file d'attente. Nous déterminons ainsi, les zones de stabilité de l'algorithme du *Point Fixe*, *ERAQLES* et *ERICA+*. Nous montrons que pour certaines bandes passantes ou pour certains délais d'aller retour, ces algorithmes ne sont pas stables. Par conséquent, un algorithme optimal doit prendre en compte le délai d'aller retour comme paramètre de contrôle.

Nous établirons également les conditions pour qu'un algorithme soit équitable et convergent. Nous analysons, en utilisant une approche analytique, ces critères d'équité et de convergence pour quelques algorithmes. Des résultats de simulations sont présentés pour confirmer les hypothèses considérées lors de l'approche analytique. Nous terminons enfin, par une conclusion sur les résultats obtenus, et les travaux que nous envisageons d'entreprendre dans un avenir proche.

# Première partie

# TCP : Transmission Control Protocol

# Introduction

L'émergence des réseaux a connu une accrue de grande ampleur depuis les années 80. Cette croissance a engendré un trafic de données très important essentiellement sur les réseaux internet (interconnexion de réseaux), particulièrement Internet (réseau mondial longue distance). La congestion dans ce dernier est devenue alors monnaie courante.

Dans le but de pallier aux pertes dues à ces congestions et aux erreurs de transmission, un protocole (algorithme) pouvant les détecter et les corriger est alors indispensable. TCP vient pour répondre à ce genre de problèmes, en proposant un contrôle de bout en bout. C'est à dire que TCP assure un transfert fiable dans la mesure où l'information manquante est toujours retransmise. Il permet également de pouvoir communiquer entre n'importe quelles machines de n'importe quels constructeurs.

Ce protocole a reçu un grand intérêt des centres de recherche et de multiples améliorations ont été proposées pour envoyer de plus en plus vite, sans causer de dommages aux autres utilisateurs. Ces travaux sur TCP sont innombrables, et chaque amélioration porte en elle même d'autres problèmes et d'autres interrogations.

Le but de cette partie est de mieux comprendre TCP essentiellement ses mécanismes de contrôle de flux, et d'illustrer en bref les améliorations les plus significatives de ces derniers. On illustrera également dans des exemples les insuffisances de chaque version et les remèdes apportés. Nous verrons essentiellement quatre versions de contrôle de flux dans TCP, ayant suscité le plus d'intérêt de la part de la communauté réseaux et protocoles en général. Ces versions sont : TCP-Tahoe, TCP-Reno, TCP-New-Reno et TCP-Vegas. Les noms de ces versions sont dûs à des lacs (Tahoe) ou à des villes (Reno, Vegas) des Etats Unis d'Amérique.

Dans le chapitre 2, une brève description du protocole TCP est donnée, et les mécanismes d'estimation du délai d'aller retour (RTT) et de détection de pertes sont également décrits.

La description des versions d'algorithmes pour le contrôle de flux et de congestion dans TCP fera l'objet du chapitre 3. Nous essayerons de garder la chronologie et les "liens" logiques entre toutes ces versions. Nous montrons aussi les insuffisances de chacune d'elles et les problèmes relevés dans quelques publications [41, 48] ainsi que ceux que nous avons constatés [6], et ce à l'aide d'exemples simples de simulation.

Nous proposons quelques améliorations à la version New-Reno. Ces améliorations viennent pour répondre à quelques insuffisances de ce dernier que nous avons constatées à l'aide des simulations. Certaines de ces améliorations ont déjà été proposées par K. Fall et al [39], sans qu'elles soient justifiées. D'autres, concernent la détection de multiples pertes du même paquet. Nous discuterons également quelques nouveaux mécanismes introduits par TCP Vegas [57, 6].

Dans le chapitre 4, une approche analytique est présentée. Le but de cette étude, est de mieux comprendre la "philosophie " de Reno, et essentiellement celle de l'algorithme Vegas. Nous considérons un modèle simple avec un trafic exogène. Nous montrons ainsi, que le trafic généré par TCP est moins prioritaire que le trafic exogène. Nous montrons également que les améliorations en performance apportées par Vegas dépendent d'un paramètre essentiel qu'est la congestion dans le réseau. Dans le cas d'une forte congestion, en particulier le cas de plusieurs connexions TCP, l'algorithme Vegas a les mêmes défaillances que Reno en ce qui concerne l'estimation de la bande passante disponible. Nous proposons des améliorations permettant de minimiser l'impact du nombre de connexions.

# Chapitre 2

# Introduction au protocole TCP

Quand une source envoie un paquet elle est incapable de savoir si le paquet est perdu, ou bien reçu par la destination, si aucune information n'est retournée par cette dernière. Aussi, pour s'assurer de la bonne réception des paquets, la destination doit acquitter les informations qu'elle reçoit, en envoyant des acquittements à la source lui indicant les informations bien reçues et/ou manquantes. Cette procédure d'echange d'information entre la source et la destination n'est autre que le protocole de communication

L'étude de tels protocoles remonte aux années 60, où les institutions gouvernementales américaines (université, et défense) étaient chargées de développer des communications à commutation de paquets. C'est ainsi que l'ancêtre de TCP, NCP a vu le jour. Initialement ce protocole est utilisé pour les communication sur ARPANET (ancêtre de Internet) reliant une cinquantaine de sites, essentiellement gouvernementaux.

Vers les années 80, ARPANET s'ouvre au monde des entreprises et sa croissance en termes de sites et de machines est exponentielle. On compte aujourd'hui pas moins de 20 millions d'utilisateurs reliés à Internet. Malheureusement cette croissance ne s'ensuit pas sans de nouveaux problèmes. En effet, le trafic sur Internet ne cesse d'augmenter causant ainsi des congestions au niveau des routeurs, et par conséquent des pertes de données ou de paquets.

Un protocole fiable permettant dans la mesure du possible d'éviter des pertes, ou de les corriger au cas où elles se produisent est alors nécessaire.

## 2.1 Le protocole TCP

C'est un protocole de bout en bout utilisant des acquittements pour s'assurer de la bonne réception des paquets à l'autre bout. Les communications par TCP peuvent être en sens unique aussi bien que dans les deux sens ("full duplex"). Dans ce dernier cas les paquets d'une source comportent également les acquittements pour les paquets de l'autre. Pour la programmation et les appels permettant d'ouvrir une connection, une description complète de TCP abordant tous les aspects peut être trouvée dans [36, 99]

La structure d'un paquet ou d'une entête TCP est donnée dans la figure 2.1. Les champs sur 32 bits 32-63 et 64-95 sont les numéros de séquence du paquet envoyé et le numéro de séquence du paquet acquitté respectivement. Un paquet TCP comporte une entête de 20 octets, et sa taille peut aller jusqu'à 64 Kocets ; elle est de 556 octets par défaut (l'entête comprise). Tout au long de ce document on inter-changera entre l'appellation paquets et segments, pour désigner la même entité de données.

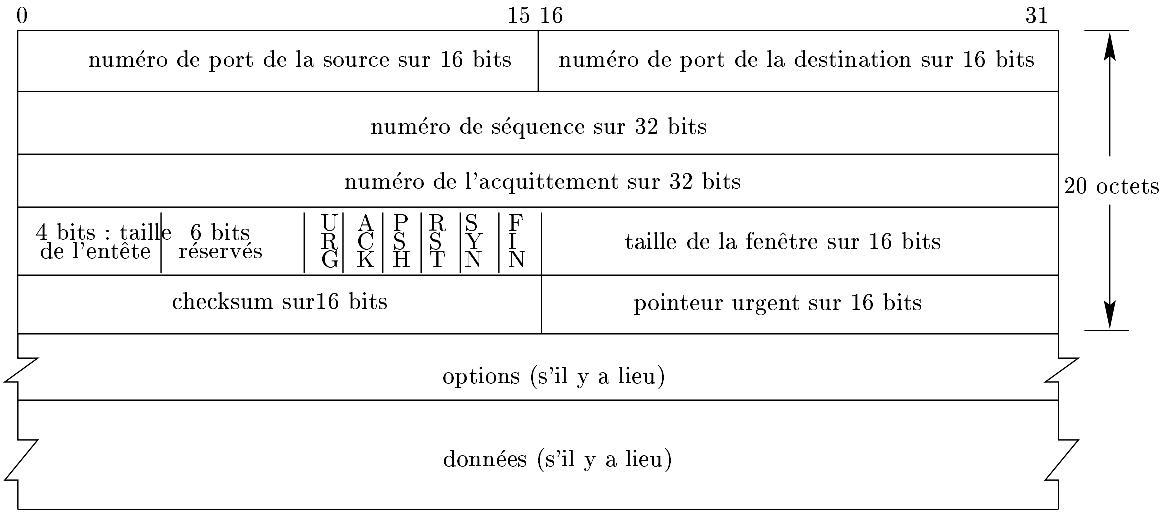



FIG. 2.1: Structure d'un paquet TCP

L'exemple type d'applications utilisant TCP est le FTP (File Transfert Protocol). Cette application permet de récupérer des données situées sur une machine distante. Les données récupérées doivent alors être une copie conforme à l'original. Par conséquent, les pertes et les erreurs doivent être corrigées.

### 2.1.1 Contrôle de flux et fenêtre de congestion

Lors de transmission de paquets, TCP partage plusieurs chemins avec d'autres trafics de données. Il est alors incapable d'estimer de manière exacte les ressources qui lui seront attribuées. Ainsi, des données peuvent se perdre à cause de la saturation des liens. Dans le souci de transmettre plus et plus vite sans avoir à saturer le réseau, le protocole TCP inclut un contrôle de flux. Ceci lui permet de réguler son débit en fonction de la congestion dans le réseau. Plusieurs algorithmes ont été proposés à cet effet. Les plus "intéressants" (connus) feront l'objet du prochain chapitre d'une manière plus détaillée.

Les pertes en général se produisent pour cause de congestion, et insuffisance de ressources dans le réseau. Ainsi, à la détection d'une perte, toutes les versions de TCP proposent de revoir leur débit de transmission. La fenêtre de congestion ($Cwnd$) étant définie comme le nombre permis, de paquets (ou octets) transmis et non encore acquittés (octets en transit). Par conséquent, en manipulant la taille de cette fenêtre, le débit de transmission peut être alors réduit ou augmenté.

Le mécanisme de contrôle (Tahoe, Reno, New-Reno ou Vegas) augmente ou réduit la taille de la fenêtre selon qu'un paquet est bien reçu ou perdu. Les différentes versions de TCP diffèrent simplement, comme on le verra dans le chapitre suivant, dans l'evolution de la fenêtre et/ou la façon de detecter les pertes ou la congestion.

### 2.1.2 Acquittements (dupliqués)

Les paquets sont envoyés de manière séquentielle. La destination s'attend à les recevoir dans cet ordre, et acquitter chaque paquet reçu par un paquet indiquant sa séquence. Quand une perte (par ex. séquence $i + 1$) se produit, tous les acquittement qui sont émis après la perte portent le même numéro de séquence ($i$ si c'était le dernier reçu en séquence), on appelera de tels acquittements, portant la même séquence, des *aquittements dupliqués*. Les paquets hors séquence reçus ($j > i + 1$) sont sauvegardés par la destination dans la mesure des tampons disponibles. Ce nombre de tampons disponibles définit la fenêtre de réception ($Rcv\_wnd$), qui est à chaque fois signalée à la source par la destination lors de l'envoi d'un acquittement. La source doit alors limiter sa fenêtre de congestion à ce nombre sous peine d'engorgement de la destination ou des pertes imminentes ($Cwnd = \min(Cwnd, Rcv\_wnd)$).

Du coté de la source, dès la réception de trois paquets dupliqués, acquittant le même paquet ($i$), ce dernier est considéré perdu et retransmis à l'occasion. Le choix du nombre trois est dû au fait que les paquets peuvent ne pas être délivrés dans l'ordre, si chacun empreinte un chemin différent. Par conséquent, la réception d'un voire même de deux acquittements dupliqués ne signifie pas forcément une perte.

#### 2.1.2.1 Fenêtre de données

L'envoi des paquets dans TCP est orchestré par la réception d'acquittements. Dans le fonctionnement "normal" de TCP (pas de pertes), la réception d'un acquittement cause la transmission de zéro (possible dans Vegas), d'un ou de deux segments, selon la version du protocole. D'une manière récursive (voir chapitre 4), une fenêtre de données est constituée de segments transmis dûs à la réception des acquittements des segments de la fenêtre précédente. Cela implique donc, que les paquets sont envoyés en "trains". Les "trains" (fenêtre de données) peuvent être espacés dans le temps tout comme ils peuvent se suivre les uns derrière les autres (transmission continue).

### 2.1.3 Temporisateur de retransmissions (Timeout) et estimation du RTT

TCP dispose d'un temporisateur d'une résolution de 500 ms en général, en le sens où une interruption est générée toutes les 500 ms et un code est exécuté. Ce temporisateur sert d'une part à arrêter une communication (avorter une communication) dans le cas de non possibilité d'établir la liaison (liens cassés, très forte congestion, etc...) ; et d'autre part à détecter les pertes. En effet, si avant le timeout calculé en nombre de ticks du temporisateur, aucun acquittement n'est reçu, le premier paquet transmis (en séquence) et non encore acquitté est considéré perdu, et par conséquent retransmis.

#### 2.1.3.1 Calcul du Timeout

TCP mesure d'abord le délai d'aller retour (RTT : Round Trip Time). Lire le temps exact sur une machine avec une résolution au Nano-seconde n'est possible que depuis très peu d'années. Cela nécessite également du temps CPU. C'est pour ces raisons que la méthode des "ticks" a été préférée. Ainsi, le délai d'aller retour est mesuré avec une granularité égale à celle du temporisateur. Les anciens temporisateurs étaient de 500 ms de résolution. Quelques versions de TCP sont aujourd'hui implémentées avec une horloge à 300 ms.

A l'envoi d'un paquet, TCP calcule le nombre de "Ticks" d'horloge depuis son départ jusqu'à la réception de son acquittement. Ce nombre est considéré comme le délai d'aller retour, mesuré a cet instant, entre la source et la destination. Puisque que ce délai est vulnérable aux changements de chemins ou de densités de trafics dans le réseau, TCP doit le réactualiser à chaque fois que cela lui est possible. Aussi, plusieurs estimations au cours d'une communication sont faites. Les séquences temporisées sont les premières séquences de chaque fenêtre de données (Timed_seq : voir figure 3.5).

Étant donné le RTT mesuré (M) à l'instant présent, TCP calcule un RTT estimé (A) de la façon suivante [99] :

$$\begin{aligned} Err &= M - A \\ A &\longleftarrow A + gErr \end{aligned} \tag{2.1}$$

Où $Err$ est la différence entre la valeur mesurée du RTT et son estimateur. $g$ est le gain pour la moyenne du RTT et pris égale à 1/8 (0.125).

Une fois que le RTT est estimé, TCP peut alors estimer à nouveau la valeur du timeout (RTO). Cette valeur est la réponse à la question suivante : combien de temps (de ticks) doit-on attendre (sans recevoir d'acquittements) avant de considérer qu'un segment est perdu.

Deux fois l'estimation du RTT, suffisent pour s'assurer qu'aucun autre acquittement ne serait reçu. Initialement, c'était cette estimation qui était adoptée avant que Van Jacobson [53] n'ait proposé le schéma suivant :

$$\begin{aligned} D &\longleftarrow D + h\left(|Err| - D\right) \\ RTO &= A + 4D \end{aligned} \tag{2.2}$$

Où $D$ est la déviation du RTT. $h$ est le gain pour la déviation (facteur de pondération), il est pris égale à 1/4.

Lors des expirations successives du timeout, la valeur de celui-ci est doublée avec une limite à 64 secondes. Cette multiplication du timeout est appelée *exponential backoff*.

#### 2.1.3.2 Algorithme de Karn

Pour l'estimation du RTT, on s'est demandé si l'on prend en compte le RTT estimé lors d'une retransmission. Cela pourrait, en effet, poser le problème d'une mauvaise estimation. Ceci se produit dans le cas où l'on retransmet un segment et juste après, le timeout expire avant le retour de son acquittement. A l'expiration du timeout, le segment sera retransmis une deuxième fois. Ainsi, à l'arrivée de son acquittement on serait incapable de déterminer si l'acquittement est dû à la première ou à la deuxième transmission. Aussi, on préfère ne pas estimer le RTT lors des retransmissions (Algorithme de Karn [99]), et d'utiliser le RTT déjà estimé, de même pour le timeout avec d'éventuels "backoff".

### 2.1.4 Temporisateur persistant (Persist Timer)

Il y a des situations où le transmetteur est beaucoup plus rapide que le récepteur. Ces situations peuvent avoir lieu quand le récepteur est "submergé" par le travail ou tout simplement, pas aussi puissant que le transmetteur. Le récepteur se verra, par conséquent, incapable de traiter les données à la même vitesse qu'il les reçoit. Il sera alors obligé de les sauvegarder, et de les traiter quand il aura un moment de répit. Pendant ce temps, pour éviter de recevoir encore des données, il avertit la source d'une fenêtre nulle lui indiquant qu'il ne peut pas recevoir de données pour l'instant.

C'est ainsi que la source rentre dans l'état "persist", où elle envoie des paquets de taille d'un octet de données à chaque fois que le temporisateur expire, pour vérifier l'état de la destination. Il se pourrait bien que la destination ait repris le traitement des données déjà envoyées. Si c'est le cas, à la réception d'un paquet "vide" par la destination, celle ci renvoit un acquittement de la dernière séquence bien reçue tout en indiquant le nombre de tampons disponibles, et la communication peut ainsi reprendre.

Le calcul du timeout est le même que celui du timeout pour les retransmissions, à la seule différence que la valeur minimale du timeout dans ce cas est de cinq fois la résolution du temporisateur. Le backoff exponentiel est toujours appliqué à chaque expiration successive du timeout.

# Chapitre 3

# Fonctionnement de TCP : problèmes et améliorations

Les mécanismes de contrôle de congestion pour les protocoles de TCP/IP ont été longtemps étudiés et de nombreux algorithmes ont été proposés dans la littérature. Beaucoup de ces algorithmes resteront sur papier sans qu'il aient été implémentés (ex. Tri-S [102]). D'autres susciteront beaucoup plus d'intérêt, comme celui proposé par Van Jacobson [52] baptisé Tahoe. Cet algorithme, comme beaucoup d'autres, est basé sur un système de fenêtrage dynamique. Révisé deux ans après par son auteur [54], en introduisant les mécanismes de *Fast Recovery* et *Fast Retransmit* [41, 99], cet algorithme a donné naissance à une nouvelle version appelée *Reno*.

A l'issue d'études sur les insuffisances et les dysfonctionnements de Reno, une autre version baptisée New-Reno a été proposée par J. Hoe [48]. Enfin, l'une des versions de TCP qui a suscité beaucoup d'attention est TCP-Vegas, qui a été proposée dans le courant de 1994, par L. Brakmo et al [26]. Cette dernière se veut plus efficace et plus performante.

Dans le présent chapitre, nous présentons avec plus de détails les versions citées ci-dessus, qui nous semblent retenir le plus d'intérêt de la part de la communauté réseaux et protocoles. Nous présentons également quelques problèmes connus des différentes versions, ainsi que des problèmes nouveaux que nous avons identifiés. Nous illustrons ces prblèmes par de simples exemples de somulation et nous proposons des solutions pour y palier, ainsi que des améliorations pour le contrôle de flux dans TCP.

Les simulations présentées tout au long de ce travail ont été obtenues à l'aide du simulateur de réseaux REAL [68]. Ce simulateur permet de simuler des réseaux à commutation de paquets. Nous avons inclus la version Vegas de TCP, par modification du code source jk_ftp.c (code source pour Tahoe dans REAL), et les modifications apportées s'inspirent du code de Vegas [27] (vtcp_input.c, vtcp_timer.c,...). Pour illustrer les différents problèmes et caractéristiques de chaque algorithme, nous avons considéré un modèle simple avec une source TCP contrôlée transmettant des paquets vers une destination et partageant un goulot d'étranglement avec une autre source (source Poissonnienne).

Nous avons permis une fenêtre maximale de 128 paquets, correspondant approximativement à 64 Koctets qui n'est jamais atteinte dans nos exemples. Tous les paquets ont une taille fixe de 576 octets (les entêtes IP et TCP comprises), ce qui donne une taille de fenêtre multiple de 576 octets.

Le chapitre est structuré comme suit : Nous présentons d'abord la version Tahoe de TCP et illustrons ses insuffisances, et ce qui a donné naissance à la version Reno. Nous présentons cette dernière dans la section 3.2 et nous la comparons à l'ancienne. Dans la section 3.3, la version Vegas de TCP est décrite, et les nouveaux mécanismes introduits sont discutés. Quelques problèmes sont présentés dans la section 3.4, et la toute recente version de TCP (TCP New-Reno [48]) venant pour palier à ces problèmes est décrite. Nous analysons cette dernière version et proposons des améliorations. Enfin, Nous proposons une méthode permettant de recouvrir des pertes des retransmissions.

## 3.1 TCP Tahoe

Dans les années 80, des protocoles basés sur le système de fenêtrage ont été proposés. Ainsi, en utilisant des fenêtres on peut raisonnablement limiter le débit d'une source. La fenêtre fixe a été la première proposée, cependant il est évident de constater qu'avec un tel système de fenêtrage, une fois que la congestion a lieu elle demeurera pour de très longues périodes. Pour éviter cela, un mécanisme de fenêtre variable lui est préférable [52, 90, 102] pour répondre aux situations de congestion dans les réseaux (longue distance).

Ainsi apparut la première version de TCP, baptisée TCP-Tahoe. Dans cette version, la fenêtre de congestion ($Cwnd$) définit le nombre de paquets (ou octets) transmis et non encore acquittés. Le contrôle de flux et de congestion est régulé en fonction des pertes constatées par une source. Ces pertes, sont détectées par un temporisateur. En effet, si le temporisateur expire ne voyant aucun acquittement arriver durant cette période, le premier paquet (en séquence) non encore acquitté est alors considéré perdu.

La source en fait exécute deux phases de transmission : le *slow start* et la phase de *congestion* :

### Phase du *Slow Start*

C'est une ouverture (croissance) "exponentielle" de la fenêtre. La source augmente la taille de sa fenêtre d'un paquet pour chaque acquittement reçu. La phase du *slow start* est initiée chaque fois qu'une perte est détectée, et elle se termine quand la taille de la fenêtre atteint un certain seuil appelé *seuil du slow start* $W_{th}$ (initialisé dans nos simulations à 32 paquets).

### Phase Congestion

Cette phase commence à la fin de la phase du *slow start.* TCP continue d'augmenter la taille de sa fenêtre de $1/Cwnd$ chaque fois qu'il reçoit un acquittement, jusqu'à ce que la fenêtre atteigne la taille de la fenêtre de réception ($Rcv\_wnd$) ou la taille maximale. Comme la valeur de la taille de la fenêtre que nous avons considérée dans nos simulations est entière, et que $1/Cwnd$ ne l'est pas, alors augmenter la fenêtre de $1/Cwnd$ pour chaque acquittement, revient à augmenter la fenêtre d'un paquet, chaque fois que tous les paquets de la précédente fenêtre sont acquittés (tous les RTTs). La phase de congestion prend fin dès la détection de perte (expiration du temporisateur). Dans ce cas, le seuil du *slow start* est mis a la moitié de la taille actuelle de la fenêtre et la fenêtre est remise à un, et un nouveau cycle commençant par un *slow start* est entamé.

### 3.1.1 Vers TCP Reno

Le problème majeur de Tahoe est le temps dépensé dans la détection de pertes. En effet, les temporisateurs sont généralement et même plus souvent d'une résolution de 500 ms, pendant que les délais d'aller retour tendent de plus en plus à se raccourcir (câbles torsadé, fibre optique,...). La détection d'une perte peut alors prendre au moins 500 ms ce qui constitue plusieurs délais d'aller retour.

Des améliorations ont été introduites pour permettre d'utiliser beaucoup plus la bande passante. Le mécanisme de *Fast Retransmit* en est un exemple. Ce mécanisme, considère qu'un paquet est perdu dès la réception de trois acquittements dupliqués par la source, ce qui permet de détecter les pertes plus rapidement. Le nombre de trois (comme expliqué précédemment) a été justifié par le fait que les paquets peuvent arriver en désordre à la destination.

En remarquant que la réception d'un acquittement dupliqué, est dûe à un départ d'un paquet du réseau (un paquet est servi, il est arrivé à la destination) ; l'envoi d'un nouveau paquet à chaque réception d'un acquittement dupliqué est alors justifié. Ce mécanisme, introduit spécialement dans Reno, est le *Fast Recovery Mechanism* [54]. Il propose donc de remplacer les paquets ayant quitté le réseau par d'autres (nouveaux paquets).

## 3.2 TCP Reno

Les principales innovations dans TCP Reno par rapport à TCP Tahoe sont le facteur de réduction de la fenêtre, dans le cas où une perte est détectée par acquittements dupliqués, et les actions à entreprendre à ce moment. En effet, dans Reno lors de la détection d'une perte par acquittements dupliqués, *seul* le paquet supposé être perdu, mais pas les suivants, est retransmis, et la phase de *congestion* au lieu du *slow start* est exécutée à nouveau :

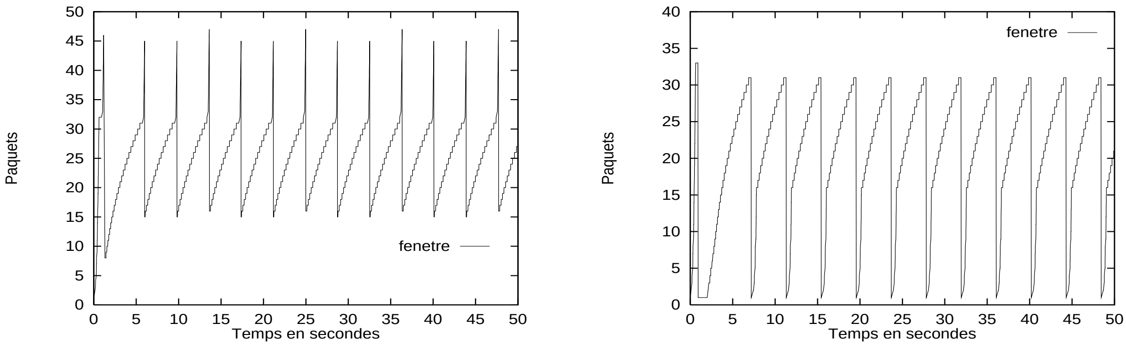
1. Le seuil du *slow start* ($W_{th}$) est mis à un demi de la taille actuelle de la fenêtre de congestion.
2. La fenêtre de congestion ($Cwnd$) est mise à $W_{th}$ plus trois fois la taille d'un paquet (MSS, i.e. Maximum Segment Size).
3. A chaque réception d'un acquittement dupliqué, la source incrémente $Cwnd$ d'un paquet et transmet un nouveau paquet (si la nouvelle taille de $Cwnd$ le permet).
4. Quand le premier acquittement non dupliqué arrive, $Cwnd$ est mise à $W_{th}$ (un nouveau cycle en phase de *congestion* est entamé).

Les points 2-4 constituent le mécanisme de *fast recovery* .

la figure 3.1, illustre les comportements de la fenêtre dans TCP-Reno et TCP-Tahoe. Notons que dans Reno, on cherche à éviter de recommencer la phase du *slow start* où les pertes sont souvent de l'ordre de la fenêtre (phase très agressive). Les lignes verticales que l'on constate après chaque perte, correspondent au mécanisme de *Fast Recovery* où la fenêtre augmente de la taille d'un paquet pour tout acquittement dupliqué reçu.

### 3.2.1 Reno vs Tahoe

Dans cet exemple, on peut facilement constater que Reno est meilleur que l'ancienne version, cependant ce n'est malheureusement pas toujours le cas. En effet, il existe des cas où les performances de Reno se dégradent de manière drastique. Ceci se produit quand plusieurs




(a) Reno (b) Tahoe

FIG. 3.1: Comportement de la fenêtre dans Reno et Tahoe

pertes successives ont lieu dans la même fenêtre de données (voir chapitre précédent ou dans [41] pour une meilleure explication d'une fenêtre de données).

Cette situation peut se résumer de la façon suivante :
Dans le cas de pertes multiples (groupées), Reno réduit sa fenêtre de moitié à chaque détection d'une perte par acquittements dupliqués (il peut recouvrir, dans certains cas, jusqu'à trois pertes ce qui est rare). En l'absence d'acquittements (dupliqués), et si d'autres pertes restent encore non détectées, Reno attend l'expiration du timeout pour initialiser la phase du *slow start* et les recouvrir. Ceci peut prendre plusieurs délais d'aller retour, puisque la granularité du timeout est de 500 ms. La phase de *slow start* ainsi initialisée est entamée avec un seuil égal à la taille de la fenêtre au moment de détection de la première perte, divisée par $2^{1+n}$, où $n$ est le nombre de pertes (successives ou d'une même fenêtre de données [41]) détectées par acquittements dupliqués.

Dans le cas de Tahoe, quand de multiples pertes se produisent, dès la détection de la première perte la fenêtre est mise à un, et le seuil du *slow start* ($W_{th}$) est mis à la moitié de la taille de la fenêtre au moment où la perte est détectée. Par conséquent, Tahoe exécute la phase du *slow start* (et par la même occasion recouvre les autres pertes) plus rapidement que Reno, et avec un seuil ($W_{th}$) beaucoup plus grand. C'est la raison pour laquelle Tahoe est meilleur que Reno dans le cas de plusieurs pertes dans la même fenêtre de données (voir [73]).

### Exemple

Supposons qu'il y ait trois pertes dans la même fenêtre de données. Supposons également que les deux premières pertes sont détectées par acquittements dupliqués et que la troisième par timeout. Reno initialise la phase du *slow start* (à la détection de la troisième perte) avec un seuil égal à la taille de la fenêtre, quand la première perte est détectée, divisée par 8 ($W_{th} = max\_win/8$) et ce pas avant la résolution du temporisateur (500 ms), au minimum. Dans la version Tahoe, la phase du *slow start* est initialisée avec un seuil quatre fois plus grand (max_win/2)), dès la détection de la première perte.

**Remarque 3.2.1** *Dans toutes les figures de ce chapitre, l'axe des ordonnées correspond au numéro de séquence des paquets (Seq-) (resp (et/ou) des acquittements (Ack-)) et l'axe des abscisses aux instants de leurs transmissions (resp (et/ou) réceptions).*

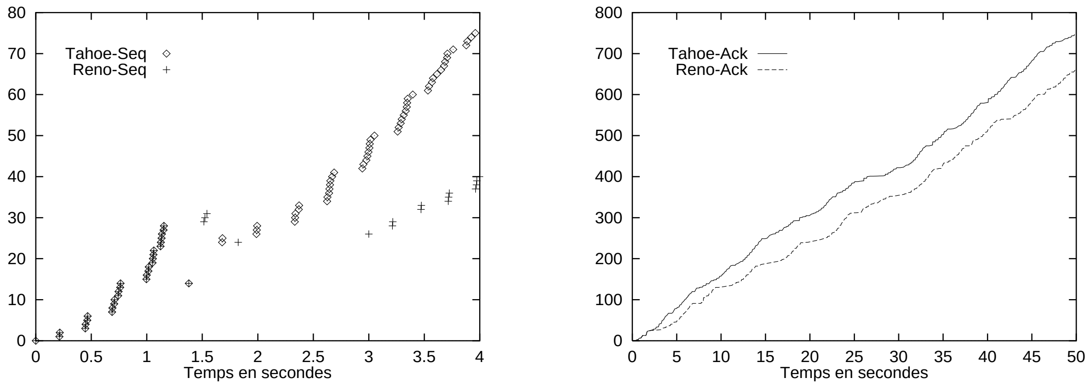



FIG. 3.2: Un exemple où Tahoe excède Reno

La figure 3.2 illustre un exemple où Tahoe excède Reno en débit tout en ayant, relativement, le même taux de retransmission pour une durée de simulation de 50 sec. On peut constater dans la figure de gauche, où uniquement les premières quatre secondes figurent, que les séquences de transmission pour Reno et Tahoe (Seq sur la figure) sont les mêmes jusqu'à l'instant 1.37 où des acquittements dupliqués acquittant le paquet 13 arrivent.

Du coté de Reno, à l'arrivée de trois acquittements, le seuil du *slow start* est mis à $W_{th} = \lfloor 15/2 \rfloor = 7$ [1], car la fenêtre est de 15 paquets. Reno retransmet le paquet 14, et les paquets 29-31 dus au mécanisme de *fast recovery* (les "plus" sur la figure). A l'instant 1.80, trois acquittements dupliqués acquittant le paquet 23 sont reçus (dûs aux paquets 14, 29-31). Le paquet 24 est alors considéré perdu, et retransmis à l'occasion (ce dernier causera la réception de l'acquittement pour le paquet 25). Une nouvelle phase de *congestion* est alors entamée : le seuil du *slow start* est une nouvelle fois divisé par deux $W_{th} = \lfloor 7/2 \rfloor = 3$. En l'absence d'acquittements dupliqués (pour le paquet 25), on attend près d'une seconde et demi pour que le temporisateur expire. La phase du *slow start* est alors initialisée ($W_{th} = max(2, \lfloor 3/2 \rfloor = 2)$, ainsi on recouvre les autres pertes lentement, car la valeur du seuil du *slow start* est la plus petite possible.

Quant à Tahoe, dès la détection de la perte du paquet 14, par réception de trois acquittements dupliqués acquittant 13, il met le seuil du *slow start* à $W_{th} = \lfloor 15/2 \rfloor = 7$ et réinitialise la phase du *slow start* en mettant la fenêtre à 1. Il retransmet donc le paquet 14, puis à la réception de chaque acquittement non dupliqué, il (re)transmet les paquets subséquents. Il recouvre ainsi toutes les autres pertes à la même occasion, sans attendre l'expiration du timeout. Par conséquent, Tahoe réinitialise la phase du *slow start* avec un seuil presque quatre fois plus grand que Reno, et une seconde et demi plutôt.

Il serait tout de même injuste de ne pas souligner que des retransmissions inutiles ne sont pas évitées par Tahoe, ce qui peut alourdir le trafic dans les réseaux et causer des pertes à d'autres sources concurrentes.

### 3.2.2 Problème de "successive fast retransmits"

Un problème lié à l'exécution de plusieurs "fast retransmit" successifs, dans la même fenêtre de données, a été retracé dans [41]. Ce problème concerne essentiellement la ver-

[1] $\lfloor . \rfloor$ désigne la partie entière inférieure. Puisque on suppose que les paquets sont de taille fixe, la fenêtre est alors un nombre entier de segments

sion Tahoe, mais il peut avoir lieu dans Reno, aussi bien que dans Vegas, sous une autre forme (moins désastreuse), comme on le verra par la suite. Ce dysfonctionnement est dû à la réinitialisation de la phase du *slow start* à chaque fois qu'une perte est détectée ; cela cause la retransmission de tous les paquets, sans qu'ils soient vraiment perdus (tous les paquets sont transmis deux fois voir figure 3.3).

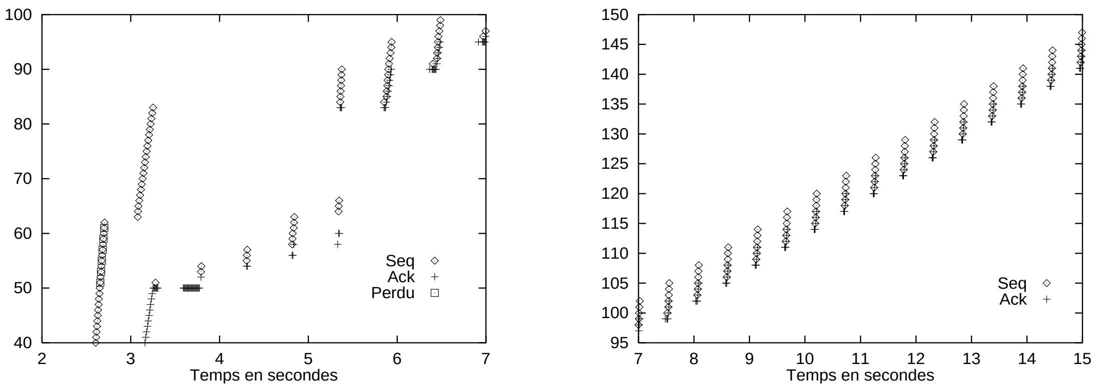



FIG. 3.3: Le problème de "fast retransmit" successifs dans Tahoe

La figure 3.3 en illustre un exemple. Le scénario est le suivant : la phase du *slow start* est exécutée initialement, et les paquets 0-83 sont transmis. Parmi ces paquets, six sont perdus (Perdu sur la figure) : 51, 53, 55, 57, 59, 61. A l'instant 3.27, la source reçoit des acquittements dupliqués acquittant le paquet 50 (figure de gauche). Le paquet 51 est alors retransmis et la phase du *slow start* est réexécutée. Les paquets 58-63 sont retransmis à l'instant 4.81, après avoir recouvert toutes les pertes précédant ces séquences.

A la réception du paquet 58 (doublement reçu), l'acquittement 58 est envoyé par la destination une nouvelle fois, puis deux fois l'acquittement 60 dûs à la réception des paquets 59 et 60 (seul 61 reste à recouvrir). Ainsi, dès la réception du paquet 61 par la destination, celle-ci envoie des acquittements dupliqués acquittant le paquet 83, tant que le paquet 84 n'est pas reçu (on recevra trois acquittements 83 dûs à la bonne réception des paquets 61-63).

C'est à ce moment où le problème apparaît. En effet, du coté de la source, à la réception des acquittements 58-60-60, les paquets 64-66 sont envoyés, la fenêtre étant de 7 (5 + deux acquittements distincts 60 et 83 ; *slow start*). Les paquets 84-90 sont envoyés dès la réception du premier acquittement du paquet 83, après quoi on reçoit encore trois acquittements dupliqués pour le paquet 83 dûs à la bonne réception des paquets 64-66, ce qui porte à six (6) le nombre de paquets dupliqués 83. Le paquet 84 est retransmis, en exécutant le "fast retransmit", puisque des acquittements dupliqués (83) sont reçus. Celui-ci est considéré à ce moment comme étant la plus grande en séquence transmise (la phase du *slow start* est recommencée).

A chaque réception d'un paquet (84-90) la destination les acquitte au fur et à mesure de leur arrivée. Ainsi, les paquets 84-90 seront retransmis à cause de leur propres acquittements. A l'arrivée de ces retransmissions à la destination, celle-ci envoie des acquittement dupliqués acquittant le paquet 90 (avant l'arrivée des autres paquets envoyés, dûs à l'ouverture exponentielle de la fenêtre : *slow start* 91-95). Finalement, le paquet 91 à son tour sera considéré perdu et retransmis, puis tous les autres à la même occasion et ainsi de suite, tous les paquets à partir du paquet 84 seront transmis deux fois sans qu'ils soient perdus, comme on peut le constater dans la figure de droite.

La solution proposée par S. Floyd [41] est de ne pas permettre un deuxième "fast retransmit" dans la même fenêtre de données. Ainsi, lors d'un premier "fast retransmit", Tahoe sauvegarde la plus grande séquence dernièrement transmise $snd\_high = snd\_nxt$, et un autre "fast retransmit" ne serait accepté que si le dernier acquittement est plus grand que $snd\_high$.

Cette solution en effet évite le problème décrit précédemment, cependant quand plusieurs pertes ont lieu dans la même fenêtre, on aura forcement recours au temporisateur pour les recouvrir. Si on inclut cette solution également dans Reno, le premier problème (cf section 3.2.1) risque alors de surgir et Reno ne serait meilleur que Tahoe que dans le cas de pertes isolées ce qui est rare sur Internet.

Dans nos simulations de Vegas et Reno, nous n'avons pas inclus cette solution, car ce problème ne les concerne visiblement pas ; du moins comme on le verra plus tard (voir section 3.4.1), ses conséquences sont moins dramatiques que dans Tahoe. D'autre part, cette solution pénalise l'objectif de recouvrir de plusieurs pertes comme il sera discuté ultérieurement. Nous verrons également par la suite que la version New-Reno qui apporte des réponses aux pertes successives, peut souffrir également de ce problème.

## 3.3 TCP Vegas

L.S Brakmo et L.L Peterson [26] ont proposé un nouvel algorithme baptisé Vegas issue de quelques modifications du coté transmetteur (source) de Reno. Les modifications concernent essentiellement la façon de prédire la bande passante disponible, et la détection de pertes. Ce nouvel algorithme, selon ses auteurs [26], est capable d'interagir avec toute implémentation valide de TCP. Il permet, en particulier, d'améliorer de 37% à 71% le débit par rapport à Reno et de retransmettre entre un cinquième et la moitié de ce que retransmet Reno. Dans [4], les auteurs confirment ces améliorations, en utilisant un émulateur de réseaux. Ils rapportent un gain en débit d'au moins 4%-20% par rapport à *Reno* pendant qu'il retransmet le même taux donné dans [26].

Les mécanismes de "fast retransmit" et "fast recovery" y sont également implémentés. Un nouveau *slow start* est introduit visant a minimiser les pertes durant cette phase, où elles peuvent être très importantes. La taille de la fenêtre de congestion est ajustée selon la phase exécutée :

#### Phase de Congestion

Dans sa phase de *congestion*, Vegas estime la quantité de données sauvegardées dans les files d'attente intermédiaires et décide d'augmenter ou de réduire la taille de la fenêtre en s'y référant. Cette estimation est faite une fois durant chaque délai d'aller retour (RTT) : Vegas initialise $BaseRTT$ au minimum de tous les délais d'aller retour (round trip time) mesurés (y compris ceux des précédents RTT) et calcule le débit $Expected$ (débit maximum) comme étant $Expected = Cwnd/BaseRTT$ où $Cwnd$ est le nombre d'octets en transit (i.e. la taille de la fenêtre). Pendant cette période, Vegas enregistre l'instant de transmission d'un paquet distingué et le nombre d'octets transmis entre cet instant et l'instant d'arrivée de son acquittement, et calcule ainsi le débit réel de transmission comme étant $Actual = Cwnd/RTT_d$, où $RTT_d$ est le délai d'aller retour du paquet distingué.

La source (transmetteur) définit trois seuils $\alpha$, $\beta$ et $\gamma$ (dans nos simulations, nous considérons $\alpha = 2$, $\beta = 4$ et $\gamma = 1$ ($Vegas_{2,4}$)). Si $Expected - Actual \leq \alpha/BaseRTT$ alors la fenêtre est augmentée de $1/Cwnd$ pour chaque acquittement reçu (comme Reno). Cependant, si $Expected - Actual > \beta/BaseRTT$, la fenêtre est décrémentée d'un paquet. Enfin, si la différence $Expected - Actual$ est comprise entre les deux seuils ($\alpha$ et $\beta$), la fenêtre est maintenue inchangée.

A noter que dans la version implémentée (Vegas release 0.8 [27]), $RTT_d$ est considéré comme la moyenne des délais d'aller retour, prise sur l'ensemble des paquets acquittés entre la transmission du paquet distingué et la réception de son acquittement (Ack).

#### Phase du *Slow Start*

Le principe est le même que dans le cas de Reno, à la seule différence que Vegas double la taille de sa fenêtre un $RTT$ sur deux, contrairement à chaque $RTT$ dans le cas de Reno. La phase du slow start prend fin une fois que Vegas détecte qu'une file se construit. Ceci est supposé être vrai dès que la fenêtre atteigne une certaine valeur $W_{th}$ pour laquelle la différence $Expected - Actual$ dépasse un certain seuil $\gamma/BaseRTT$, à ce moment Vegas bascule vers sa phase de *congestion*. Cette technique peut permettre d'éviter des pertes pendant la phase du *slow start*, où elles peuvent être de l'ordre de la moitié de la fenêtre dans le cas de Reno.

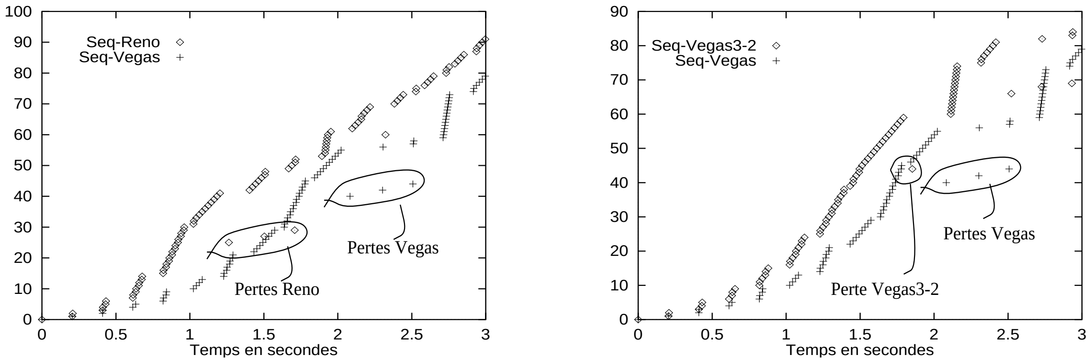
### 3.3.1 Slow Start dans TCP Vegas et TCP Reno

Le fait de doubler la fenêtre une fois tous les deux RTT ne change rien de l'agressivité de la phase du *slow start*. Cette phase, dans beaucoup de cas, prend uniquement plus de temps que Reno et cause exactement les même pertes que ce dernier. En effet, Vegas met entre 0 et 75 % de temps en plus pour transmettre le même nombre de segments que Reno durant cette phase (voir chapitre 4). En s'appuyant sur des simulations, on remarque que les pertes durant cette phase dépendent essentiellement du délai d'aller retour et de la capacité des files d'attente aux routeurs. Quand les délais d'aller retour sont suffisamment grands, les files intermédiaires de petites capacités et le trafic exogène est faible, la taille de la fenêtre pour laquelle les pertes se produisent dans Reno est, souvent, la même que dans Vegas. De plus, le nombre de pertes est également le même.

Par contre, sous les mêmes hypothèses, si au lieu de doubler la fenêtre une fois sur deux RTT on envoie trois paquets à la réception de deux acquittement, on obtient des performances meilleures que celles de Vegas. Autrement dit, durant la phase du *slow start* on envoie un paquet pour le premier acquittement et deux paquets pour le second. La phase du *slow start* ainsi obtenue est plus rapide que celle de Vegas et permet de meilleures performances.

la figure 3.4 (a) illustre un exemple assez simple où une source envoie des segments vers une destination à travers un routeur qu'elle utilise seule (sans trafic exogène). Cet exemple montre qu'on améliore en rien les pertes dans Vegas par rapport à Reno. En effet, on peut remarquer que les pertes ont lieu quand la fenêtre atteint la taille de seize segments pour la première fois dans les deux cas (Reno et Vegas), et le nombre de pertes constatées, dans la même fenêtre, est le même (trois). Ainsi, on peut dire que dans ces situations, hormis le mécanisme de détection de la bande passante dans Vegas durant la phase du *slow start*, il n'y a aucune amélioration par rapport à Reno. Vegas ne fait que retarder les pertes.

La figure 3.4 (b) quant à elle, illustre un exemple du *slow start* dans Vegas et le *slow start* modifié (au lieu de doubler la fenêtre une fois sur deux RTT on envoie deux paquets pour un acquittement une fois sur deux). D'une part, on peut voir qu'il ne se produit qu'une seule




(a) Slow start dnas Reno et Vegas

(b) Slow start dans Vegas et Vegas3-2

FIG. 3.4: *slow start* dans Vegas et Reno

perte par fenêtre. En fait, cela dépend également de la capacité de la file d'attente au niveau du routeur, mais le nombre de pertes est toujours moins important que dans Vegas ou Reno.

La conclusion à tirer de ces constatations est que les pertes durant la phase du *slow start* dépendent essentiellement de la capacité des files d'attente et du seuil du *slow start.* Elle dépendent moins de la bande passante disponible. Si on choisit le bon seuil du *slow start* dans Reno, alors on aurait de meilleures performances que Vegas durant cette phase pour des délais assez grands. D'autres exemples analytiques pour analyser ce cas seront donnés dans le chapitre 4.

**Détection de pertes**

Dans ce qui suit, nous décrirons le mécanisme de retransmission de Vegas et discuterons les différentes techniques utilisées pour détecter les pertes.

Vegas élargie le mécanisme de retransmission de Reno comme suit : Vegas lit et sauvegarde l'instant de transmission de chaque paquet. Quand un acquittement est reçu, Vegas lit l'heure et calcule ainsi le délai d'aller retour moyen (smooth RTT), la déviation moyenne, et le timeout à granularité fine (fine grain timeout (*v_timeout*)). Ce timeout est calculé de la même manière que Reno avec une granularité très fine, non pas un multiple de 500 ms. Le timeout à grosse granularité (coarse timeout) de Reno reste effectif sous Vegas. Vegas décide de retransmettre un paquet dans les situations suivantes :

– A la réception d'un acquittement dupliqué, Vegas vérifie si le temps écoulé depuis la transmission du premier (en séquence) paquet non encore acquitté est plus grand que le timeout à granularité fine (v_timeout). Si c'est le cas, le paquet est alors considéré perdu et retransmis sans attendre la réception du troisième acquittement dupliqué (requis par Reno).

Cette technique parfois permet de détecter des pertes plus rapidement à cause du fait que le temps de service des paquets n'est pas négligeable, et les capacités des files d'attente au niveau des routeurs tendent à être de plus en plus importantes. Ce n'est pourtant pas l'avis de V. Jacobson [57] qui estime que très souvent le mécanisme en question échoue. Les arguments avancés étant que le timeout (v_timeout) doit être au moins égale à 2RTT, au moment où le temps entre le premier et le troisième acquittement dupliqué est typiquement très petit. En d'autre termes, si perte il y a, le temps entre l'acquittement du paquet bien reçu et le troisième acquittement dupliqué ne dépasse pas souvent le $RTT$.

– A la réception du premier ou du deuxième acquittement après une retransmission dûe aux acquittements dupliqués, Vegas vérifie encore une fois si le temps écoulé depuis la transmission du premier paquet non encore acquitté dépasse la valeur du timeout de Vegas (v_timeout). Si c'est le cas, alors Vegas le retransmet.

Cette technique permet de détecter les pertes plus rapidement et de recouvrir les pertes groupées (multiples), sans réinitialiser la phase du *slow start*. Par conséquent, cela permet de palier au problème dont souffre la version Reno, notamment plusieurs pertes dans la même fenêtre de données comme expliqué précédemment. Cette idée est reprise dans TCP New-Reno, comme on le verra par la suite, sans toutefois que la vérification du temps écoulé depuis la transmission soit faite. En effet, New-Reno retransmet dès la réception d'un acquittement non dupliqué acquittant des séquences inférieures à la plus grande séquence transmise (non seulement le premier et le deuxième acquittement non dupliqué après une retransmission).

Les performances de Vegas semblent en effet dépasser celles de Reno. D'autant plus que cette dernière version est moins agressive et cause moins de pertes aux trafics concurrents. Cependant, elles reposent essentiellement sur l'estimation du délai d'aller retour [57]. Si ce délai est mal estimé, on risque soit de saturer le réseau ou de sous estimer la bande passante disponible, ne serait-ce que pour un moment assez court. Ajouter à cela que les performances de Vegas se dégradent en fonction du nombre de connexions ; comme il sera montré dans le prochain chapitre, plus il y a de connexions moins les performances de Vegas sont bonnes.

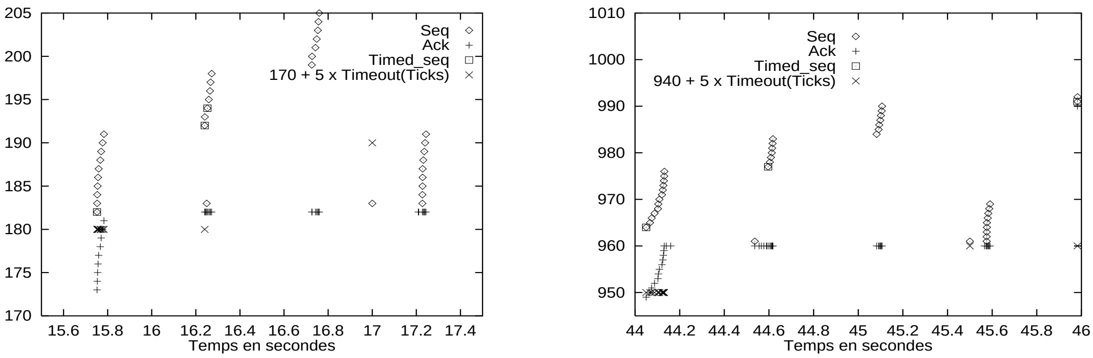
Parmi les autres désavantages à mettre au compte de cette version, le temps que Vegas met pour vérifier et mettre à jour les différents champs concernant les dates d'envoi des paquets [57] (overhead). Ce desavantage est mineur, du moment que cet excès de temps est de l'ordre de quelques dizaines d'operations par seconde [4].

## 3.4 Problèmes dans TCP Reno (Vegas) : vers New-Reno

Les mécanismes de "Fast retransmit" et "Fast recovery", sont connus pour être efficaces dans l'amélioration du débit et la détection rapide des pertes. Toutefois, comme on l'a souligné ci-dessus, ils peuvent être la source de certains dysfonctionnements, et par conséquent causent quelques problèmes. En plus des problèmes relatés ci-dessus, à savoir le problème des "fast retransmit" successifs (successive fast retransmit) et les pertes multiples, nous allons voir dans ce qui suit d'autres problèmes qui surgissent dans Vegas et Reno. Le premier problème concerne l'exécution d'un faux "fast retransmit" suivis d'un faux "fast recovery", après un timeout. Il se produit souvent quand les délais d'aller retour sont de l'ordre de grandeur de la résolution du temporisateur (500 ms) et que les capacités des files d'attente dans les routeurs (dans le réseau) sont suffisamment larges [6]. Le deuxième concerne l'exécution d'un faux "fast retransmit" [48], qui peut être vu comme cas particulier du premier.

### 3.4.1 Problème des faux "fast (retransmit) recovery" après timeout

Ce problème apparaît dans Vegas et Reno, et se traduit par la transmission inutile de plusieurs paquets. Il peut se produire quand on exécute le mécanisme de *fast recovery* (fast retransmit) après l'expiration du temporisateur de retransmission. Un problème similaire "false fast retransmit" est rapporté dans [48] voir section 3.4.2 ; c'est également un cas particulier des "fast retransmit" successifs. Un exemple est illustré dans la figure 3.5 (Vegas), où les instants auxquels les paquets sont transmis et ceux auxquels leurs acquittements arrivent sont représentés. Dans la figure, les acquittements sont représentés par des "plus", les instants de mise à jour du timeout sont représentés par des "X", les séquences des paquets transmis par des "diamants" et enfin, les paquets temporisés pour l'estimation du RTT par des "carrés". Les acquittements ayant la même séquence correspondent à des acquittements dupliqués indiquant qu'une perte a lieu.




(a) Vegas (b) Reno

FIG. 3.5: Problème du "fast recovery" après timeout à grosse granularité (Reno)

Le scénario est le suivant (figure 3.5 (b)) : à l'instant 44.53 la source reçoit trois acquittements (Acks) dupliqués acquittant le paquet 960 ; ce qui traduit la perte du paquet 961, qui est, par conséquent, retransmis après avoir réduit la fenêtre à 11 ( $\lfloor 16/2 \rfloor + 3$). Tant que des Acks dupliqués (acquittant le paquet 961) arrivent, la taille de la fenêtre augmente à raison d'un paquet par Ack reçu (fast recovery) et un nouveau paquet est transmis si la taille actuelle de la fenêtre le permet. Les paquets 977-983 seront ainsi transmis (977 était le dernier paquet transmis avant la réception du premier acquittement dupliqué). Le paquet 961 est une deuxième fois perdu. A l'instant 45.08, on reçoit encore des acquittements dupliqués dûs à la bonne réception des paquets 977-983, et les paquets 984-990 sont alors transmis (toujours dans la phase de "fast recovery").

A l'instant 45.500 le temporisateur expire, la phase du *slow start* est entamée en réinitialisant pratiquement tous les paramètres : la plus grande séquence transmise est considérée comme étant le premier paquet non acquitté, le paquet 961 en l'occurrence, la fenêtre est remise à 1, le seuil du *slow start* à $[30/2] = 15$, car la fenêtre atteint 30 à cause du mécanisme de fast recovery, et enfin le nombre d'acquittements dupliqués reçus est remis à zéro. Le paquet 961 est alors retransmis.

Les acquittements des paquets 984-990, qui étaient bien reçus arrivent juste après l'expiration du temporisateur. A la réception de trois d'entre eux (7 acquittements en tout), le paquet 961 est considéré perdu (faux "fast retransmit") et retransmis à tort. A ce moment, le seuil du *slow start* est mis à $ssthresh = max(2, [Cwnd/2]) = 2$, la fenêtre, quant à elle, atteint 5 ($ssthresh + 3$ acquittements dupliqués) et les paquets 962-965 sont ainsi (re)transmis (fast recovery). La source recevra encore quatre acquittements dupliqués. Dû au mécanisme de "fast recovery", les paquets 966-969 seront retransmis eux aussi à tort.

On remarque que les paquets 961-969 sont retransmis sans qu'ils soient perdus pour autant, et ce à cause de l'exécution du "fast retransmit" suivis d'un "fast recovery" qui ne devrait pas avoir lieu.

#### 3.4.1.1 Solution

La dernière retransmission du paquet 961 peut être évitée en initialisant le nombre d'acquittements dupliqués reçus à trois juste après l'expiration du timeout à grosse granularité (ceci peut se traduire dans le code de TCP (TCP Vegas release 0.8 [27] ou dans BSD.4.3) par $tp \rightarrow t\_dupacks = tp \rightarrow t\_rexmtthresh$ dans (v)tcp_timer.c au lieu de $tp \rightarrow t\_dupacks = 0$).

La solution donnée par S. Floyd [41] comme remède au problème de "fast retransmit" successifs ne peut être solution à ce problème. Même si elle peut éviter la retransmission du paquet 961 pour la quatrième fois, il n'en demeure pas moins que les autres paquets seront retransmis du moment que les paramètres sont pratiquement tous réinitialisés (c'est le premier paquet en séquence non encore acquitté qui est considéré comme étant la plus grande séquence transmise). D'autre part, cette solution n'est pas envisageable pour les raisons citées ci-dessus (cf section 3.2.1).

Le problème de la retransmission des paquets 962-969 peut être résolu en évitant d'exécuter le mécanisme de "fast recovery" après l'expiration du temporisateur de retransmission (i.e. timeout à grosse granularité). Ceci peut se faire en désactivant un indicateur (fast_recovery) quand le timeout expire, et l'activant une fois qu'un acquittement non dupliqué soit reçu (ceci peut se traduire dans le code TCP (TCP Vegas release 0.8 [27] ou BSD.4.3) par l'incrémentation de la fenêtre uniquement si ($tp \rightarrow t\_dupacks > tp \rightarrow t\_rexmtthresh$ && fast_recovery)).

Une deuxième solution que nous proposons consiste simplement à incrémenter la fenêtre à chaque réception d'un acquittement dupliqué, uniquement si la fenêtre actuelle est plus large que le seuil du *slow start*. Cette solution est envisageable du moment que le fast recovery suit toujours le "fast retransmit". Ainsi, si on ne permet pas de fast retransmit, la fenêtre dans ce genre de situation sera toujours inférieure au seuil du *slow start*.

Sous Vegas, ce problème peut également apparaître. On peut avoir exactement le même comportement, si on considère que le scénario est le même pour Vegas jusqu'à l'instant 44.53 (b). La figure 3.5 (a) en illustre un autre exemple pour Vegas.

Il faut tout de même rappeler que la situation retracée dans cette section ne se produit pas si les délais d'aller retour sont suffisamment petits, et que la capacité de stockage à la destination est limitée. A ce moment, ce qui risque de se produire réellement, est que la source continue d'envoyer de nouveaux paquets, dûs au fast recovery, et de recevoir des acquittements dupliqués pour le paquet 960, jusqu'à ce que la fenêtre soit réduite à zéro, à cause de la capacité de stockage au niveau du récepteur. Ainsi, la source rentre dans l'état "persist", enclenché chaque fois que la fenêtre se réduit à zéro, pour une autre raison que celle pour laquelle est prévu cet état (voir section 2.1.4 ou [99, 100]).

Un paquet de taille d'un octet (i.e. formant un datagramme IP de 41 octets) est à ce moment envoyé, et le premier (en séquence) paquet non encore acquitté est considéré comme étant la plus grande séquence envoyée. Le problème pourrait être résolu une fois que tout le segment manquant est reçu par la destination, octet par octet, ce qui prendrait un temps énorme (au minimum deux fois le produit taille d'un segment en octet fois le délai d'aller retour). Ceci se produit dans le cas où la destination acquitte les octets un par un. Cependant, la solution de Clark [33, 100], au problème du **syndrome de la fenêtre stupide** (*silly window syndrome*) empêche une destination d'acquitter octet par octet, du moment que la capacité de stockage à la destination n'a pas changé. Par conséquent, dans notre cas la source retransmet le même octet à chaque expiration du temporisateur de "persistance", et une situation d'**interblocage** (*deadlock*) apparaît. La source demeurera ainsi, dans cet état jusqu'à l'avortement de la connexion, bien que la communication aurait pu se dérouler de la plus belle façon ! Ce qui constitue un problème de plus.

On pourrait penser que le problème de "faux fast (retransmit) recovery" est simplement un problème potentiel dans TCP. Toutefois, beaucoup d'articles suggèrent que la résolution du temporisateur soit réduite pour améliorer le débit dans les réseaux à haut débit (ATM par exemple). Ajouter à cela que les liens satellites peuvent aisément engendrer des délais de l'ordre de 300 ms, ainsi avec une résolution de 300 ms (existante), on peut s'attendre à rencontrer ce genre de dysfonctionnements.

### 3.4.2 Le problème du faux "fast retransmit" : New-Reno

Un problème similaire à celui retracé ci-haut dans Reno est décrit dans [48]. La solution proposée pour y palier a donné naissance à une version de TCP appelée TCP New-Reno. En réalité cette version se rapproche beaucoup plus de Tahoe que de Reno, puisque à chaque détection d'une perte, une "partie" de la phase du *slow start* est exécutée.

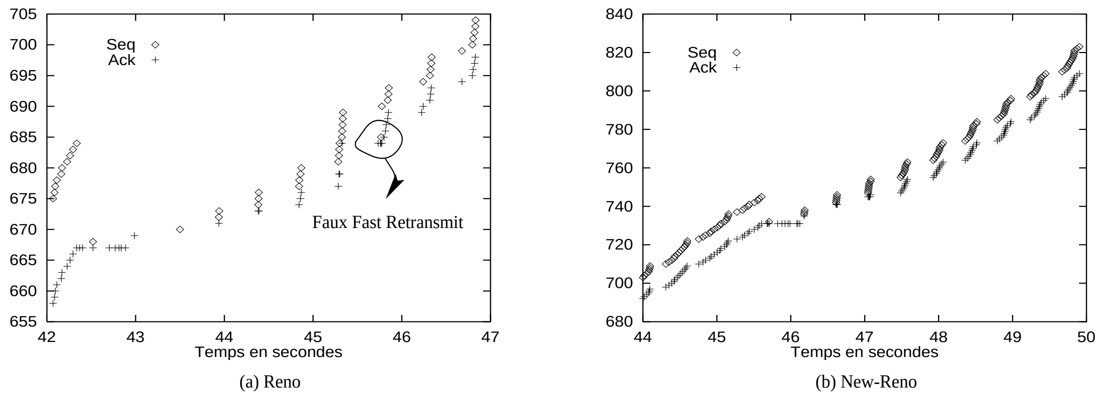



FIG. 3.6: Problème du faux "fast retransmit" dans Reno

Le problème du faux "fast retransmit" est retracé dans la figure 3.6 où figurent les séquences des paquets et de leurs acquittements. Dans cette figure, le scénario est le suivant : Après un timeout à l'instant 43.5, Reno déclenche la phase du *slow start* et retransmet le paquet 670, qui causera la réception de l'acquittement 671. Les paquets 672-673 sont alors (re)transmis, et causeront à leur tour la réception de deux acquittements dupliqués pour le paquet 673, et les paquets 674-676 sont ainsi retransmis (la fenêtre est à 3 et le seuil du *slow start* à 4). A l'instant 44.84, dûs aux acquittements des paquets 674-676, les paquets 677-680 sont retransmis.

L'acquittement 677 (678 était perdu lors de sa première transmission), et deux acquittements dupliqués pour 679 sont renvoyés par la destination à la réception des paquets 677-679 (le paquet 680 était perdu lors de sa première transmission). Ainsi, les paquets 681-684 sont (re)transmis (il étaient bien reçus lors de leurs première transmission) avant la réception de l'acquittement 984 dû à la bonne réception du paquet 980 retransmis. A cet instant, la source transmet les paquets 985-989. A l'arrivée des paquets 981-984 à la destination, celle-ci renvoie encore des acquittements dupliqués pour le paquet 984, qui sera une fois de plus considéré perdu à cause des trois acquittements dupliqués, et par la suite retransmis inutilement.

Le comportement qu'on souhaiterait avoir idéalement dans ce cas de figure, est retracé dans la figure 3.6 (b). C'est en fait le comportement souhaité de New-Reno ; la nouvelle version conçue pour répondre à ce genre de problèmes. Nous verrons dans ce qui suit que cette version comme toutes les autres a sa part de dysfonctionnements.

New-Reno est proposé non seulement pour palier au problème du " faux fast retransmit", mais aussi pour améliorer les performance de Reno dans le cas de pertes multiples. L'idée est de s'inspirer du fonctionnement de Tahoe puisque ce dernier recouvre nettement mieux que Reno les pertes multiples dans la même fenêtre de données.

Il s'agit donc d'utiliser le *slow start*, tout comme Tahoe, lors d'une détection de pertes par acquittements dupliqués, et de recommencer la phase de *congestion* une fois que toute la fenêtre est recouverte (toutes les données d'une même fenêtre de données sont acquittées).

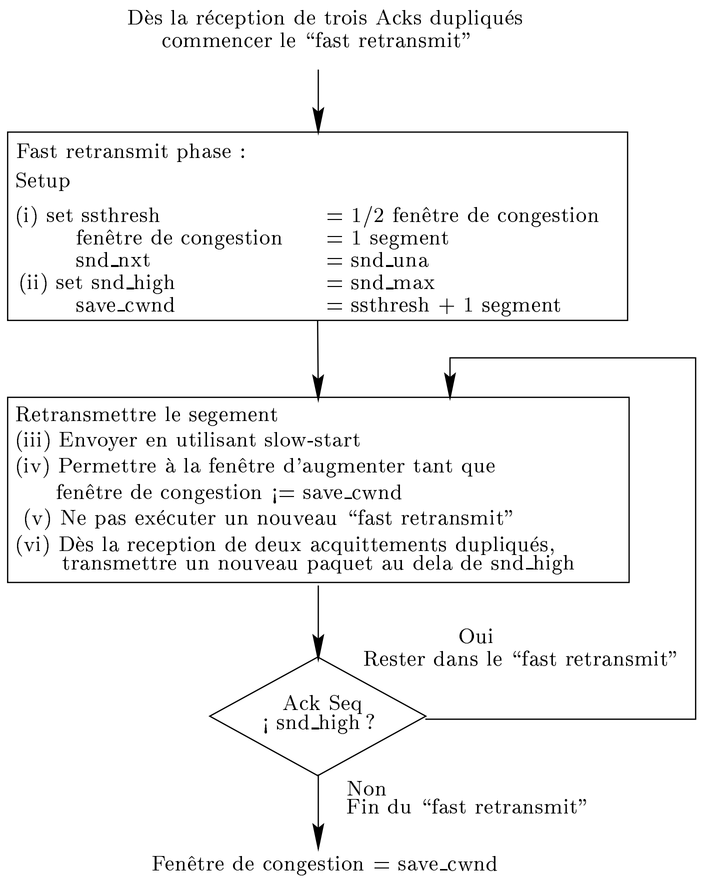



FIG. 3.7: "fast retransmit" dans New-Reno

Les changements apportés à Reno sont résumés dans le diagramme 3.7 où l'essentiel des modifications concernent le mécanisme de "fast retransmit".

### 3.4.3 New-Reno et problème des "fast retransmit" successifs

Nous avons simulé la nouvelle version de TCP (New-Reno) [48], et essayé de déceler les dysfonctionnements qui pouvaient surgir. Nous avons utilisé le même modèle que précédemment. Nous avons constaté que si on considère que la phase de "fast retransmit" se termine dès la réception de l'acquittement du plus grand en séquence, *snd_high* en l'occurrence (tel est le cas dans [48] pp. 276), alors le problème des "fast retransmit" successifs décrit dans la section 3.2.2 pour Tahoe, réapparaît dans cette version.

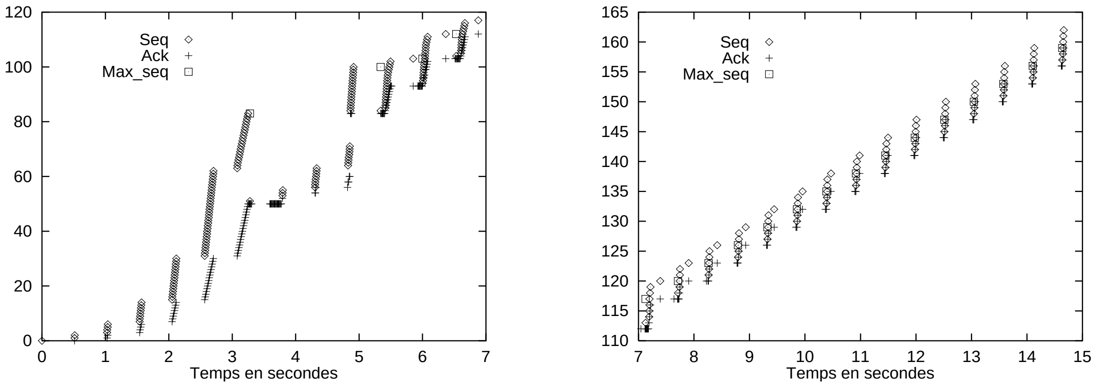



FIG. 3.8: Problème de “fast retransmit” successifs dans New-Reno

La figure 3.8 en illustre un exemple. Dans cette figure, les séquences des paquets transmis sont représentés par des ”diamants”, leurs acquittements par des ”plus” et enfin les séquences qui doivent être acquittées avant de permettre un nouveau “fast retransmit” sont représentées par des ”carrés”. La phase (vi) dans le diagramme de New-Reno (figure 3.7) n’est pas prise en compte dans cet exemple, ainsi on ne permet pas de “fast recovery”. Cela ne change rien à l’apparition de ce problème.

#### 3.4.3.1 Solutions et Remarques

Le problème disparaît par contre si la fin de la phase de “fast retransmit” se termine, non pas une fois que $snd\_high$ est acquitté ($Ack \geq snd\_high$), mais quand le segment envoyé après $snd\_high$ est acquitté ($Ack > snd\_high$).

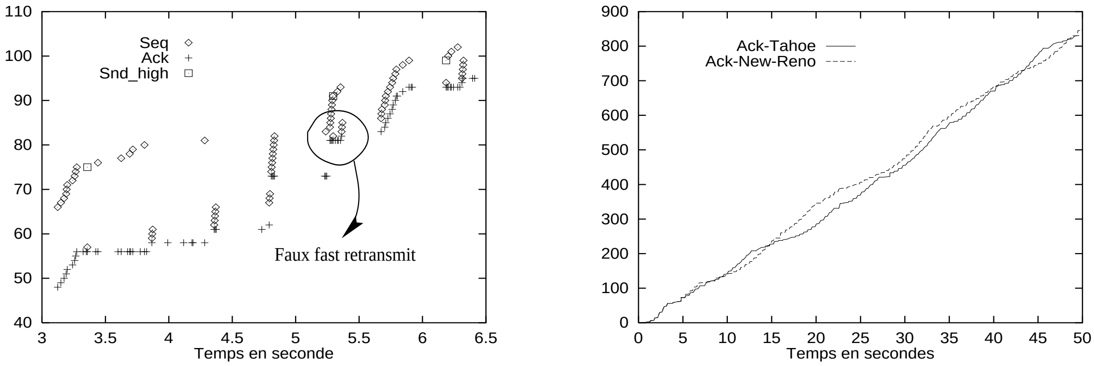
Le problème de retransmissions inutiles n’est toutefois pas résolu dans ce dernier cas, si le “fast recovery” est permis (phase (vi) dans le diagramme 3.7). En effet, la figure 3.9 (a) illustre un exemple où plusieurs paquets sont retransmis inutilement, dûs essentiellement aux acquittements des paquets transmis durant la phase de “fast recovery”. C’est pour cette raison, d’ailleurs, que le fast recovery dans Tahoe risque d’engendrer des situations catastrophiques.

Finalement, avec l’inclusion du “fast recovery”, on peut remarquer que le problème du “faux fast retransmit” n’est pas résolu non plus. Le cercle dans la figure 3.9 (a) en illustre un, et pire encore on retransmet plus d’un paquet inutilement.

Mettre à jour $snd\_high$ à chaque fois qu’un paquet nouveau (au delà de $snd\_max$) est envoyé (de telle sorte que $snd\_high = snd\_max$), peut paraître une solution permettant de résoudre ce problème. Cependant, cette solution risque de ”bloquer” la fenêtre et se trouver indéfiniment dans le *slow start*. La fenêtre risque également d’atteindre $save\_cwnd$ et rester à cette valeur, envoyant ainsi avec une fenêtre restreinte (constante).

Finalement, la seule solution qui nous paraît raisonnable est de ”se passer” du mécanisme de “fast recovery”. Ainsi, on est presque certain que tous les problèmes cités ci-dessus n’apparaîtront plus. Au moins, dans de nombreuses simulations que nous avons faites avec cette version “modifiée”, aucun de ces problèmes n’est rencontré. La version de New-Reno ainsi obtenue est la même que celle décrite dans le diagramme 3.7, sans l’étape (vi) (“fast recovery”) et on test sur $Ack \leq snd\_high$ au lieu de $Ack < snd\_high$.

Si on inclut la solution au problème de “fast retransmit” successifs proposée par S. Floyd [41], il est facile de remarquer que finalement, la version New-Reno qu’on obtient est à notre




(a) Faux ''fast retransmit'' (b)New-Reno vs Tahoe

FIG. 3.9: Faux “fast retransmit” dans New-Reno (a), et comparaison avec Tahoe (b)

avis plus que jamais proche de Tahoe que de Reno. La seule différence entre cette version et Tahoe est que cette dernière oublie de remettre la fenêtre à $ssthreh + 1$ une fois que la plus grande séquence dans la fenêtre est acquittée, et permet également à la fenêtre d'augmenter au delà de $save\_cwnd$.

Par conséquent, il est évident que dans beaucoup de cas, l'ancienne version de Reno dépasse cette nouvelle version en débit. Les mêmes études comparatives entre Reno et Tahoe pourraient être valables pour New-Reno et Reno. La figure 3.9 (b) illustre un exemple où New-Reno et Tahoe ont pratiquement la même allure et permettent le même débit de transmission.

### 3.4.4 Remarques et Propositions

Dans ce qui suit, nous proposons quelques changements à Reno, pour obtenir quelques améliorations. Ces changements s'inspirent de Vegas et concernent essentiellement la détection de perte. On peut penser à introduire exactement les mêmes techniques de détection de pertes de Vegas dans Reno pour recouvrir des pertes multiples. Cependant, ces techniques demandent un peu de temps d'exécution, bien que ça soit négligeable, nous préférons les utiliser autrement.

Dans [48], comme dans beaucoup de papiers, on insiste sur le fait que le seuil du *slow start* doit être minutieusement choisi pour éviter les pertes multiples. Dans Vegas, on propose d'estimer la bande passante disponible en calculant le RTT exact de chaque paquet transmis. Cette dernière solution peut être retenue, si on fait abstraction du temps dépensé dans le calcul des différents débits (débit_actuel, débit_attendu ...etc. Voir section 3.3).

Une autre méthode consiste à estimer la bande passante disponible (ou produit bande passante délai), dès l'établissement de la connexion, et ce par la technique de “packet pair” [88]. En considérant que le délai d'aller retour est facilement mesurable, on peut avoir une estimation de la bande passante en envoyant deux paquets consécutifs. A la réception de leurs acquittements respectifs, le temps entre ces deux acquittements peut donner des indications sur le temps de service dans le réseau. C'est une méthode qui peut donner de bons résultats, si les changements de la densité du trafic sont suffisamment limités. Par contre, elle peut facilement engendrer une surestimation ou une sous-estimation des capacités du réseau, si le trafic est très variable, mais tout cela est une autre histoire!

On a vu dans New-Reno que l'objectif est de retransmettre tous les paquets tant que la plus grande séquence n'est pas encore acquittée, en utilisant le *slow start*. Dans le cas où la première retransmission par "fast retransmit" est perdue une seconde fois, toutes les versions (Tahoe, Reno, Vegas, New-Reno) doivent attendre l'expiration du timeout pour la recouvrir. Ainsi, New-Reno (et les autres) suppose qu'une retransmission est toujours bien reçue. On peut se demander, à ce moment, à quoi pourrait servir l'exécution d'un *slow start* après un "fast retransmit" ?

Il est vrai que l'exécution d'un *slow start* permettrait de recouvrir de plusieurs pertes au même temps, si les pertes ont lieu de façon groupées. Mais il est également vrai que si les pertes sont dispersées (des pertes durant la phase de *congestion* par exemple), alors beaucoup de données bien reçues par la destination seraient retransmises inutilement (problème très connu de Tahoe). Par conséquent, l'agressivité de la phase du *slow start* risque, fort bien, d'engendrer de nouvelles pertes et même la perte des données qu'on essaye de recouvrir. L'interminable attente de l'expiration du timeout serait alors inévitable, du moment qu'un nouveau "fast retransmit" dans la même fenêtre de données n'est pas permis.

#### 3.4.4.1 Proposition

Puisque New-Reno (et les autres) suppose qu'une retransmission est toujours bien reçue, on pourrait alors envisager de retransmettre uniquement les paquets jugés susceptibles d'être perdus. Cela peut se faire en retransmettant un paquet (ou un nombre limité de paquets successifs) à la réception d'un acquittement non dupliqué après une retransmission (technique utilisées dans Vegas).

Dans le cas de retransmission d'un unique paquet, le recouvrement de plusieurs pertes se fera durant un nombre de RTT égale au nombre de pertes. Cela peut constituer un défaut ; cependant, les délais d'aller retour tendent de plus en plus à être très petit (fibre optique) et peuvent être négligés par rapport à la résolution du temporisateur. A rappeler également que le "faux fast retransmit" dans New-Reno est dû aux multiples retransmissions durant le "fast retransmit", en exécutant la phase du *slow start*. C'est la raison essentielle qui justifie le choix de ne retransmettre qu'un seul paquet à la réception d'un acquittement non dupliqué, après une retransmission.

En incluant les solutions de la section 3.4.1, pour éviter le problème du "faux fast recovery", ainsi que la retransmission à la réception d'un acquittement non dupliqué durant un "fast retransmit", on peut raisonnablement améliorer Reno. En effet, la version ainsi obtenue n'empêche pas de "fast recovery" dont l'utilité est tout à fait justifiée.

Les légères modifications apportées à Reno sont décrites dans le diagramme 3.10. Ce diagramme montre l'essentiel des opérations à effectuer à chaque réception d'un acquittement, lors de l'exécution d'une phase de "fast retransmit". Cette dernière commence dès la retransmission d'un paquet, engendrée par la réception de trois acquittements dupliqués, jusqu'à ce qu'un segment plus grand que la plus grande séquence transmise avant cette retransmission, soit acquitté.

A remarquer que lors de la retransmission d'un paquet dûe à la réception d'un acquittement non dupliqué durant la phase de "fast retransmit", la fenêtre reste inchangée. Ceci, pour éviter le problème de réduction de la fenêtre dans Reno lors de multiples pertes (cf section 3.2.1).

Les opérations à entreprendre quand le timeout expire, sont retracées dans la figure 3.11 (solution au problème des "faux fast recovery". Voir section 3.4.1.1) :

On remarquera que les changements ne sont pas substantiels et qu'uniquement le trans-

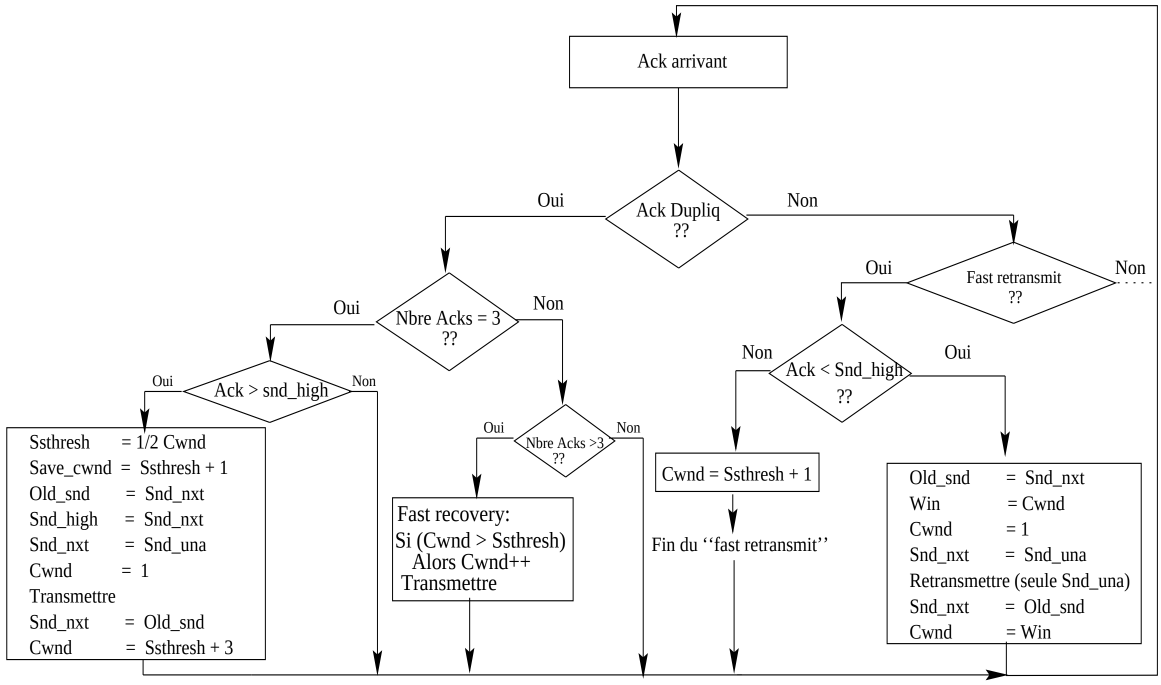



FIG. 3.10: Diagramme de Reno modifiée lors de la réception d'un Ack dans la phase de "fast retransmit"

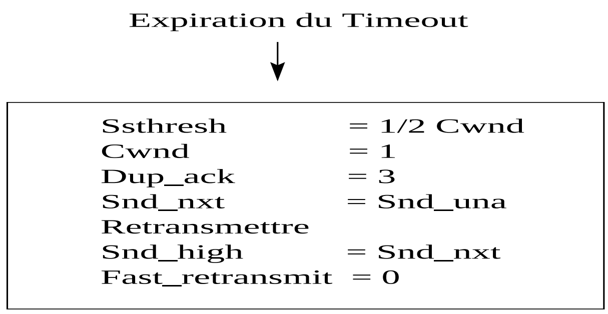
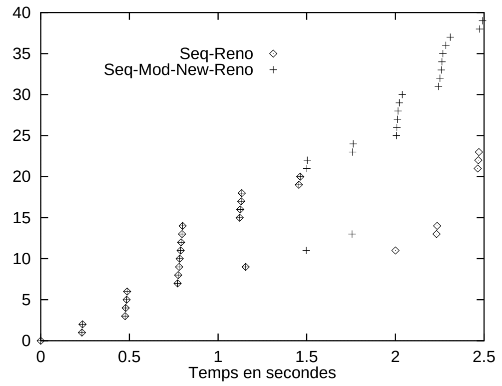



FIG. 3.11: Les opérations à effectuer lors de l'expiration du timeout

metteur est concerné. Les améliorations par rapport à TCP-Reno sont indiscutables, et tous les problèmes décrits le long de ce chapitre ne figurent pas dans cette dernière version de Reno. Cette version de TCP est très proche de TCP Vegas, excepté que Vegas essaye d'estimer la bande passante disponible. Ce qui est vraiment apporté de nouveau dans cette version est la retransmission à la réception d'un acquittement non dupliqué après une première retransmission. Une version similaire à celle décrite ici a été analysée à l'aide des simulations par K. Fall et S. Floyd [39].

La figure 3.12 (a) illustre l'exemple précédent où Tahoe et New-Reno ont pratiquement le même débit de transmission. On peut remarquer que le New-Reno modifié est plus performant que New-Reno. En effet, dans l'exemple en question, plusieurs pertes ont eu lieu (trois). Reno recouvre d'une première, après la réception de trois acquittements dupliqués ; la deuxième et la troisième perte, quant à elles, n'échappent pas au recours au timeout pour les détecter, à l'inverse de New-Reno (modifié).



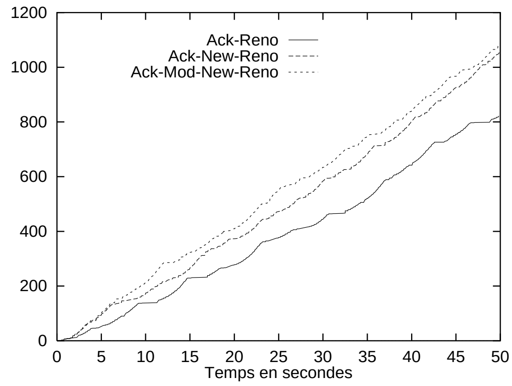




FIG. 3.12: Reno, New-Reno et les modifications apportées à New-Reno

### 3.4.5 Perspectives

Il est facile de vérifier, que lorsqu'une retransmission est perdue sous New-Reno (et toutes les autres versions), le recours au timeout est inévitable pour la recouvrir. On pourrait envisager d'utiliser les acquittements des paquets envoyés durant la phase de "fast recovery" pour y palier. Ainsi, si les paquets transmis durant le "fast recovery" sont bien reçus par la destination, ou du moins avec de très faibles pertes (ex. un paquet sur 4), on pourrait recouvrir de toutes les pertes uniquement avec des acquittements dupliqués. Cela suppose bien évidemment que la destination envoie un acquittement pour chaque paquet dès sa réception.

Le mécanisme de "fast recovery" permet de transmettre de nouveaux paquets si la taille actuelle de la fenêtre le permet, en la faisant croître d'un paquet à chaque réception d'un acquittement dupliqué. Cependant, pour qu'on puisse détecter les pertes ayant lieu pour une deuxième fois, on devrait avoir envoyé des paquets durant cette phase ("fast recovery"). Du moment que la première perte est détectée par la réception de trois acquittements dupliqués, il est donc tout à fait raisonnable de transmettre au moins deux paquets en "fast recovery" pour remplacer les paquets ayant causé la réception de ces acquittements dupliqués. Dans ce qui suit, on propose d'envoyer au minimum trois paquets dans la phase de "fast recovery" en plus des paquets susceptibles d'être transmis durant son exécution "normale".

La proposition est la suivante :

Lors de la réception de trois acquittements dupliqués indiquant le commencement d'une phase de "fast retransmit", on compte le nombre de segments déjà transmis est non encore acquittés, qui est une indication sur le nombre maximal d'acquittements dupliqués qu'on pourrait recevoir après la première retransmission. Soit $Nack\_ancien = Snd\_nxt - Snd\_una - 1$ ce nombre. A la première retransmission, on envoie au minimum trois paquets en "fast racovery", et on compte le nombre de segment transmis durant cette phase, qu'on désigne par $Nack\_nouveau$. On compte également le nombre d'acquittements reçu par la source $t\_dupacks$. Si ce dernier est supérieur à $Nack\_ancien + 2$, alors on décide de retransmettre le premier paquet non encore acquitté. Cela signifie que les paquets envoyés en "fats recovery" continuent d'acquitter le dernier paquet bien reçu, et ça ne peut traduire que la perte du segment retransmis.

A la deuxième retransmission du même paquets, on envoie également au minimum trois paquets en phase de "fast recovery", on met à jour $Nack\_ancien = Nack\_ancien + Nack\_nouveau - 1$, et $Nack\_nouveau = 0$ tout en recommençant à compter le nombre de paquets transmis durant la phase de "fast retransmit" (y compris les retransmissions), on réitère alors la même procédure que précédemment. Tous les paramètres sont remis à zéro une fois qu'un segment plus grand que la plus grande séquence, envoyée juste avant le "fast retransmit", soit acquitté (à la fin de la phase de "fast retransmit").

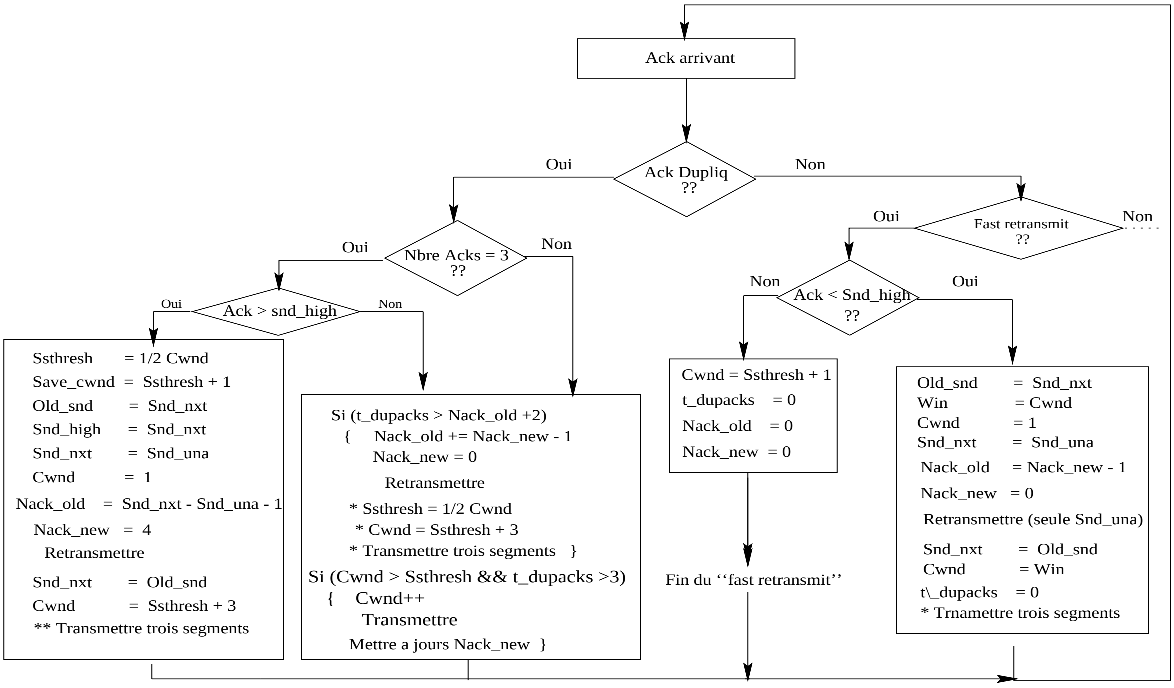



FIG. 3.13: Diagramme pour détecter la perte d'une retransmission

On peut également réduire la fenêtre encore une fois, tout comme dans un "fast retransmit", à la détection d'une perte qui a lieu pour une seconde ou troisième fois. Cela se justifie par le fait que les pertes successives d'un paquet peuvent signifier une très forte congestion dans le réseau. Cela risque cependant de diminuer le nombre de fois qu'une perte puisse être détectée. Les trois paquets transmis durant cette phase, peuvent venir à ce moment pour compenser la réduction de la fenêtre. Le diagramme 3.13 décrit en détails les instructions à exécuter pour recouvrir d'une perte ayant lieu une seconde fois.

Sur Internet, le nombre de RTT nécessaires pour détecter une perte qui a lieu pour la deuxième fois, dépend de la probabilité de perte des paquets transmis en phase de "fast recovery". Toutefois, il est tout à fait intéressant d'envisager de telles méthodes dans les réseaux à haut débit. Dans ces réseaux, le RTT est, généralement, moins d'un dixième de la granularité du timeout. Il est également très connu que le problème de TCP sur ATM réside essentiellement dans la granularité du timeout (500 ms) [95]. Si l'on ajoute à cela le fait que la reception d'un seul acquittement dupliqué signifie une perte dans un réseau ATM, il est alors tout à fait possible de détecter toutes les pertes rien qu'en utilisant les acquittements dupliqués. Cela économiserait un temps considérable et par la même occasion accroîtrait le débit d'une manière spectaculaire.

La figure 3.14 (a), illustre un exemple où une retransmission est perdue, le paquet 360 en l'occurrence, pour une deuxième fois. On peut voir qu'à l'aide de la méthode décrite dans cette section (Seq-Pertes-Suc), il est tout à fait possible de la détecter, pendant que New-Reno (Seq-New-Reno) doit attendre près de deux secondes pour pouvoir la retransmettre, soit près de 20 fois le délai d'aller retour. La figure 3.14 (b), quant à elle, illustre la séquence des acquitements

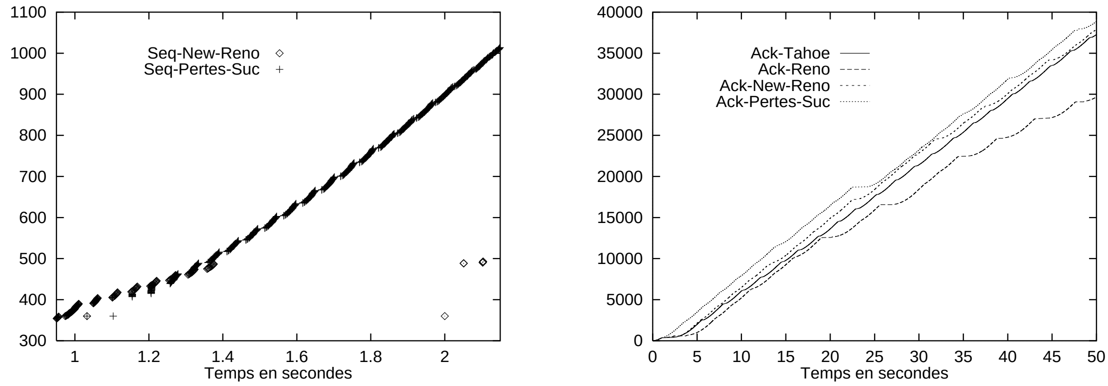



FIG. 3.14: Détection des retransmissions perdues, par acquittements dupliqués

en fonction du temps, pour les différentes versions analysées dans ce chapitre (Tahoe, Reno, New-Reno (modifié) et New-reno avec détection de perte des retransmissions), quand une retransmission est perdue. On peut bien constater que la dernière version excède en débit toutes les autres version, à cause du fait que ces dernières perdent, près de deux secondes pour détecter la perte d'une retransmission (paquet 360). A rappeler que l'amélioration proposée dans cette section ne change rien quant aux performances de New-Reno, en l'absence de retransmissions perdues (la nouvelle version a le même débit que New-Reno (modifié) dans ce cas).

On peut envisager d'utiliser uniquement les paquets envoyés durant le fast recovery par New-Reno, sans en envoyer trois segments après chaque retransmission, comme on l'avait proposé (sans les instructions précédées d'"*" dans le diagramme 3.13). A ce moment, le nombre de fois qu'une perte puisse être détectée se réduit. En effet, dans certains cas l'envoi de paquets après chaque retransmission est essentiel pour recevoir d'autres acquittements, mais avec le risque de saturer le réseau et par conséquent de les perdre lors de fortes congestions. Toutefois, après la première retransmission ("fast retransmit" précédé d'un "**" dans le diagramme 3.13) il est tout à fait raisonnable de transmettre trois segments dans la phase de "fast recovery" pour remplacer les trois acquittements dupliqués reçus.

Dans l'exemple de la figure 3.14, le nombre maximal de segments envoyés à la réception d'un acquittement est limité à quatre paquets, ceci pour éviter les rafales (bursts) lorsqu'un segment beaucoup plus grand que la séquence retransmise est acquitté.

## 3.5 Conclusion

Nous avons vu dans ce chapitre toutes les versions de TCP ayant suscité un grand intérêt. Ainsi, nous avons gardé le lien chronologique et logique entre les différentes versions (Tahoe Reno New-Reno). Nous avons également commenté les nouvelles techniques apportées par Vegas. Cette dernière version, bien qu'elle permette de meilleures performances que Reno, consomme cependant beaucoup de temps dans l'estimation de la bande passante, qui à son tour repose sur l'estimation du plus petit RTT susceptible à des changements en fonction du trafic dans le réseau. Vegas voit ses performances se dégrader également quand le nombre de connexions actives devient important (voir chapitre 4).

Nous avons illustré quelques problèmes par de simples exemples de simulation. Quelques uns de ses problèmes (faux “fast recovery” et faux “fast retransmit”) ont fait l’objet d’une précédente publication [6]. Des problèmes similaires à ceux la ont été décrit dans [48], et une version de TCP, TCP New-Reno en l’occurrence a été proposée. Nous avons inclu les solutions au problèmes de “fast recovery” après timeout dans cette dernière, et discuté quelques détails techniques (problème des “fast retransmit” successifs dans New-Reno).

Enfin,nous avons proposé une méthode permettant de détecter des pertes sans avoir recours au timeout. Cette méthode peut sembler compliquée, mais ne nécessite pas un temps énorme d’exécution. Seules deux variables à rajouter suffisent. Son intérêt dans les réseaux de demain ne serait pas des moindres.

# Chapitre 4

# TCP Vegas *vs* Reno : approche analytique

Nous avons vu dans le chapitre précédent les problèmes techniques des différentes versions de TCP. Les performances de TCP ne sont pas uniquement limitées par ces problèmes, mais par la philosophie de l'algorithme lui même (*slow start* etc...). Plusieurs travaux ont été entrepris pour illustrer les performances de TCP et sa dynamique [22, 89, 88, 105], essentiellement sur Internet.

Souvent de simples modèles de files d'attente permettent une meilleure compréhension d'un algorithme. Ainsi, le modèle à une seule file, pour résumer tout un réseau, a été justifié dans [22] et considéré dans plusieurs articles [16, 72, 73, 87]. Dans [72, 73], une approche fluide pour illustrer l'évolution de la taille de la fenêtre a été présentée. Cette étude concerne le cas général d'un réseau asymétrique où le débit de réception des acquittements n'est pas forcement le débit de service des paquets. Il a également été question de possibles pertes d'acquittements. Dans [72], les pertes aléatoires ont été considérées, et les différentes mesures de performance ont été calculées en fonction de ces probabilités. Une étude comparative à cette dernière est faite dans [87]. Un modèle stochastique est proposé avec prise en compte aussi bien du timeout que des acquittements dupliqués pour la détection de pertes. La version Reno a été considérée, et les délais d'aller retour sont considérés indépendant de la taille de la fenêtre. Les résultats obtenus son validés par des mesures sur Internet. Toutefois, les modèles étudiés dans les précédents articles ne prennent pas en compte explicitement le flux exogène, ajouter à cela que la version Vegas de TCP n'a pas été traitée.

Dans [16], le trafic exogène a été pris en compte en considérant une source contrôlée partageant un goulot d'étranglement avec une source constante. L'approche fluide a été utilisée et les différentes mesures de performance pour Tahoe ont été calculées. Il est apparu que le flux généré par TCP se comporte comme un flux à faible priorité devant le trafic exogène.

Les études qui ont été faites jusqu'à présent pour évaluer les performances de Vegas se sont limitées à des simulations [26] ou des expériences sur Internet [4]. Dans le présent chapitre, nous utilisons une approche analytique dans le but d'analyser les différentes facettes de Vegas et de Reno. Nous comparerons Vegas à l'algorithme Reno, essentiellement la phase du *slow start*. Quand la bande passante est assez large, nos résultats confirment les performances rapportées dans [4, 26]. En effet, Vegas peut retransmettre moins d'un cinquième de ce que retransmet Reno. Ainsi, plus la bande passante est large, meilleures sont les performances de Vegas. Cependant, sous de fortes congestions Vegas a le même comportement que Reno. Les résultats analytiques qu'on obtient sont : l'évolution de la taille de la fenêtre, les délais d'aller retour et leurs moyennes, et le débit moyen. Nous proposons également quelques améliorations permettant d'avoir moins de pertes dans la phase du *slow start*, et moins de paquets aux niveau des files d'attente.

## 4.1 Remarques et définitions

Dans la présente analyse, nous allons considérer des instants discrets (discrétiser le temps) $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2...$, où un intervalle $(\mathcal{T}_n, \mathcal{T}_{n+1})$ est appelé *mini-slot* (voir ex. [72]). Les $\mathcal{T}_n$ sont définis, récursivement, comme suit : $\mathcal{T}_0 := 0$ ; à l'instant $\mathcal{T}_0$ TCP envoie le premier paquet. Celui-ci est appelé la première génération de paquets. $\mathcal{T}_1$ est défini comme étant l'instant auquel l'acquittement du premier paquet arrive à la source. Dans toutes les version de TCP, toute arrivée d'un acquittement provoque la transmission de zéro, un ou deux paquets (selon la phase exécutée). On définit la $n+1$ème génération de paquets comme étant les paquets envoyés dûs à l'arrivée des acquittements de la $n$ème génération. $\mathcal{T}_{n+1}$ est l'instant de réception de l'acquittement du dernier paquet de la $n$ème génération. On notera $W_n$, le nombre de paquets dans la $n$ème fenêtre et $\delta_n = \mathcal{T}_{n+1} - \mathcal{T}_n$ la durée du $n$ème mini-slot.

Dans ce qui suit on utilisera un modèle qui décrit l'évolution du système en fonction de la génération $n$. A l'intérieur de chaque mini-slot $\delta_n$, l'approche fluide est utilisée. Plus précisément, on utilise le fait, établi dans [16, 72] pour la phase du *slow start*, que les paquets sont transmis en *bursts* ou en *trains*, où chaque train représente une génération. Des périodes de silence peuvent exister entre les trains, puisque l'intervalle de temps entre la transmission du premier paquet de deux générations consécutives est le délai d'aller retour (fixe) $T$ (défini ci-dessous). Comme il sera montré plus tard, une situation similaire dans la phase de congestion peut également avoir lieu.

Dans les deux phases, cette situation a lieu tant que la période de transmission d'une génération est inférieure au délai d'aller retour $T$. Elle est caractérisée par le fait que le premier paquet d'une génération trouve une file vide. Une fois que la période atteint $T$, on aura une transmission continue de paquets (pas de périodes de silence), et la file ne se videra plus. Dans ce qui suit, on va approcher la transmission de paquets durant un "train" par un flux de "fluide" continu qui a un débit de $Thp_{in}(W_n)$. $Thp_{in}(W_n)$ est zéro pendant les périodes de silence. Cette approximation implique, en particulier, que le nombre de paquets dans la file est une valeur réelle (au lieu d'un entier) et peut être vu comme un fluide.

## 4.2 Modèle

On considère deux sources transmettant vers la même destination et partageant un goulot d'étranglement. Une des sources est contrôlée (TCP), l'autre est modélisée par une source à débit constant (sans contrôle). Les paquets TCP sont considérés de taille fixe (la taille de 576 octets par défaut est prise dans nos simulations).

On utilisera les même notations que dans [16]. On défini

$W_{max} \triangleq$ La taille maximale de la fenêtre dans Reno, avant qu'une perte ait lieu.

$W_{vmax} \triangleq$ la taille de la fenêtre à laquelle Vegas se stabilise.

$W^v_{max} \triangleq$ La taille maximale de la fenêtre atteinte par Vegas (maximum de $W_{vmax}$).

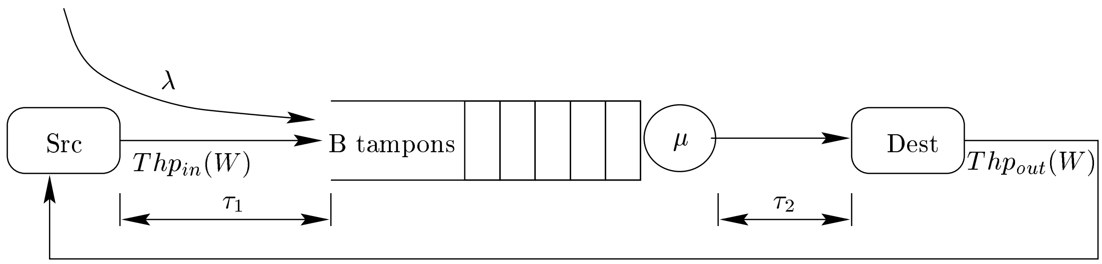



FIG. 4.1: Modèle d'une source avec un trafic exogène

$Q(W_n) \triangleq$ Le nombre de paquets dans la file (routeur), quand le premier paquet de la $n$ème fenêtre y arrive.
$B \triangleq$ La capacité maximale de la file d'attente (routeur).
$\mu \triangleq$ La capacité de service au routeur (paquets/seconds).
$\lambda \triangleq$ Le débit de transmission des paquets exogènes. On suppose que $\lambda < \mu$.
$Thp_{in}(W_n) \triangleq$ Le débit de transmission des paquets de la $n$ème fenêtre.
$Thp_{out}(W_n) \triangleq$ Le débit auquel les paquets de la $n$ème fenêtre sont servis.
$C \triangleq$ Le débit maximal auquel la source contrôlés peut transmettre (ex. due à la vitesse de transmission de la ligne). On suppose que $C$ est large (de telle sorte que les transmissions ne sont pas ralenties par la capacité du lien).
$\tau_1 \triangleq$ Le temps entre l'instant de transmission d'un paquet et l'instant de son arrivée à la file (routeur).
$\tau_2 \triangleq$ Le temps entre l'instant de départ d'un paquet de la file et l'instant d'arrivée à la destination.
$\tau = 2(\tau_1 + \tau_2) \triangleq$ Délai d'aller retour quand la file est vide (i.e. Le temps entre l'instant de transmission d'un paquet et l'instant de réception de son acquittement, moins le temps de séjour et de service dans la file).
$T \triangleq \tau + \mu^{-1}$ "Temps de séjour" d'un paquet dans un système vide i.e. le délai d'aller retour plus le temps de service $\mu^{-1}$ (le plus petit temps de séjour).
$\overline{RTT}_n \triangleq$ Le "temps de séjour (dans le réseau)" moyen pour les paquets de la $n$ème fenêtre.
$\overline{Thp} \triangleq$ Le débit moyen de transmission.
$\delta_n \triangleq$ Le temps que prend la $n$ème fenêtre ($W_n$) pour être acquittée. C'est le temps séparant l'instant de réception du dernier acquittement de la fenêtre précédente ($W_{n-1}$) et l'instant d'arrivée de l'acquittement du dernier paquet de la $n$ème fenêtre.
$\epsilon_{W_n} \triangleq \overline{RTT}_n - T =$ L'excès de temps, du "temps de séjour" moyen sur la fenêtre $W_n$ par rapport au minimum "temps de séjour".

Tous les débits et bandes passantes sont exprimés en paquets par seconde.

## 4.3 Résultats essentiels

Notre approche fluide implique la propriété suivante :

– **(Q0)** La file ne se vide pas durant la période où les trains de paquets contrôlés y arrivent.

Ceci peut se montrer par récurrence. Quand le burst de taille deux arrive à la file, initialement, les deux paquets sont espacés d'une période de temps de $1/C$. Par conséquent, le débit de transmission des paquets contrôlés dans le système est $C \geq \mu - \lambda$. Le débit auquel les paquets sont servis est borné inférieurement par $\mu - \lambda$. Le débit $\mu - \lambda$ serait obtenu si le trafic exogène est plus prioritaire que le trafic contrôlé. Ainsi, les acquittements retournent à la source à un débit pas plus petit que $\mu - \lambda$. Ce qui implique que, dans la génération suivante, les paquets seront transmis à un débit pas plus petit que $\mu - \lambda$. Supposons à présent que les $W_n$ paquets de la $n$ème génération sont transmis à un débit pas plus petit que $\mu - \lambda$. Alors, en utilisant les mêmes arguments, leurs acquittements reviennent à la source avec un débit pas plus petit que $\mu - \lambda$. Ce qui permet de déduire, que les $W_{n+1}$ paquets de la prochaine génération seront transmis à un débit pas plus petit que $\mu - \lambda$ (même dans le cas où $W_{n+1} < W_n$, qui peut arriver dans Vegas).

### 4.3.1 Délai d'aller retour et durée d'un mini-slot en fonction de $W_n$

Tout d'abord, on calculera $\delta_n$ le temps nécessaire pour recevoir tous les acquittements de la $n$ème fenêtre, et $\overline{RTT}_n$ le "temps de séjour moyen" sur la fenêtre $W_n$, dans la phase de congestion et du *slow start*. La preuve du Théorème suivant est donnée dans l'annexe A.1.

**Théorème 4.3.1** *(i) Considérons la phase de* congestion. *Dans Vegas, quand la fenêtre reste constante (se stabilise) on a*

$$\overline{RTT}_n = T + \epsilon_{W_n} = \frac{W_n}{\mu - \lambda} = \delta_n. \tag{4.1}$$

*Pour Vegas et Reno, quand la fenêtre augmente, on a*

$$\delta_n \cong \begin{cases} T + \dfrac{1}{\mu - \lambda} & si \quad 1 \leq W_n - 1 \leq T(\mu - \lambda) \\ \\ \dfrac{W_n}{\mu - \lambda} & ailleurs, \end{cases} \tag{4.2}$$

$$\overline{RTT}_n \cong \begin{cases} T & si \quad W_n - 1 \leq T(\mu - \lambda) \\ \\ \dfrac{W_n(W_n - 1)}{W_n(\mu - \lambda) + \lambda} & ailleurs \end{cases}$$

*$W^v_{max}$ et $W_{max}$ sont données par*

$$W^v_{max} = \left(\frac{B}{\mu} + T\right)(\mu - \lambda) \tag{4.3}$$

$$W_{max} = \begin{cases} \dfrac{\mu + B\lambda}{\mu - B(\mu - \lambda)} & si \quad \lambda > \dfrac{(B-1)\mu}{B} \quad et \quad W_{max} - 1 \leq T(\mu - \lambda) \\ \\ \left(\dfrac{B}{\mu} + T\right)(\mu - \lambda) & ailleurs \end{cases} \tag{4.4}$$

*(ii) Dans la phase* slow start, *on a pour Reno*

$$\delta_n = \begin{cases} T + \dfrac{W_n}{2\mu - \lambda} & si \quad T(\mu - \lambda/2) \geq W_n/2 \geq 1 \\ \\ \dfrac{W_n}{\mu - \lambda/2} & ailleurs \end{cases} \tag{4.5}$$

$$\overline{RTT}_n = \begin{cases} T + \dfrac{W_n}{4(\mu - \frac{\lambda}{2})} - \dfrac{1}{2C} & si \quad T(\mu - \lambda/2) \geq W_n/2 \\ \\ \dfrac{3W_n}{4\mu - 2\lambda} - \dfrac{1}{2C} & ailleurs \end{cases} \tag{4.6}$$

*(iii) Pour Vegas, dans la phase* slow start, *puisqu'il double sa fenêtre un $\delta$ sur deux, on distingue deux cas.*
*Le cas où Vegas double sa fenêtre :*

$$\delta_n = \delta_n^{(1)} = \begin{cases} T + \dfrac{W_n\mu + \lambda}{2\mu^2 - \lambda^2} & si \quad T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_n/2 \geq 1 \\ \\ \dfrac{W_n(2\mu + \lambda)}{2\mu^2 - \lambda^2} & ailleurs \end{cases} \tag{4.7}$$

$$\overline{RTT}_n = \overline{RTT}_n^{(1)} = \begin{cases} T + \dfrac{W_n\mu + \lambda}{4\mu^2 - 2\lambda^2} - \dfrac{1}{2C} & si \quad T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_n/2 \\ \\ \dfrac{W_n(3\mu + 2\lambda) - \lambda}{4\mu^2 - 2\lambda^2} - \dfrac{1}{2C} & ailleurs \end{cases} \tag{4.8}$$

*et le cas où la fenêtre reste constante, où on a*

$$\delta_n = \delta_n^{(2)} = \begin{cases} T + \dfrac{(W_n - 1)\lambda}{2\mu^2 - \lambda^2} & si \quad T\dfrac{2\mu^2 - \lambda^2}{2\mu + \lambda} \geq W_n \\ \\ \dfrac{W_n(\mu + \lambda)}{\mu^2 - \lambda^2/2} & ailleurs \end{cases} \tag{4.9}$$

$$\overline{RTT}_n = \overline{RTT}_n^{(2)} = \begin{cases} T + \dfrac{(W_n - 1)\lambda}{4\mu^2 - 2\lambda^2} & si \quad T\dfrac{2\mu^2 - \lambda^2}{2\mu + \lambda} \geq W_n \\ \\ \dfrac{W_n(2\mu + \frac{3}{2}\lambda) + \frac{\lambda}{2}}{2\mu^2 - \lambda^2} & ailleurs \end{cases} \tag{4.10}$$

### 4.3.2 Slow Start dans Reno et Vegas

Des équations (4.7) et (4.9), on déduit que

$$\frac{2W_n}{\mu - \lambda/2} \leq \delta_n^{(1)} + \delta_n^{(2)} = \frac{2W_n}{\mu - \lambda/2}\left(\frac{1}{1 - \frac{1}{4}\frac{\mu\lambda + \lambda^2/2}{(\mu - \lambda/2)(\mu + 3\lambda/4)}}\right) \leq \frac{3.5W_n}{\mu - \lambda/2} \tag{4.11}$$

Cette relation montre l'excès de temps que met Vegas par rapport à Reno pour transmettre le même nombre de paquets dans la phase du *slow start*, et ce quand la file ne se vide jamais (il met entre 0- 75% de temps en plus que Reno). Cet excès de temps permet à Vegas d'avoir moins de données dans le réseau et par conséquent une plus grande probabilité d'éviter les pertes successives, qui peuvent avoir lieu dans le cas de Reno. Cela se produit essentiellement et surtout quand le trafic généré par Vegas est continue (la file ne se vide jamais). En effet, dans ce cas, Vegas laisse le temps à la file de se vider en gardant sa fenêtre constante durant un RTT. Ainsi, lors du doublement de la fenêtre, les paquets envoyés trouveront une file de taille moins importante que dans le cas de Reno.

Toutefois, dans le cas où la file se vide après chaque RTT, l'unique raison qu'il ait des débordements de la file (des pertes) est le débit de transmission de la source, ainsi que la taille du burst transmis (fenêtre). Du moment que Vegas double sa fenêtre durant un certain RTT, cela peut causer exactement les mêmes effets que dans Reno (à savoir les mêmes pertes, pour les mêmes tailles de fenêtre). En effet, le débit de transmission de Vegas durant cette phase quand la fenêtre est doublée, peut être le même que dans Reno. Cela dépend de l'intensité du trafic exogène. Par conséquent, le fait de doubler la taille de la fenêtre une fois sur deux RTT, dans ce cas précis, n'améliore pas les performances de TCP. Rappelons que dans le chapitre 3 (section 3.3.1), nous avons montré par un exemple de simulation que les pertes durant la phase du *slow start* dans Vegas peuvent être du même ordre que dans Reno.

On peut voir cela à partir des résultats du théorème précédent. En effet, d'après les équations (A.8) et (A.15) (dans l'annexe), il est facile de vérifier que lorsque Vegas double sa fenêtre, le rapport du débit moyen de Vegas sur le débit moyen de Reno n'est pas très petit, si le trafic exogène est assez sporadique. Ce rapport détermine l'agressivité de Vegas par rapport à Reno, on a ($Thp_{Reno}(W_n)$ pour Reno et $Thp_{Vegas}(W_n)$ pour Vegas)

$$\begin{aligned}\frac{Thp_{Vegas}(W_n)}{Thp_{Reno}(W_n)} &= \frac{2\mu^2-\lambda^2}{(2\mu-\lambda)(\mu+\lambda)} = \frac{2\mu^2-\lambda^2}{2\mu^2-\lambda^2+\mu\lambda} \\ &= 1-\frac{\mu\lambda}{2\mu^2-\lambda^2+\mu\lambda} \to 1 \quad \text{quand} \quad \lambda \to 0. \end{aligned} \tag{4.12}$$

Il est facile de prouver à partir de l'équation (4.12), que

$$0.5 < \frac{Thp_{Vegas}(W_n)}{Thp_{Reno}(W_n)} \leq 1 \ , \ \forall\ \lambda < \mu. \tag{4.13}$$

Quand la file se vide après chaque RTT, on a

$$\frac{\delta_{n+1}}{\delta_n^{(1)}+\delta_n^{(2)}} = \frac{T+\frac{2W_n}{2\mu-\lambda}}{2T+\frac{(\mu+\lambda)W_n}{2\mu^2-\lambda^2}} \Longrightarrow 0.5 \leq \frac{\delta_{n+1}}{\delta_n^{(1)}+\delta_n^{(2)}} \leq 0.5+\frac{3W_n}{4T(2\mu-\lambda)+2W_n},\ \forall\ \lambda<\mu.$$

De l'équation (4.9), on a

$$\frac{W_n}{2\mu^2-\lambda^2} \leq \frac{T}{2\mu+\lambda} \Longleftrightarrow W_n \leq \frac{2\mu^2-\lambda^2}{2\mu+\lambda}T. \tag{4.14}$$

En substituant cette relation dans la précédente équation, on obtient finalement

$$0.5 \leq \frac{\delta_{n+1}}{\delta_n^{(1)}+\delta_n^{(2)}} \leq 1-\frac{1}{5-\frac{3\lambda^2}{2\mu^2}} \leq 0.8,\ \forall\lambda<\mu. \tag{4.15}$$

C'est le rapport entre le temps que met Reno et celui que met Vegas, pour transmettre le même nombre de paquets durant la phase du *slow start*. On peut remarquer que ce rapport dépend du RTT pendant que l'agressivité n'en dépend pas (les débits de transmissions dépendent du débit de service). Aussi, il est facile de vérifier que pour des RTT assez grand le gain en agressivité est très loin de ce qu'on perd en temps. On peut perdre au mieux 25% du temps en plus que Reno, et au pire 100% de plus. D'autre part, il est facile de vérifier à partir de l'équation (4.14) que $W_n \leq (\mu - \lambda/2)T$. En utilisant ce fait, on peut obtenir la relation suivante :

$$\text{Si} \quad \lambda \leq (\sqrt{6} - 2)\mu \quad \text{alors} \quad \frac{\delta_{n+1}}{\delta_n^{(1)} + \delta_n^{(2)}} < \frac{Thp_{Vegas}}{Thp_{Reno}}.$$

A noter également que même quand la file ne se vide jamais (transmission de la source en continu), l'agressivité de Vegas par rapport à Reno, définie précédemment, est dans tous les cas plus grande que le rapport de temps dépensé dans le *slow start* (équation (4.11)) entre Reno et Vegas.

Pour ces raisons, il serait plutôt préférable d'accroître la fenêtre d'un facteur de 1.5 à chaque RTT, au lieu de la doubler une fois sur deux RTT. On gagnerait à ce moment, et en agressivité (un *slow start* moins agressif) et en temps de réponse. En effet, on transmettra plus de données durant la même période de temps. Aucune étude comparative de cette dernière solution, par rapport au *slow start* dans Vegas n'est faite. Par conséquent, on pourrait simplement dire que cette proposition améliore les performances du *slow start* dans le cas où les RTT sont importants, ce qui est le cas sur Interne. En effet, dû à la variabilité du trafic sur Internet (par exemple), les pertes se produisent avant que la taille de la fenêtre atteigne la bande passante multipliée par le délai (i.e avant que les liens se remplissent). Les fenêtres sont ainsi envoyées en rafales ("bursts") (les fenêtres de données successives sont séparées par des périodes de silence).

### 4.3.3 Seuil du Slow Start dans Vegas et la taille maximale de la fenêtre

Dans ce qui suit, on calculera la taille maximale de la fenêtre, et la taille de la fenêtre à la fin de la phase du *slow start* pour Vegas, qui facilitera le calcul du débit moyen.

La preuve du corollaire suivant est donnée dans l'annexe A.2.

**Corollaire 4.3.1** *Considérons Vegas. (i) La fenêtre se stabilise quand elle atteint la taille de $W_{vmax}$, telle que*

$$T(\mu - \lambda) + \alpha < W_{vmax} \leq T(\mu - \lambda) + \beta \tag{4.16}$$

*(ii) Le seuil du* slow start *est donné par*

$$W_{th} = 2^{\lfloor \log_2 Min_{th} \rfloor + 1} \tag{4.17}$$

*où $\lfloor x \rfloor$ est la partie entière de $x$, et*

$$Min_{th} = \begin{cases} \frac{(\gamma\mu - \lambda)C + 2\mu^2 - \lambda^2}{2C\mu} \times \\ \qquad \left(1 + \sqrt{1 + \frac{4\gamma C\mu((2CT-1)(2\mu^2-\lambda^2) + C\lambda)}{((\gamma\mu - \lambda)C + 2\mu^2 - \lambda^2)^2}}\right) & si \ \ T\frac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_{th}/2 \\ \frac{(3\gamma\mu + (2\gamma+1)\lambda)C + (2CT+1)(2\mu^2 - \lambda^2)}{2C(3\mu + 2\lambda)} \times \\ \qquad \left(1 + \sqrt{1 - \frac{4\gamma(C\lambda + 2\mu^2 - \lambda^2)(3\mu + 2\lambda)}{((3\gamma\mu + (2\gamma+1)\lambda)C + (2CT+1)(2\mu^2-\lambda^2))^2}}\right) & ailleurs \end{cases} \tag{4.18}$$

**Remarque 4.3.1** *La taille du seuil du* slow start *est calculée par Vegas, en se basant sur l'estimation de la différence entre le débit actuel et le débit attendu. Cette estimation est faite pour les fenêtres précédentes (quand la fenêtre a été doublée). Même si cette différence n'est pas très importante, pourvu qu'elle dépasse $\gamma = 1$, Vegas décide de* <u>doubler</u> *la fenêtre dans le prochain $RTT$.*

*Le débit de transmission est à ce moment deux fois le débit de service des paquets (voir section précédente), et la file se construit à un débit égale à ce dernier. Dans le cas où l'estimation est faite pour une fenêtre assez large (bande passante assez large), lors de son doublement le nombre de paquets qui seront empilés dans le routeur est autour de la moitié de cette fenêtre. Ceci peut causer des pertes si les files d'attente sont de petites capacités. Par conséquent, si le produit délai $\times$ bande passante est assez grand et les capacités des files d'attente au niveau des routeurs sont assez petites, les pertes durant la phase du* slow start *dans Vegas sont très probables. Cela est essentiellement dû au fait de "doubler" systématiquement la fenêtre (le seuil du* slow start *est dans tous les cas une puissance de deux).*

### 4.3.4 Délai d'aller retour et débit moyens

On calculera le débit moyen pour Reno et Vegas, sous les hypothèses suivantes :
**(i)** La bande passante est assez large ($\mu - \lambda > \alpha\mu/B$ voir Théorème 4.3.2), alors la fenêtre de Vegas se stabilise quand elle atteint $W_{vmax}$.
**(ii)** Durant la phase de *congestion*, il ne peut y avoir qu'au plus une perte par fenêtre de données dans Reno, et quand elle a lieu, elle est détectée par acquittements dupliqués.

A noter, que l'hypothèse (ii) implique qu'il y a seulement quelques phases de *slow start* dans une session (nombre fini au début de la session), après quoi c'est un comportement périodique dans la phase de *congestion* qui prend place. L'hypothèse (i) décrit en fait le comportement souhaité de Vegas. Le cas où l'hypothèse (i) n'est pas vérifiée est reporté au Théorème 4.3.2. On peut également analyser les situations où l'hypothèse (ii) n'est pas vérifiée, en la remplaçant par d'autres hypothèses (voir [73]).

Le Théorème 4.3.1 (i) implique ce qui suit :

**Corollaire 4.3.2** *Considérons Vegas et l'hypothèse (i) ci-dessus. La fenêtre se stabilise à* $W = W_{vmax}$. *Le débit moyen est* $\overline{Thp} = \mu - \lambda$, *et le "temps de séjour" moyen est, alors*

$$\overline{rtt} = W_{vmax}/(\mu - \lambda). \tag{4.19}$$

*Considérons Reno sous l'hypothèse (ii) ci-dessous. Alors, on distingue deux cas :*
*1)*$(W_{max} + 1)/2 > T(\mu - \lambda)$ *: (ceci correspond au cas où la file ne se vide pas)*

*Le débit moyen* $(\overline{Thp})$, *la durée d'un cycle* $(T_1)$, *le nombre total de paquets transmis* $(N_1)$ *et le "temps de séjour" moyen* $(R_1)$ *sont, respectivement, donnés par*

$$\overline{Thp} \cong \mu - \lambda, \quad T_1 = \frac{3}{8}\frac{(W_{max}+1)(W_{max}+3)}{(\mu-\lambda)}, \quad N_1 = \frac{3}{8}(W_{max}+1)(W_{max}+3),$$

$$R_1 = \frac{8T_1}{9N_1}W_{max}.$$

*2)* $(W_{max} + 1)/2 \leq T(\mu - \lambda)$ *: La durée d'un cycle est donnée par* $T_1 = T_{21} + T_{22}$ *où*

$$T_{21} = \left(T + \frac{1}{\mu - \lambda}\right)\left(\frac{2T(\mu-\lambda) + 1 - W_{max}}{2}\right),$$

$$T_{22} = \frac{(W_{max}+1)^2 - T^2(\mu-\lambda)^2 + W_{max} - T(\mu-\lambda) + 1}{2(\mu-\lambda)}.$$

*Le nombre de paquets transmis (transmis et acquittés) dans un cycle est donné par* $N_1 = N_{21} + N_{22} = \frac{8T_1}{9N_1}W_{max}$, *où*

$$N_{21} = \frac{(2T(\mu-\lambda) + 1 - W_{max})(2T(\mu-\lambda) + 1 + W_{max})}{8}, \quad N_{22} = (\mu-\lambda)T_{22}$$

*et le "temps de séjour" moyen (par paquet) est* $R_1 = \dfrac{N_{21}T + N_{22}R_{22}}{N1}$ *où*

$$R_{22} = \frac{2T_{22}}{3N_{22}}(W_{max} + T(\mu-\lambda) + 1) - \frac{(W_{max} + 1 - T(\mu-\lambda))(W_{max}+1)(T(\mu-\lambda)+2)}{3N_{22}(\mu-\lambda)}$$

*Le débit moyen est, finalement, donné par* $\overline{Thp} = N_1/T_1$. ∎

La preuve de ce corollaire est donnée dans l'annexe A.3.

#### 4.3.4.1 Simulations

Dans la table 4.1, sont résumés les différents résultats de plusieurs simulations et ceux obtenus analytiquement pour le $RTT$, $W_{th}$ et la taille maximale de la fenêtre $W_{vmax}$ sous $Vegas_{2,4}$. Les paramètres choisis, sont tels que la fenêtre de Vegas se stabilise quand elle atteint $W_{vmax}$.

On peut constater que les résultats analytiques sont très proches des simulations. L'erreur relative $|analytic - simulation|/simulation$ dans tous les cas, n'excède pas les 6% pour la taille maximale de la fenêtre et les 0.02% pour le $RTT$. En fait, la taille maximale de la fenêtre obtenue dans les simulations est un entier, en d'autres termes on a

$$[T(\mu-\lambda) + \alpha] < W_{vmax} \leq [T(\mu-\lambda) + \beta]$$

| *paramètres* | | | $W_{th}$ | | | $W_{vmax}$ | | | $\overline{RTT}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\lambda$ | $\tau$ | $Min_{th}$ | *analy* | *sim* | *analy* | *sim* | *%err* | *analy* | *sim* | *%err* |
| 150 | 10 | 0.3241 | 15.07 | 16 | 16 | 49.30 | 49 | 0.61 | 0.3500 | 0.3499 | 0.02 |
| 160 | 10 | 0.3237 | 15.51 | 16 | 16 | 52.49 | 52 | 0.94 | 0.3466 | 0.3466 | 0.00 |
| 160 | 50 | 0.1241 | 9.909 | 16 | 16 | 17.33 | 17 | 1.94 | 0.1545 | 0.1545 | 0.00 |
| 160 | 50 | 0.2037 | 12.30 | 16 | 16 | 26.09 | 27 | 3.37 | 0.2454 | 0.2454 | 0.00 |
| 200 | 100 | 0.1240 | 10.50 | 16 | 16 | 15.90 | 15 | 6.00 | 0.1500 | 0.1500 | 0.00 |
| 200 | 100 | 0.3248 | 16.17 | 32 | 32 | 35.90 | 35 | 2.57 | 0.3500 | 0.3500 | 0.00 |
| 350 | 250 | 0.2638 | 17.67 | 32 | 32 | 29.66 | 29 | 2.27 | 0.2900 | 0.2900 | 0.00 |
| 120 | 00 | 0.4050 | 15.08 | 16 | 16 | 52.60 | 52 | 1.15 | 0.4333 | 0.4333 | 0.00 |

TAB. 4.1: Résultats de simulation et résultats analytiques

où $[x]$ est la partie entière de $x$. Par conséquent, l'erreur relative peut être proche de $(1/([T(\mu-\lambda)]+\alpha+1))$. Le $\overline{RTT}$ analytique est le délai moyen d'aller retour calculé pour la fenêtre $W_{vmax}$ obtenue par simulation. $Min_{th}$ est la taille minimale de la fenêtre à la fin de la phase du *slow start* (équation (4.18)).

Dans la table 4.2, on montre le délai d'aller retour moyen par paquet $\overline{rtt}$, le débit moyen de transmission, qui est le nombre de paquets transmis divisé par le temps depuis le commencement. A partir de ces résultats, on peut voir à quel point les résultats analytiques sont proches des résultats de simulation pour le débit moyen (0.80-1.08%). L'erreur relative pour le délai d'aller retour moyen est du même ordre que celle de la taille maximale de la fenêtre pour Vegas, tel que le prévoyait le corollaire 4.3.2 (équation (4.19).

| *paramètres* | | | $\overline{Thp}$ | | | $\overline{rtt}$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\lambda$ | $\tau$ | *analy* | *sim* | *%err* | *analy* | *sim* | *%err* | *Pertes* |
| 150 | 10 | 0.3241 | 130.57 | 131.86 | 0.98 | 0.3503 | 0.3486 | 0.48 | 0 |
| 160 | 10 | 0.3237 | 139.16 | 140.51 | 0.96 | 0.3482 | 0.3454 | 0.75 | 0 |
| 160 | 50 | 0.1237 | 109.16 | 110.22 | 0.96 | 0.1570 | 0.1544 | 1.68 | 0 |
| 160 | 50 | 0.2037 | 107.82 | 108.86 | 0.95 | 0.2365 | 0.2443 | 3.19 | 0 |
| 200 | 100 | 0.1238 | 99.32 | 100.24 | 0.91 | 0.1585 | 0.1501 | 5.59 | 0 |
| 200 | 100 | 0.3238 | 97.16 | 98.09 | 0.94 | 0.3579 | 0.3493 | 2.46 | 0 |
| 350 | 250 | 0.2638 | 98.11 | 98.91 | 0.80 | 0.2963 | 0.2903 | 2.06 | 0 |
| 120 | 00 | 0.4050 | 109.11 | 110.31 | 1.08 | 0.4353 | 0.4312 | 0.95 | 0 |

TAB. 4.2: Débit moyen et *rtt* moyen : Simulation (100 sec) et résultats analytiques sous Vegas

Quelque résultats analytiques et de simulations concernant Reno sont résumés dans la Table 4.3. Le comportement de Reno considéré dans ces simulations, est le comportement idéal (voir la figure 3.1). L'erreur relative pour le débit moyen est, approximativement, la même que pour le délai d'aller retour moyen, et ne dépasse pas les 2.5%. En fait, cette erreur dépend du nombre de pertes ayant lieu durant la simulation, car dans les calculs analytiques, le temps pour recouvrir les pertes n'est pas pris en compte.

| *paramètres* | | | | $\overline{Thp}$ | | | $\overline{rtt}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\lambda$ | $\tau$ | *File* | *analy* | *sim* | *% err* | *analy* | *sim* | *%err* | *Pertes* |
| 150 | 10 | 0.3240 | 30 | 131.95 | 132.78 | 0.61 | 0.3960 | 0.3916 | 1.12 | 6 |
| 160 | 10 | 0.4037 | 25 | 134.90 | 135.54 | 0.47 | 0.4609 | 0.4539 | 1.54 | 5 |
| 160 | 50 | 0.4037 | 25 | 100.62 | 101.41 | 0.77 | 0.4625 | 0.4589 | 0.78 | 6 |
| 160 | 50 | 0.2437 | 35 | 108.39 | 110.12 | 1.57 | 0.3637 | 0.3665 | 0.76 | 9 |
| 178.3 | 50 | 0.4039 | 35 | 119.13 | 120.18 | 0.87 | 0.4833 | 0.4774 | 1.23 | 4 |
| 200 | 100 | 0.4038 | 35 | 92.85 | 94.87 | 2.12 | 0.4723 | 0.4672 | 1.09 | 6 |
| 120 | 00 | 0.4050 | 40 | 115.29 | 116.15 | 0.74 | 0.5728 | 0.5591 | 2.45 | 3 |
| 120 | 00 | 0.4050 | 20 | 109.91 | 110.46 | 0.49 | 0.4707 | 0.4643 | 1.35 | 5 |

TAB. 4.3: Débit moyen et $rtt$ moyen : Simulation (100 sec) et résultats analytiques sous Reno

### 4.3.5 Comportement de Vegas

**Définition 4.3.1** *On dit qu'un algorithme de contrôle de congestion (basé sur le mécanisme de fenêtrage) "a le même comportement que Reno", si il a un comportement cyclique, dans lequel sa fenêtre continue de croître jusqu'à ce qu'au moins une perte ait lieu.*

**Théorème 4.3.2** *Pour* $B \in \left[0, \dfrac{\alpha\mu}{\mu-\lambda}\right]$, $Vegas_{\alpha,\beta}$ *"a le même comportement que Reno" et ne peut estimer la bande passante disponible dans ces cas.*

**Preuve:** On a $W_{vmax} > T(\mu-\lambda)+\alpha$, par conséquent si $T(\mu-\lambda)+\alpha \geq W^v_{max}$ alors il y aura des pertes avant que Vegas n'atteigne le seuil inférieur de la fenêtre. Ainsi, le Théorème est obtenu en remplaçant $W^v_{max}$ par sa valeur donnée dans (4.3). Une autre preuve équivalente peut être formulée comme suit :
La taille de la fenêtre de congestion dépend fortement des variations de $\overline{RTT_n}$, le "temps de séjour" moyen sur une fenêtre de données. Alors, quand cette moyenne ne peut pas excéder suffisamment $T$, à cause d'une bande passante assez petite ou d'une file de capacité également petite, la fenêtre continue de croître jusqu'à ce qu'il ait des pertes. Dans la phase de *congestion*, il y a perte si la fenêtre atteint ou dépasse $W^v_{max}+1$, et Vegas incrémente sa fenêtre dans le prochain $\delta_i$ si $T(Expected - Actual) \leq \alpha$. Puisque le "temps de séjour" moyen sur une fenêtre augmente tant que la taille de la fenêtre augmente alors, il y aura perte avant que la fenêtre atteigne le seuil inférieur ($T(\mu-\lambda)+\alpha$ ou se stabilise) si

$$\frac{W^v_{max}}{T} - \frac{W^v_{max}}{\overline{RTT}} \leq \frac{\alpha}{T} \quad \text{équivalent à} \quad \epsilon_{W^v_{max}} \leq \frac{\alpha T}{W^v_{max}-\alpha} \tag{4.20}$$

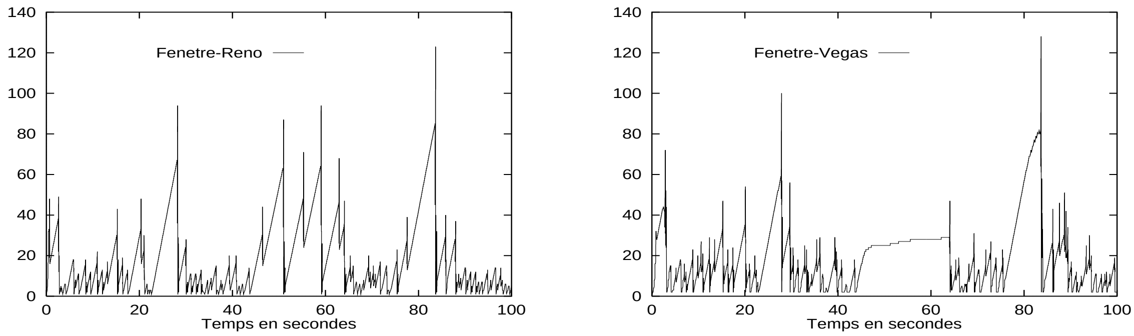
En substituant (4.3) dans (4.20), et puisque $\epsilon_{W^v_{max}} = B/\mu$, on déduit que (4.20) est équivalente à

$$\frac{B}{\mu} \leq \frac{\alpha T}{\left(\frac{B}{\mu}+T\right)(\mu-\lambda)-\alpha} \quad \Longleftrightarrow \quad B^2 + \left(T\mu - \frac{\alpha\mu}{\mu-\lambda}\right)B - \frac{\alpha T\mu^2}{\mu-\lambda} \leq 0 \tag{4.21}$$

En résolvant cette équation, on obtient la condition nécessaire et suffisante pour que (4.20) soit vérifiée : $B \in \left[0, \dfrac{\alpha\mu}{\mu-\lambda}\right]$, avec $W_{max} > \alpha$.

▮

**Remarque 4.3.2** *Durant les périodes de fortes congestions, on a* $\lim_{\lambda \to \mu} (\alpha\mu)/(\mu - \lambda) = +\infty$. *Par conséquent, il découle du Théorème 4.3.2 que quelle que soit la capacité maximale de la file, Vegas a le même comportement que Reno si* $\lambda$ *est suffisamment proche de* $\mu$.




(a) Reno (b) Vegas

FIG. 4.2: Fenêtre de congestion pour Vegas et Reno, pour $\tau = 81.2\ ms$ et la capacité maximale de la file de 20 paquets en présence d'un trafic vidéo.

La Figure 4.2 illustre un exemple où Vegas se comporte de la même façon que Reno, pour une capacité de la file de 20 paquets. On a utilisé le simulateur de réseaux REAL [68] dans lequel nous avons introduit le protocole Vegas à partir de [27]. Le modèle simulé est décrit précédemment avec des traces vidéo de *Star war* comme trafic exogène. La source vidéo est supposée transmettre une trame toutes les 30 $ms$, en divisant ses trames en paquets de taille fixe de 576 octets. Durant les 100 secondes de simulation, le débit moyen de la source exogène est de 934.136 $pkts/sec$. La capacité du goulot (routeur) est supposée être de 1500 $pkts/sec$. On peut constater que le comportement de la fenêtre est relativement le même pour Vegas et Reno, sauf durant la période 40 à 60 sec (Figure 4.2 (b), entre les instants 40-60 *sec*), où la fenêtre de Vegas tend à se stabiliser. Ceci illustre que les performances de Vegas sont meilleurs quand le débit est assez grand. Notons que la capacité maximale de la file considérée dans cet exemple est plus grande que celle calculée à partir du Théorème 4.3.2 ($\sim 6$ paquets).

Pour le cas de plusieurs connexions ($N$ connexions), supposons que dès que la file d'attente dépasse sa capacité maximale, toutes les fenêtres de Vegas sont réduites (pertes synchrones, voir section suivante). Le débit total des connexions Vegas, dans ce cas, quand toutes les fenêtres se stabilisent reste toujours égale à $\mu - \lambda$. De plus, suivant le même raisonnement que précédemment, Vegas dans ce cas *a le même comportement que Reno* si la fenêtre maximale $W_{vmax}^{v,i}$ d'une connexion $i$ satisfait l'équation (4.20), i.e ($T_i$ est le délai d'aller retour pour la connexion $i$)

$$\begin{aligned}
& \frac{W_{max}^{v,i}}{T_i} - \frac{W_{max}^{v,i}}{RTT_i} \leq \frac{\alpha}{T_i} \Longleftrightarrow \frac{W_{max}^{v,i}}{RTT_i}\frac{B}{\mu} \leq \alpha \\
\Longrightarrow \quad & \frac{B}{\mu}\sum_{i=1}^{N}\frac{W_{max}^{v,i}}{RTT_i} \leq N\alpha \Longleftrightarrow \frac{B}{\mu}(\mu - \lambda) \leq N\alpha \\
\Longleftrightarrow \quad & B \leq \frac{N\alpha\mu}{\mu - \lambda} \qquad (4.22)
\end{aligned}$$

Par conséquent, le Théorème précédent se généralise, dans le cas de pertes synchrones et de plusieurs ($N$) connexions comme suit :

Dans le cas de $N$ connexions partageant le même goulot d'étranglement avec pertes synchrones, si $B \leq \frac{N\alpha\mu}{\mu - \lambda}$ alors $Vegas_{\alpha,\beta}$ "a le même comportement que Reno", et ne peut estimer la bande passante disponible dans ce cas. C'est la raison pour laquelle plus le nombre de connexions augmente, plus les performances de Vegas se détériorent [57].

## 4.4 Le cas de plusieurs connexions non équivalentes

### 4.4.1 Le modèle

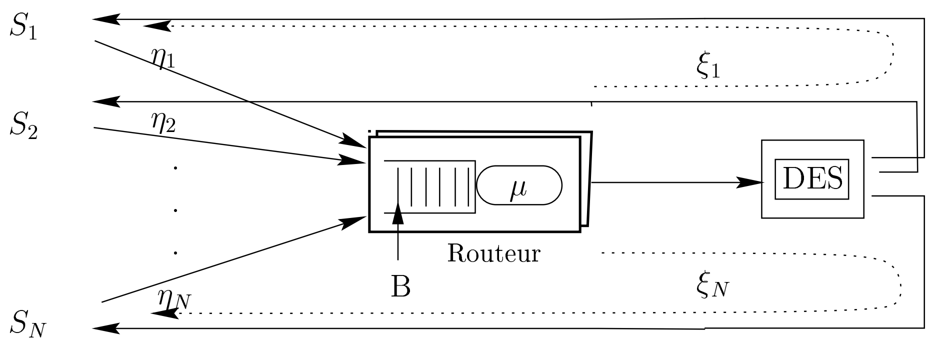



FIG. 4.3: Modèle à plusieurs sources partageant le même goulot d'étranglement

Considérons le modèle où plusieurs ($N$) connexions transmettent vers la même destination et partagent le même goulot d'étranglement, dont la capacité de transmission est de $\mu$ et la capacité de stockage est de $B$. Un paquet envoyé par une source $i$ prend $\eta_i$ (sec) pour arriver au routeur. $\xi_i$ est le temps qui sépare l'instant où un paquet quitte le routeur jusqu'à l'instant où son acquittement est reçu par la source (voir figure 4.3). On définit ainsi le délai d'aller-retour comme $\tau_i \triangleq \eta_i + \xi_i + \mu^{-1}$. C'est le temps qui sépare l'instant d'envoi d'un paquet et l'instant de réception de son acquittement, dans un système vide (voir figure 4.3). Nous appelons le "délai de séjour dans le réseau" ($R_i(t)$ pour la connexion $i$) le délai que prend un paquet depuis son envoi (à l'instant $t$) jusqu'à réception de son acquittement (à l'instant $t + \tau_i + Q(t + \eta_i)/\mu$ i.e $R_i(t) = \tau_i + Q(t + \eta_i)/\mu$ ). Soit $Q(t)$ la taille de la file d'attente au niveau du routeur à l'instant $t$, et $Q_0$ la taille de la file à la fin d'un cycle.

Soit $\mu_i(t)$ le débit alloué à la connexion $i$ quand la file n'est pas vide ($\sum_{i=1}^{N} \mu_i(t) = \mu$), c'est également l'intensité de service des paquets de la connexion $i$ à l'instant $t$. Soit également $S_i(t)$ le débit de transmission de la source $i$ à l'instant $t$.

### 4.4.2 Hypothèses

Nous considérons que quand une perte a lieu toutes les sources recevront une indication de perte, et par conséquent réduisent leur fenêtre d'un facteur de $\delta < 1$. Cette hypothèse peut être vue comme le fonctionnement de TCP avec des indications de congestion (ECN-TCP [42, 29] : Explicit Congestion Notification). Ces indications sont envoyées pour chaque source, une seule fois pendant un certain nombre de RTT dès que la taille de la file d'attente dépasse la capacité maximale $B$. Pendant l'intervalle de temps qui suit, avant que le débit total de toutes les sources soit réduit, on suppose que le débit d'entrée global est proche de $\mu$. Ainsi, la file d'attente commence à se vider une fois qu'une source parmi les $N$ sources réduit sa fenêtre. Par conséquent, il est tout à fait raisonnable de supposer qu'il n'y aura pas plus d'une perte par cycle et par source (c'est à dire que si chaque source perd un paquet, cela assure que la taille de la file d'attente reste inférieur à sa capacité maximale, pendant l'intervalle de temps qui sépare l'instant où une perte se produit jusqu'au moment où l'une des sources réduit la taille de sa fenêtre).

Nous nous intéresserons uniquement au comportement périodique de Reno et Vegas, ainsi seule *la phase de congestion* sera prise en compte. Ceci correspond au fonctionnement idéal des deux algorithmes. Nous considérons également que le délai d'aller-retour minimal calculé par Vegas, est effectivement $\tau_i$, par conséquent $base\_RTT_i = \tau_i$.

L'évolution de la fenêtre $w_i(t)$ en fonction du temps dans Reno et dans Vegas pour une connexion $i$ peut se résumer comme suit [7, 16, 23] :

$$\textbf{TCP Reno} \qquad \begin{cases} \dfrac{dw_i(t)}{dt} = \dfrac{1}{R_i(t)} \\ \\ Q(t) = B \Longrightarrow w_i(t^+ + \xi_i) = \delta w_i(t + \xi_i). \end{cases} \tag{4.23}$$

Ceci est justifié par le fait que dans la phase de congestion, Reno augmente la taille de sa fenêtre de $1/w_i(t)$ à chaque réception d'un acquittement, par conséquent $\Delta w_i(t) = 1/w_i(t)$. Comme les acquittements sont reçus à un débit de $\mu_i(t)$, alors $\frac{dw_i(t)}{dt} \approx \frac{\Delta w_i(t)}{\Delta t} = \frac{\mu_i(t)}{w_i(t)} = \frac{1}{R_i(t)}$. La dernière équation traduit le fait que la taille de la fenêtre est multiplié par un facteur $\delta$ (typiquement 1/2), à la détection d'une perte. Le même raisonnement pour Vegas nous donne :

$$\textbf{TCP Vegas} \qquad \begin{cases} \dfrac{dw_i(t)}{dt} = \dfrac{\epsilon_i(t)}{R_i(t)} \\ \\ Q(t) = B \Longrightarrow w_i(t^+ + \xi_i) = \delta w_i(t + \xi_i), \end{cases} \tag{4.24}$$

où

$$\epsilon(t) = \begin{cases} 1 & \text{si} \quad \Delta_i(t) < \alpha \\ 0 & \text{si} \quad \alpha \le \Delta_i(t) \le \beta \\ -1 & \text{si} \quad \Delta_i(t) > \beta, \end{cases}$$

avec

$$\Delta_i(t) = w_i(t)\left(1 - \frac{\tau_i}{R_i(t)}\right).$$

Le débit de service des paquets de la source $i$ est alors donné par :

$$\mu_i(t) = \begin{cases} \dfrac{w_i(t-\eta_i)}{\tau_i} & \text{si} \quad \displaystyle\sum_{k=1}^{N} \frac{w_i(t-\eta_k)}{\tau_k} \le \mu \\ \dfrac{S_i(t-\eta_i)}{\sum_{k=1}^{N} S_i(t-\eta_k)}\mu & \text{ailleurs.} \end{cases} \tag{4.25}$$

La dernière équation est justifiée par le fait que le débit de sortie pour une connexion est proportionnel à son débit d'entrée. $S_i(t)$, le débit de transmission de la source $i$ satisfait l'équation suivante :

$$S_i(t+\xi_i) = \mu_i(t) + \frac{dw_i(t+\xi_i)}{dt}. \tag{4.26}$$

Ceci s'explique par le fait que, pendant un RTT, TCP envoie un paquet pour chaque acquittement reçu (ce qui correspond au débit de service des paquets de la source : $\mu_i(t)$), plus ce que lui permet la variation de la fenêtre pendant ce temps. Dans le cas de Reno, la fenêtre augmente d'un paquet tous les RTT, par conséquent

$$S_i(t+\xi_i) = \mu_i(t) + \frac{dw_i(t+\xi_i)}{dt} = \mu_i(t) + \frac{\mu}{\tau_i\mu + Q(t+\tau_i)} = \mu_i(t) + \frac{w_i(t+\xi_i)}{\mu_i(t)}. \tag{4.27}$$

Pour Vegas, la variation de la fenêtre dépend de $\Delta_i(t)$ défini ci-dessus, par conséquent

$$S_i(t+\xi_i) = \mu_i(t) + \frac{dw_i(t+\xi_i)}{dt} = \mu_i(t) + \frac{\mu\epsilon_i(t+\xi_i)}{\tau_i\mu + Q(t+\tau_i)} = \mu_i(t) + \epsilon_i(t+\xi_i)\frac{w_i(t+\xi_i)}{\mu_i(t)}.$$

### 4.4.3 Réduire l'influence du nombre de connexions sur Vegas

Comme mentionné précédemment le nombre de connexions a une influence sur les performances de Vegas. Pour le scénario décrit dans la figure 4.3, le nombre de paquets dans la file d'attente est au moins égal à $N\alpha$ [23]. $N$ est le nombre de connexions partageant le même lien.

En effet, d'après l'équation (4.24), la fenêtre $w_i(t)$ de la source $i$, ayant un temps de séjour $R_i(t)$ et un délai d'aller retour $\tau_i$, se stabilise ($w_i(t) = w_i \ \forall t > t_0$), quand elle vérifie l'équation suivante :

$$\frac{\alpha}{\tau_i} < w_i(t)\left(\frac{1}{\tau_i} - \frac{1}{R_i(t)}\right) \le \frac{\beta}{\tau_i} \tag{4.28}$$

Si toutes les fenêtres se stabilisent, alors la file d'attente également. Car le débit d'entrée est égal au débit de sortie. Par conséquent, le délai d'aller retour de la source $i$ est $R_i(t) = R_i \triangleq \tau_i + \bar{Q}/\mu$, où $\bar{Q}$ est la taille de la file d'attente. Soit $\mu_i$ la bande passante attribuée à la source $i$, on a $\mu_i = w_i/R_i$ ; l'équation (4.28) est alors équivalente à

$$\frac{\alpha}{\tau_i} < \mu_i\left(\frac{\tau_i + \frac{\bar{Q}}{\mu}}{\tau_i} - 1\right) \le \frac{\beta}{\tau_i} \Longleftrightarrow \alpha < \frac{\mu_i}{\mu}\bar{Q} \le \beta. \tag{4.29}$$

Cette relation est vérifiée même si les délais d'aller retour des sources n'est pas le même. En sommant sur le nombre de connexions $N$, on obtient finalement :

$$N\alpha < \sum_{i=1}^{N}\frac{\mu_i}{\mu}\bar{Q} \le N\beta \Longleftrightarrow N\alpha < \bar{Q} \le N\beta.$$

#### 4.4.3.1 Améliorations

Dans la section 4.3.2, nous avons proposé d'augmenter la fenêtre d'un facteur de 1.5 tous les RTTs, au lieu de la doubler tous les RTTs (envoyer deux paquets pour un premier acquittement, puis un seul pour le suivant). Cela peut en effet diminuer les pertes dans la phase du *slow start*. Cependant, comme on vient de le voir en présence de plusieurs connexions la taille de la file d'attente au niveau du routeur devient très large. Dans ce qui suit, nous proposons des améliorations qui permettent d'avoir une taille de la file d'attente moins importante. Cette proposition concerne la façon de détecter la bande passante. Elle consiste à augmenter ou à réduire la fenêtre en comparant la différence entre le débit actuel et le débit attendu, non pas aux seuils $\frac{\alpha}{\tau_i}$ et $\frac{\beta}{\tau_i}$, mais aux seuils suivants : $\frac{\alpha}{R_i}$ et $\frac{\beta}{R_i}$.

Ainsi, dans la phase du *slow start* on augmente la fenêtre d'un facteur de 1.5 tous les RTTs tant que la relation suivante est vérifiée :

$$w_i(t)\left(\frac{1}{\tau_i}-\frac{1}{R_i(t)}\right)\leq\frac{\gamma}{R_i(t)}.$$

Dans la phase de congestion, la fenêtre est augmentée d'un paquet tous les RTTs tant que

$$w_i(t)\left(\frac{1}{\tau_i}-\frac{1}{R_i(t)}\right)\leq\frac{\alpha}{R_i(t)},$$

et enfin, elle est réduite d'un paquet tous les RTTs tant qu'on a

$$w_i(t)\left(\frac{1}{\tau_i}-\frac{1}{R_i(t)}\right)>\frac{\beta}{R_i(t)}.$$

On a pu vérifier par de nombreuses simulations, en changeant uniquement la façon de détecter la bande passante, que la taille de la file d'attente est sensiblement moins importante que dans le cas précédent, bien que le débit soit le même. En considérant le modèle décrit plus haut, on peut montrer que la taille de la file d'attente avec ces changements est plus petite que $N\alpha$. En effet, quand le système se stabilise, de l'équation (4.29) on déduit

$$\frac{\alpha}{R_i}<w_i\left(\frac{1}{\tau_i}-\frac{1}{R_i}\right)\leq\frac{\beta}{R_i}\iff\frac{\alpha\mu}{\tau_i\mu+\bar{Q}}<\mu_i\frac{\bar{Q}}{\tau_i\mu}\leq\frac{\beta\mu}{\tau_i\mu+\bar{Q}}\tag{4.30}$$

En résolvant cette équation, on obtient

$$N\alpha>\frac{-1+\sqrt{1+4N\alpha\frac{1}{\mu^2}\sum_{i=1}^N\frac{\mu_i}{\tau_i}}}{2\frac{1}{\mu^2}\sum_{i=1}^N\frac{\mu_i}{\tau_i}}<\bar{Q}\leq\frac{-1+\sqrt{1+4N\beta\frac{1}{\mu^2}\sum_{i=1}^N\frac{\mu_i}{\tau_i}}}{2\frac{1}{\mu^2}\sum_{i=1}^N\frac{\mu_i}{\tau_i}}<N\beta\tag{4.31}$$

Soit

$$f_k(N)\triangleq\frac{-1+\sqrt{1+4Nk\frac{1}{\mu^2}\sum_{i=1}^N\frac{\mu_i}{\tau_i}}}{2\frac{1}{\mu^2}\sum_{i=1}^N\frac{\mu_i}{\tau_i}}$$

La figure 4.4 illustre la taille maximale ($\beta N$ *resp.* $f_\beta(N)$) et minimale ($\alpha N$ *resp.* $f_\alpha(N)$) de la file d'attente dans Vegas, respectivement avant et après l'introduction des améliorations pour la détection de la bande passante, pour $k=\alpha=1$, $k=\beta=3$ et $\left(\frac{1}{\mu^2}\sum_{i=1}^N\frac{\mu_i}{\tau_i}\right)^{-1}=256\leq\tau_{max}\mu$ (correspondant approximativement à la taille maximale de la fenêtre (en paquets) qu'une connexion pourrait avoir). On peut facilement remarquer que la proposition que nous avons faite, permet d'avoir une taille de la file d'attente qui varie lentement en fonction du nombre de connexions, contrairement au cas de Vegas où elle est linéaire, spécialement pour la taille maximale $f_\beta(N)$. Dans cet exemple, pour 100 connexions la taille minimale (*resp.* maximale) de la file d'attente dans Vegas est environ quatre tiers (*resp.* deux fois) la taille obtenue après l'amélioration.

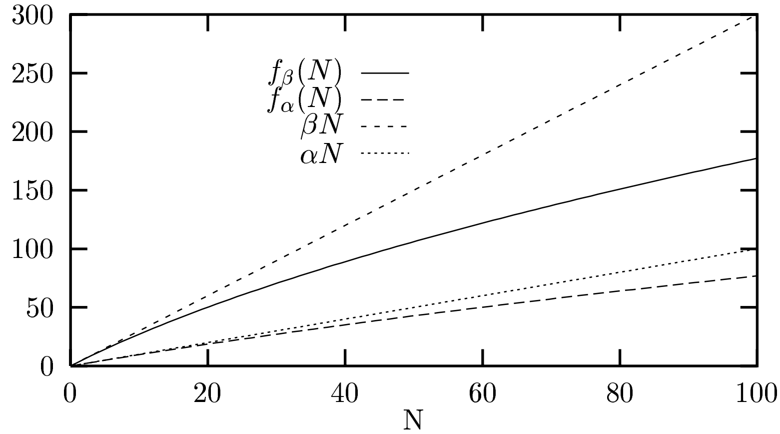



FIG. 4.4: Taille minimale et maximale de la file d'attente en fonction du nombre de connexions, avant et après l'amélioration

La figure 4.5 illustre la différence entre la taille maximale et la taille minimale de la file d'attente en fonction du nombre de connexions et de $\left(\frac{1}{\mu^2}\sum_{i=1}^{N}\frac{\mu_i}{\tau_i}\right)^{-1}$. On peut remarquer en particulier que quand $\left(\frac{1}{\mu^2}\sum_{i=1}^{N}\frac{\mu_i}{\tau_i}\right)^{-1}$ n'est pas très grand, cette différence dans la version améliorée de Vegas est nettement moins importante que dans le cas de Vegas. Même si la quantité $\left(\frac{1}{\mu^2}\sum_{i=1}^{N}\frac{\mu_i}{\tau_i}\right)^{-1}$ est assez grande, dans le cas d'un nombre important de connexions, la différence entre la taille maximale et la taille minimale reste toujours plus importante dans le cas de Vegas qu'après l'introduction des améliorations pour la détection de la bande passante. Par conséquent, la version de Vegas améliorée permet d'avoir moins de paquets dans la file, avec une différence entre la taille maximale et la taille minimale de la file d'attente nettement réduite.

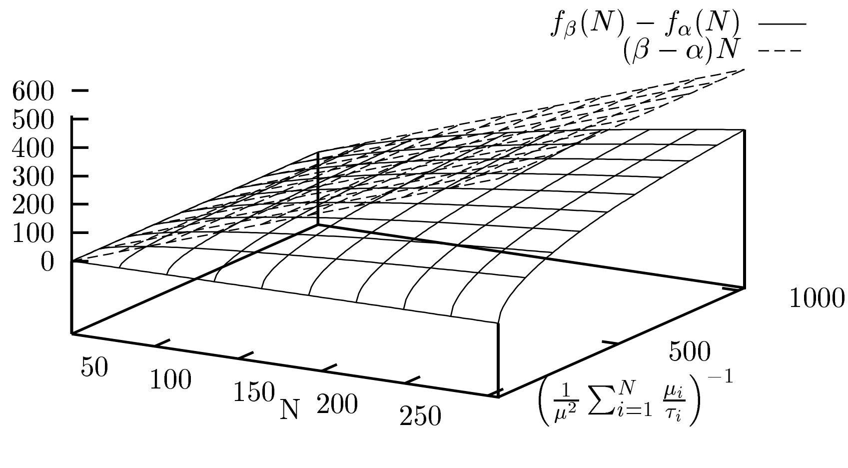



FIG. 4.5: Différence entre la taille maximale et la taille minimale de la file d'attente dans Vegas en fonction du nombre de connexions, avant et après l'amélioration

### 4.4.4 Équité dans TCP-Vegas et TCP-Reno

#### 4.4.4.1 Vegas

Dans Vegas il est facile de constater d'après l'équation (4.29) [23], que le rapport entre le débit attribué à une source $i$ $(\mu - i)$ et le débit attribué à une autre source $j$ $(\mu_j)$, quel que soit les délais des sources, est borné comme suit :

$$\frac{\alpha}{\beta} < \frac{\mu_i}{\mu_j} \leq \frac{\beta}{\alpha} \quad \forall\, i, j.$$

Ainsi, plus $\alpha$ est proche de $\beta$ plus Vegas est équitable. Cependant, les oscillations de la taille de la fenêtre autour de la taille optimale dans ce cas seront plus importantes et pendant une longue période (convergence lente).

#### 4.4.4.2 Reno

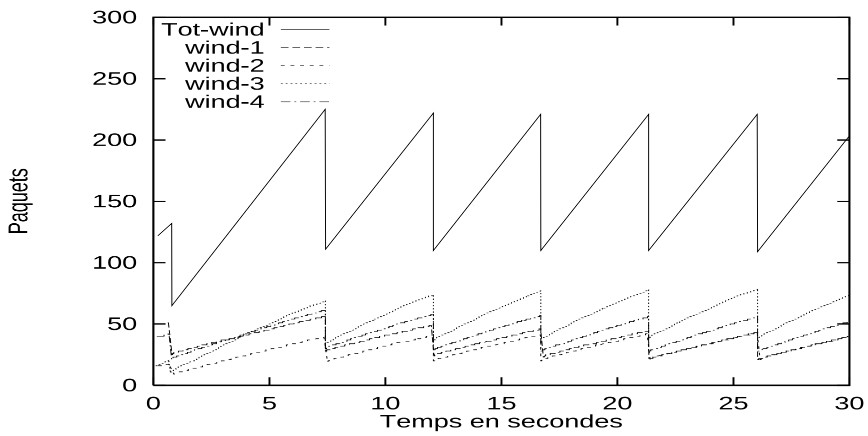



FIG. 4.6: Évolution de la fenêtre pour quatre connexions, dont deux connexions équidistantes

Dans le cas de Reno, nous supposons que le régime périodique existe. La figure 4.6 illustre un exemple où les fenêtre tendent à avoir le même comportement dans chaque cycle. Dans cette figure $Tot - wind$ représente la somme de toutes les fenêtres. On constate que cette dernière atteint son comportement périodique juste après la première perte. Soit $t_0$ l'instant de détection d'une perte $(w_i(t_0 + \xi_i) = \delta w^i_{max})$, et $T$ la durée d'un cycle. La dynamique des fenêtres nous permet d'écrire, à partir de l'équation (4.23) et (4.26) :

$$
\begin{aligned}
& \frac{dw_i(t+\xi_i)}{dt} = \frac{1}{R_i(t+\xi_i)} = \frac{\mu_i(t)}{w_i(t+\xi_i)} \\
\iff\quad & w_i(t+\xi_i)\frac{dw_i(t+\xi_i)}{dt} = \mu_i(t) \\
\iff\quad & \int_{t_0}^{t_0+T} w_i(s+\xi_i)\frac{dw_i(s+\xi_i)}{ds}ds = \int_{t_0}^{t_0+T} \mu_i(s)ds \\
\iff\quad & \left[\frac{w_i(s+\xi_i)^2}{2}\right]_{t_0}^{t_0+T} = T\bar{\mu}_i \\
\Longrightarrow\quad & \frac{1}{2}\left({w_{max}^i}^2 - \delta^2\left(w_{max}^i\right)^2\right) = T\bar{\mu}_i \\
\iff\quad & \left(w_{max}^i\right)^2 - \frac{2T}{1-\delta^2}\bar{\mu}_i = 0 \\
\Longrightarrow\quad & \frac{\bar{\mu}_i}{\bar{\mu}_j} = \left(\frac{w_{max}^i}{w_{max}^j}\right)^2, \qquad (4.32)
\end{aligned}
$$

où $\bar{\mu}_i$ est la moyenne du débit de service des paquets de la source $i$ durant un cycle.

Posons $D_i \triangleq \frac{w_{max}^i}{\bar{\mu}_i}$ $(\tau_i \leq D_i \leq \tau_i + B/\mu)$, alors

$$
\begin{aligned}
& D_i^2 = \frac{2T}{(1-\delta^2)\bar{\mu}_i} \\
\iff\quad & \bar{\mu}_i = \frac{2T}{(1-\delta^2)D_i^2} \Longrightarrow \frac{\bar{\mu}_i}{\bar{\mu}_j} = \frac{D_j^2}{D_i^2} \qquad (4.33)
\end{aligned}
$$

Ainsi, le débit d'une source est proportionnel au carré de son "délai d'aller retour". Contrairement à Vegas, dans Reno le rapport entre le débit de deux sources n'est borné que par le rapport entre leurs délais respectifs. Par conséquent, Vegas est beaucoup plus équitable que Reno, essentiellement quand le rapport des délai est assez large. Cependant pour des sources relativement équidistantes, Reno semble être plus équitable que Vegas, ceci dépend des fenêtres initiales dans Vegas. Si une connexion vient de s'établir, pendant qu'une autre est à son état stationnaire dans Vegas, et si la bande passante dans ce cas est suffisamment large pour qu'il n'y ait pas de pertes, alors la deuxième source même si elle est équidistante à la première n'aura pas le même débit que la première.

#### 4.4.4.3 Exemples numérique

Dans cette section on suppose uniquement le cas où la bande passante est totalement utilisée. Dans ce cas, on a $\sum_{1=1}^{N}\bar{\mu}_i = \mu$. Par conséquent,

$$
T = \frac{(1-\delta^2)\mu}{2\sum_{k=1}^{N}\left(\frac{1}{D_k}\right)^2} \Longrightarrow \bar{\mu}_i = \frac{\mu}{D_i^2\sum_{k=1}^{N}\left(\frac{1}{D_k}\right)^2}. \qquad (4.34)
$$

D'après l'équation (4.25) et (4.27), on peut supposer que $\bar{\mu_i} \approx \mu_i(t) \approx \mu_i$. Par conséquent $D_i \approx \frac{w^i_{max}}{\mu_i} = \tau_i + \frac{B}{\mu}$. Ainsi, de l'équation (4.34), on a

$$\bar{\mu_i} = \frac{\mu}{\left(\tau_i + \frac{B}{\mu}\right)^2 \sum_{k=1}^{N} \left(\frac{1}{\tau_k + \frac{B}{\mu}}\right)^2}$$
$$w^i_{max} = \bar{\mu_i}\left(\tau_i + \frac{B}{\mu}\right) \tag{4.35}$$

**Exemples :** Dans la table 4.4, sont résumés les résultats analytiques obtenus de l'équation (4.35) et ceux obtenus par simulation, pour la taille maximale de la fenêtre de chaque connexion et le débit qui lui est associé. Le scénario utilisé est celui décrit dans le modèle ci-dessus (figure 4.3). Le protocole simulé est bel et bien TCP Reno, avec ses mécanismes de *fast retransmit* et de *fast recovery.* Nous avons considéré un routeur de telle sorte que le dépassement de la capacité maximale du routeur provoque une seule perte pour chaque source.

| | *paramètres* | | | $w^i_{max}$ | | | $\bar{\mu_i}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| *Source* | $\mu$ | $B$ | $\tau_i$ | *analy* | *sim* | *% err* | *analy* | *sim* | *%err* |
| S-1 | 200.0 | 100 | 0.2 | 114.96 | 119.00 | 3.39 | 164.23 | 171.57 | 4.27 |
| S-2 | 200.0 | 100 | 1.0 | 53.64 | 46.00 | 16.6 | 35.76 | 25.47 | 40.4 |
| S-1 | 1000.0 | 150 | 0.1 | 92.08 | 96.00 | 4.14 | 368.34 | 407.94 | 9.70 |
| S-2 | 1000.0 | 150 | 0.15 | 76.73 | 78.00 | 1.62 | 255.79 | 245.16 | 4.33 |
| S-3 | 1000.0 | 150 | 0.2 | 65.77 | 62.00 | 6.08 | 187.93 | 164.59 | 14.1 |
| S-4 | 1000.0 | 150 | 0.2 | 65.77 | 62.00 | 6.08 | 187.93 | 164.72 | 14.09 |
| S-1 | 400.0 | 100 | 0.1 | 69.79 | 75.00 | 6.94 | 199.403 | 225.75 | 11.6 |
| S-2 | 400.0 | 100 | 0.2 | 54.28 | 53.00 | 2.41 | 120.62 | 111.104 | 8.55 |
| S-3 | 400.0 | 100 | 0.4 | 37.57 | 33.00 | 13.8 | 57.81 | 44.96 | 28.5 |
| S-4 | 400.0 | 100 | 0.8 | 23.26 | 19.00 | 22.42 | 22.15 | 15.33 | 44.4 |
| S-1 | 1000.0 | 70 | 0.1 | 112.53 | 116.00 | 2.99 | 661.97 | 673.45 | 1.70 |
| S-2 | 1000.0 | 70 | 0.2 | 70.85 | 65.00 | 9.00 | 262.43 | 206.61 | 27.0 |
| S-3 | 1000.0 | 70 | 0.5 | 33.56 | 27.00 | 24.2 | 58.88 | 39.75 | 48.1 |
| S-4 | 1000.0 | 70 | 1.0 | 17.87 | 15.00 | 19.1 | 16.70 | 10.73 | 55.6 |
| S-1 | 500.0 | 150 | 0.1 | 115.29 | 120.00 | 3.84 | 288.23 | 314.06 | 8.22 |
| S-2 | 500.0 | 150 | 0.2 | 92.23 | 87.00 | 6.01 | 184.47 | 165.20 | 11.6 |
| S-3 | 500.0 | 150 | 1.0 | 17.44 | 29.00 | 39.8 | 27.28 | 17.97 | 51.8 |

TAB. 4.4: Débit moyen et taille maximale de chaque fenêtre : Simulations et résultats analytiques

On remarque que quand les délais sont suffisamment proches (exemple 2) l'approximation (4.35) est tout à fait raisonnable, essentiellement pour la taille maximale de chaque fenêtre. Cependant, dès que les délais tendent à être très différents (exemple 3 et 4), l'erreur relative ($|sim - analy|/sim$) tend aussi à être assez grande, bien qu'elle ne dépasse pas dans la majorité des cas les 30 % pour la taille maximale de la fenêtre.

Ces erreurs sont dues au choix des $D_i$. Le choix précédent suppose implicitement que le nombre de paquets empilés dans la file pour chaque source est proportionnel au débit de service (moyen) des paquets de la source en question. Ce qui n'est apparement pas le cas, cependant les résultats obtenus sont de bonnes approximations. Leurs améliorations fait l'objet d'un travail en cours.

## 4.5 Conclusion

Dans ce chapitre, nous avons étudié le comportement de Vegas et de Reno et analysé leur mécanisme de *slow start*. Nous avons considéré un modèle assez simple. Celui ci nous a permis d'exhiber les caractéristiques de chaque algorithme, de mieux comprendre la réaction de Vegas au nombre de connexions, les insuffisances de son *slow start*, et de le comparer à Reno.

Nous avons ainsi vu que Vegas a de très bonnes performances quand les capacités des files d'attente au niveau des routeurs sont assez larges. Il permet une utilisation maximale, ainsi qu'une meilleure équité que Reno. Cependant, quand la bande passante est assez grande avec des files d'attente de petites capacités, le *slow start* dans Vegas peut engendrer des pertes tout comme Reno. Nous avons aussi montré que dans le cas où la bande passante disponible n'est pas suffisamment large, Vegas a le même comportement que Reno.

Nous avons proposé une autre façon d'accroître la fenêtre durant le *slow start* (augmenter la fenêtre d'un facteur de 1.5). Nous avons également proposé des améliorations pour la détection de la bande passante disponible. Ces améliorations permettent d'avoir moins de paquets dans les routeurs intermédiaires dans le cas de plusieurs connexions de TCP Vegas. Toutefois, de plus amples analyses sont nécessaires pour statuer sur les performances de ces propositions en considérant différentes situations. Ceci fait l'objet d'un travail en cours.

Nous appuyant sur de nombreuses simulations, nous avons constaté que Vegas interagit nettement mieux que Reno avec le service ABR (au dessus de ATM en général [3, 25, 35, 40, 81, 85, 94, 95, 96]). Si les seuils $\alpha$ et $\beta$ sont choisis en fonction de la capacité des files d'attente dans les commutateurs intermédiaires et/ou du nombre de connexions, alors TCP fonctionnerait en parfaite harmonie au dessus de ce service. Ceci fera l'objet de futures investigations.

# Deuxième partie

# ABR dans les réseaux ATM : Available Bit Rate

# Introduction

Les mécanismes adaptatifs pour le contrôle de congestion ont un rôle central dans le partage efficace des ressources du réseau entre différents utilisateurs. Ces mécanismes ont également le rôle de prévention de congestion dans le réseau [52]. Cependant, le fait que le contrôle soit assuré par les sources (qui ne sont pas gérées par le réseau), et non pas par le réseau, ne permet pas de protéger ce dernier des applications n'utilisant pas de mécanismes de contrôle (ex. des vidéos conférence utilisant UDP).

Dans les réseaux ATM, le service ABR (Available Bit Rate) [1] a été défini pour supporter les applications "Best Effort". Les décisions de contrôle sont prises par le réseau au niveau des commutateurs, visant ainsi à assurer l'équité entre les connexions actives et à contrôler le taux de pertes. Ce service tente de récupérer la bande passante non utilisée par les applications qui ont des qualités de service garanties (VBR et CBR), et la partager entre les sources ABR en leur signalant le débit de transmission qui leur est alloué. Le comportement de la source et de la destination est spécifié dans [1] ; est spécifiée également, la façon dont les informations (de congestion et autres) sont transmises à la source [1, 60]. Par contre, le comportement du commutateur (le calcul du débit alloué ... etc) est laissé aux constructeurs.

Plusieurs algorithmes ont été proposés pour les commutateurs. Ils sont soit basés sur (i) l'indication de congestion par un bit (EFCI : Explicit Forward Congestion Indication), qui indique si la congestion est détectée ou non, ou sur (ii) ER (Explicit Rate) qui informe la source du débit qui lui est alloué (débit disponible) (voir le chapitre suivant et [30, 32, 62] également les références qui y figurent).

Le chapitre 5 dans cette partie est consacré à la description du service ABR, tel qu'il est défini par l'ATM forum [1]. Une brève description des autres services dans les réseaux ATM est également faite (CBR, VBR, ABT, GFR ...). Les algorithmes de contrôle de flux pour l'ABR analysés tout au long de cette partie, sont décrits au fur et à mesure que nous les évaluons.

Dans le chapitre 6, essentiellement les algorithmes basés sur l'indication de congestion sont analysés. Un modèle à plusieurs commutateurs en tandem est étudié. Cette étude est une généralisation d'une précédente étude où seuls deux commutateurs en tandem étaient considérés. Nous déterminons plusieurs paramètres de performances, tels que la taille maximale de la file d'attente au niveau de chaque commutateur, le commutateur responsable de la congestion et d'autres. Une comparaison au modèle consistant à simplifier le réseau en un seul commutateur est également faite.

Des exemples d'algorithmes basés sur les tailles des files d'attente aux commutateurs intermédiaires, ainsi que ceux basés sur le débit explicite sont présentés dans le chapitre 7. La stabilité de chacun est étudiée à l'aide de la théorie de stabilité des solutions aux équations différentielles à retard.

Le chapitre 8 est consacré aux méthodes d'évaluation des algorithmes de contrôle de flux pour l'ABR. Nous définissons ainsi, les critères essentiels qu'un algorithme doit satisfaire pour être stable, convergent et équitable. Des exemples d'algorithmes où au moins un de ces critères n'est pas satisfait sont décris. Une approche analytique pour le confirmer, ainsi que des simulations sont présentées.

# Chapitre 5

# Gestion du trafic dans l'ATM

Les réseaux à haut débit, utilisant la technique ATM (Asynchronous Transfer Mode) en particulier, viennent pour répondre tant à l'attente des utilisateurs qu'à celles des constructeurs quant à pouvoir multiplexer différents trafics. Cela pourrait remplacer un nombre de réseaux (réseaux téléphonique, réseaux de données, câbles TV...etc) par un seul, capable de supporter toute une variété d'applications avec des caractéristiques toutes différentes les unes des autres.

Certaines de ces applications (besoins des utilisateurs) tolèrent les délais, tel est le cas des transferts de fichiers, d'autres ont une dépendance très fortes du délai et de sa variation, mais avec de petites tolérances aux pertes (comme la "HI-FI" par exemple). Pour supporter cette variété de trafics tout en respectant les besoins de chaque, des classes de services ont été définies. Chaque classe de service a des paramètres propres permettant ainsi, de définir une qualité de service qu'une application souhaiterait avoir, ou le cas échéant que le réseau est capable d'offrir.

Pour répondre à ces besoins, les réseaux ATM ont apporté des nouveautés par rapport à l'existant, à savoir une taille fixe de 53 octets des entités de données appelées cellules ATM, l'asynchronisme, le contrôle de congestion (et d'admission) qui est régulé non par par les applications (TCP sur Internet par exemple) mais par le réseau lui même, les priorités assignées à chaque type de service et enfin le multiplexage statistique. Ce dernier permet au réseau d'optimiser l'utilisation des ressources [70]. Le réseau contrôle également le trafic d'une source donnée pour s'assurer que celle-ci respecte le contrat passé lors de son admission. Dans ce qui suit, nous verrons brièvement tous les points ci-dessus [1].

## 5.1 Contrat de trafic et mécanismes de gestion du trafic

### 5.1.1 Contrôle d'Admission d'une Connexion (CAC)

Le CAC est définit comme l'ensemble des actions prises par le réseau lors d'une demande d'établissement d'une connexion, pour déterminer s'il doit accepter ou rejeter l'appel [1] (pp. 2). Il peut également être sollicité dans le cas où une source voudrait renégocier sa qualité de service (ses paramètres). Aucune spécification n'est donnée par l'ATM Forum quant aux décisions à prendre, il est donc laissé à l'appréciation de chaque constructeur et n'est pas standardisé.

Plusieurs propositions ont été faites [17, 76, 101, 105], certaines reposent sur le cas pire (worst case), d'autres sur la bande passante effective [64, 65] ou encore sur des mesures (statistiques). Le CAC a un rôle préventif en le sens où une nouvelle connexion ne serait pas admise, si le réseau juge qu'elle est susceptible de perturber les connexions déjà existante. Cette décision de rejet ou d'admission est prise en fonction de la qualité de service que la connexion souhaiterait avoir, et les statistiques sur les ressources qui lui seront nécessaires.

### 5.1.2 La fonction UPC

La fonction UPC (Usage Parameter Control) est définie comme les actions prises par le réseau pour gérer et contrôler le trafic, en terme du trafic offert et de la validité de la connexion ATM, aux accès du réseau [1, 70]. Son objectif est de protéger le réseau des applications malicieuses, aussi bien que des erreurs non intentionnelles qui peuvent affecter la qualité de service des connexions déjà établies. A la réception des cellules, l'UPC détermine si elles correspondent à une connexion valide et si elles respectent le contrat de trafic passé par la connexion. Dans le cas contraire, il a la capacité de prendre les mesures qui s'imposent, en considérant ces cellules comme *non conforme*, et peuvent par conséquent être, soit rejetées (perdues) ou marquées pour être les premières candidates aux pertes. Pour cette fonction également, différentes propositions ont été faites. La méthode du *Leaky Bucket* [70] est sans doute la plus admise.

### 5.1.3 Contrat de trafic

Pour pouvoir déterminer si une connexion respecte le contrat passé avec le réseau (UPC) ou si le réseau est capable d'admettre une nouvelle connexion (CAC), un certain nombre de paramètres ont été définis à cet égard. Chaque service a des paramètres spécifiques, qui peuvent figurer dans un autre service également.

Ces paramètres sont des paramètres définissant la qualité de service (QoS), ou des paramètres descripteurs de trafic d'une connexion.

#### 5.1.3.1 Paramètres pour la qualité de service

Paramètres négociables :

- maxCTD (maximum Cell Transfer Delay) : le temps de propagation maximum entre la source et la destination (niveau UNI).
- peak-to-peak CDV (peak-to-peak Cell Delay Variation) : la variation maximum du temps de propagation. Le terme "peak-to-peak" se réfère au meilleur et au pire des cas de CTD.
- CLR (Cell Loss Ratio) : le taux de perte de cellule définit comme, le nombre de cellules perdues divisé par le nombre total de cellules transmises.

Paramètres non négociables :

- CER (Cell Error Ratio) : le taux de cellules erronées, qui est le nombre de cellules erronées divisé par le nombre de cellules bien transmises plus le nombre de cellules erronées.

- SECBR (Severely-Errored Cell Block Ratio) : le taux de blocs de cellules erronés. C'est le nombre de blocks erronés sur le nombre total de blocks transmis. Un bloc est typiquement le nombre de cellules de données transmises entre deux cellules OAM (Operations, Administration and Maintenance).
- CMR (Cell Misinsertion Rate) : le débit de cellules insérées par erreur. Ce paramètre n'est pas définit comme un taux de cellules insérées par erreur, pour la simple raison que ces cellules ne dépendent pas du nombre de cellules transmises par une connexion. Ainsi, ce débit est le nombre de cellules insérées par erreur, pendant un intervalle de temps divisé par ce même intervalle, qui peut être la durée de la connexion.

#### 5.1.3.2 Descripteur de trafic

Ce descripteur donne un aperçu sur le comportement future de la connexion. Il se résume en un descripteur de trafic source, de la tolérance aux variations du délai de propagation CDVT (Cell Delay Variation Tolerance), et de la définition de conformité.

Descripteur de tafic source :

C'est un essemble de paramètres de la source ATM. Il est utilisé durant l'établissement d'une connexion pour capturer les caractéristiques intrinsèques du trafic de celle-ci [1]. Il peut être composé des paramètres de trafic négociables suivants :

- PCR (Peak Cell Rate) : débit crête de la source.
- MCR (Minimum Cell Rate) : débit minimum de la source.
- SCR (Sustainable Cell Rate) : débit soutenu (voir VBR, ABT).
- MBS (Maximum Burst Tolerance) : taille maxiale des rafales.

Pour des besoins de conformité, le paramètre BT (Burst Tolerance) est introduit [70]. Il est définit comme $BT = (MBS - 1)(SCR^{-1} - PCR^{-1})$.

CDVT :

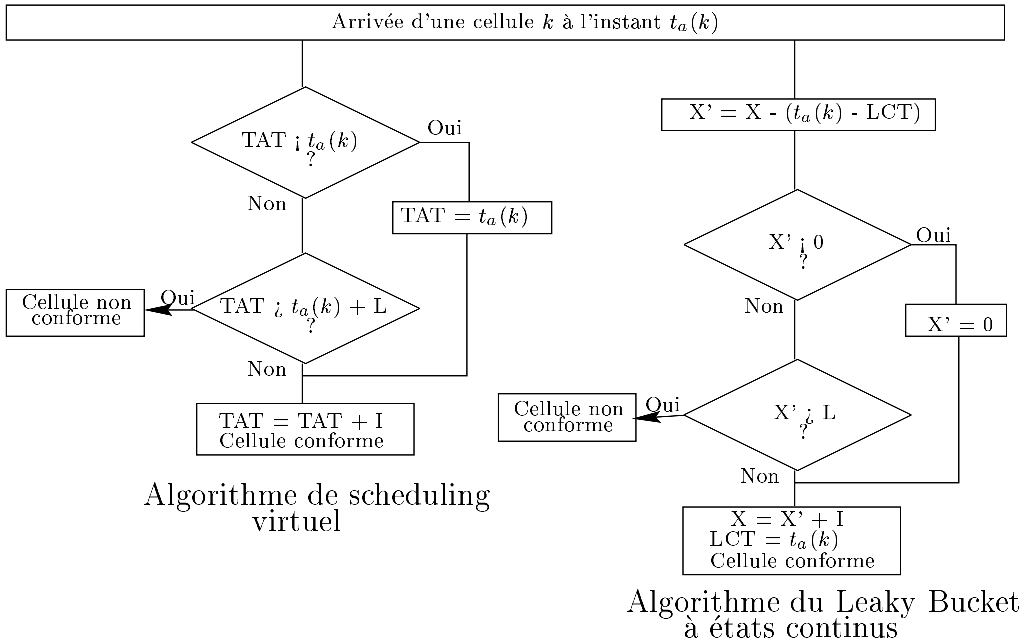
Le multiplexage de cellules peut altérer les caractéristiques d'un trafic, en introduisant une variation du délai entre les cellules. En effet, la transmission de cellules d'une connexion peut être retardée à cause de la transmission des cellules d'une autre connexion. Par conséquent, un délai aléatoire apparaît entre les cellules successives d'une même connexion. Le test de conformité se rapportant uniquement au temps théorique entre deux cellules d'une même connexion (l'inverse de PCR) pourrait rejeter des cellules tout à fait conformes. Ainsi, CDVT est introduit et représente la limite imposée de ces variations de délai.

Conformité :

L'algorithme GCRA (Generic Cell Rate Algorithm) est désigné pour déterminer la conformité des cellules, essentiellement pour pousser les utilisateurs à respecter le contrat passé avec le réseau en toute transparence. La figure 5.1 illustre deux versions équivalentes du GCRA. La notion GCRA(I, L) indique que les paramètres de l'algorithme sont $I$ et $L$ pour l'incrément et la limite, respectivement. A chaque débit est associé un GCRA. Le GCRA(PCR$^{-1}$, CDVT), par exemple permet de définir les cellules non conformes au paramètre PCR.

D'autres mécanismes de gestion de trafic existent également. On peut citer :

- Le lissage de trafic qui consiste à modifier le comportement du trafic d'une connexion, afin d'utiliser au mieux les ressources du réseau.




| | |
|---|---|
| TAT | Temps d'Arrivée Théorique |
| $t_a(k)$ | Instant d'arrivée de la cellule |
| X | La valeur du compteur du Leaky Bucket |
| X' | Variable auxiliaire |
| LTC | Temps précédent de conformité |
| I | Incrément |
| L | Limite |

A l'instant d'arrivée de la première cellule de la connexion, TAT = $t_a(1)$

A l'instant d'arrivée de la première cellule de la connexion, X=0 et LTC = $t_a(1)$

FIG. 5.1: Deux versions équivalentes du GCRA

- EFCI : Un commutateur ATM, peut altérer le contenu du bit EFCI (Explicit Forward Congestion Indication), pour signaler d'éventuelles congestions du commutateur, aux extrémités. Celles-ci peuvent alors réduire leurs débits de transmission pour avoir une faible probabilité de perte.
- Généralement, la perte d'une seule cellule dans une trame rend toute la trame inutilisable par les extrémités. Envoyer une telle trame accentue simplement la congestion dans le réseau, sans qu'elle soit utile. Ainsi, un mécanisme (tel EPD : Early Packet Discard [43]) peut être implémenté pour pouvoir dans ces cas rejeter toute la trame, et libérer de l'espace pour les prochaines.

## 5.2 Le service CBR

Le service CBR est définit par l'ATM forum dans le but d'assurer le transport des trafics fortement isochrones, dont les contraintes de temps d'acheminement et de la variation du délai sont beaucoup plus fortes que dans les autres types de trafics. Son correspondant dans UIT est le DBR (Deterministic Bit Rate)

Les ressources dont ce trafic peut en disposer sont calculées à base du PCR de la connexion (Peak Cell Rate). Ainsi, toute cellule émise à un débit inférieur à PCR est conforme. Ce service garantit les trois critères de service suivants : CLR, maxCDT et peak-to-peak CDV.

En somme, c'est une émulation de circuit, c'est le service le plus simple à implémenter. Tous les commutateurs sur le marché le garantissent. La téléphonie de haute qualité est l'application type utilisant ce service. Les trafic VBR temps réel également peuvent utiliser ce service en allouant un débit crête PCR, si la source est incapable de connaître toutes les caractéristiques du trafic hormis le débit crête [70].

## 5.3 Les services rt-VBR et nrt-VBR

Beaucoup d'applications ayant des contraintes temporelles assez fortes ne peuvent se permettre de négocier une connexion CBR, à cause de la variabilité de leur débit. Ouvrir une connexion au débit crête coûterait très cher pendant que l'utilisation est très faible. Le service VBR est défini pour supporter ce genre d'applications. Les paramètres négociés dans ce cas sont le PCR, SCR et le MBS.

Deux sortes de services VBR existent : le rt-VBR (real time-VBR) avec de forte contrainte de délai, pour qui le paramètre CDV est négocié, et le nrt-VBR (non real time-VBR) auquel seul le taux de perte est garanti. Une source peut transmettre au débit constant de PCR. Seulement, si le marquage est utilisé, alors L'UPC marquera les cellules en excès du contrat SBR/MBS du flux CLP0 (le flux à haute priorité devient un flux à faible priorité), et n'auront ainsi aucune garantie quant à la probabilité de perte. La source dans ce cas peut être informée par la destination, et réduire son débit. En fait, la conformité dans ce service obéit à deux GCRA ; GCRA($PCR^{-1}$, CDVT) et GCRA($SCR^{-1}$, BT+CDVT). Pour plus de détail, voir [1, 70]. Le correspondant de cette capacité de transfert (VBR) dans les recommandations de l'UIT [2] est le SBR (Sustainable Bit Rate).

## 5.4 Le service ABR

C'est le service qui nous intéressera le plus au cours de ce qui suit. Par conséquent, une description plus détaillée serait une nécessité et un atout pour la suite. Ce service est destiné essentiellement aux sources capables de moduler leurs débits de transmission. Ainsi, il permet la récupération de la bande passante non utilisée par les services de plus grande priorité, à des instants réguliers ou variables, et de l'attribuer de manière équitable sur l'ensemble des connexions concurrentes.

Les comportements de la source et de la destination sont spécifiés par l'ATM forum [1] ; est spécifiée également la façon dont les informations (de congestion et autres) sont transmises à la source (voir ci-après). Par contre, le comportement du commutateur est laissé aux constructeurs.

Une source ABR commence à envoyer à un débit ICR (Initial Cell Rate) et peut envoyer des cellules à un débit au moins égale à MCR (Minimum Cell Rate), si celui-ci est négocié plus grand que zéro dès l'ouverture de la connexion. Le débit en général ne doit jamais dépasser le ACR (Allowed Cell Rate). Ce dernier, selon l'état de congestion du réseau, varie entre le MCR et le PCR (Peak Cell Rate) négociés par la source à l'établissement de la connexion. L'état du réseau est spécifié à la source à l'aide des cellules de "contrôle" appelées cellules RM (Ressource Management). Elles permettent d'indiquer à la source de réduire ou d'augmenter leur débit de transmission, ou de leur spécifier exactement le débit auquel elle doivent transmettre, selon l'algorithme de partage de la bande passante implémenté au niveau des commutateurs.

Une cellule RM toutes les $N_{rm}$ cellules de données est envoyée. Elle effectue un aller retour entre la source et la destination et collecte des informations à son passage au niveau des commutateurs. On spécifiera par "en aval" la direction de la source vers la destination (forward) et "en amont" la direction inverse (backward).

### 5.4.1 Paramètres pour le contrôle de flux dans l'ABR

Différents paramètres existent dans le service ABR, certains d'entre eux sont négociés ou calculés obligatoirement à l'établissement d'une connexion, pendant que d'autres sont facultatifs ou fixes :

- Les paramètres négociés à l'établissement d'une connexion :
  - PCR, MCR et ICR ; ICR est le débit de transmission juste après l'ouverture d'une connexion ou après une période de silence, de durée supérieure à ADTF (voir ci-dessous) ;
  - RDF (Rate Decrease Factor) et AIR (Additive Increase Rate) correspondent, respectivement au facteur de réduction et au taux d'augmentation du ACR à la réception d'une cellule RM ;
  - TBE (Transient Buffer Exposure), nombre maximum de cellules pouvant être envoyées par la source avant de recevoir la première cellule RM (lié à ICR) ;
  - FRTT (Fixed Round Trip Time) le délai d'aller retour entre la source et la destination, nécessaire pour le calcule de ICR qui doit être inférieur à $TBE/FRTT$.
- Les paramètres facultatifs :
  - $N_{rm}$, nombre de cellules de données à envoyer avant l'envoi d'une cellule RM. Par défaut ce paramètre est mis à 32 ;
  - $T_{rm}$, durée minimale entre deux cellules RM. Si pendant $T_{rm}$ aucune cellule RM n'est transmise, alors une cellule sera envoyée avec une faible priorité, appelée *out-of-rate* ; ceci pour recouvrir des situations de famine où le ACR peut être nul ;
  - ADTF (ACR Decrease Time Factor) délai minimum pour considérer une période d'inactivité comme un sommeil (on appellera par la suite les source ayant ce comportement par des sources silencieuses ou sources on-off) ;
  - CDF (Cutoff Decrease Factor), contrôle la décroissance de ACR, il est associé à CRM.

  Enfin les paramètres non négociables
  - CRM (Missing RM-cell Count) limite le nombre de cellules RM envoyées entre deux cellules RM "en amont" reçues (revenant à la source). Il est calculé comme suit : $CRM = TBE/N_{rm}$

### 5.4.2 Cellule RM

Comme souligné précédemment, les cellules RM ont un rôle primordial dans le contrôle de flux et l'ajustement des débits au niveau des sources ABR. Les principaux champs que comporte une cellule RM sont les suivants :

- DIR, indiquant la direction de la cellule "en aval" ou "en amont" ;
- BN, mis à un, il indique que la cellule RM est générée non pas par la source mais par un commutateur intermédiaire ou une destination.
- CI (Congestion Indication) bit indiquant à la source si une congestion est détectée par un commutateur, à ce moment la source doit réduire son débit de transmission ACR ;
- NI (No Increase) bit indiquant à la source, quand il est à un, qu'elle doit garder son débit inchangé ;
- ER (Explicit Rate), dans le cas des commutateurs pouvant calculer exactement le débit alloué à la connexion, cette dernière doit se conformer exactement à ce débit de transmission, et ne peut ainsi envoyer à un débit supérieur. Ce champs est initialisé par la source à PCR ;
- CCR (Current Cell Rate), pour informer les différents commutateurs du débit courant de la source. Ce débit peut être instantané ou une moyenne.
- MCR (Minimum Cell Rate), pour informer les différents commutateurs du débit minimal négocié par la source.

### 5.4.3 Comportement d'une source

Une des contraintes principales d'une source ABR est de pouvoir transmettre à un débit modulable ou variable calculé soit à partir des bits CI et NI des cellules "en amont", ou recopier directement le champs ER de ces dernières si celui-ci est effectif.

Dans le cas où CI et NI sont effectifs, la source calcule son débit de transmission en le révisant à la hausse, en lui additionnant $N_{rm}.AIR$ si les bit CI et NI de la cellule RM reçue sont à zéro, et garde son débit inchangé si le bit CI est à zéro et NI est à un. Elle réduit son débit multiplicativement d'un facteur RDF, si le bit CI de la cellule RM reçue est à un. L'algorithme de la mise à jour du débit à la source, à la réception d'une cellule RM, est le suivant :

```
Si (CI == 1)
   ACR := ACR - Nrm * ACR/RDF;
Sinon
   Si (NI == 0)
      ACR := ACR + Nrm * AIR;
   Sinon
      Garder le débit inchangé;

ACR := Min (ACR, PCR);
ACR := Max (ACR, MCR);
ACR := Min (ACR, RM->ER);
```

Si pendant ADTF ou plus, la source est restée silencieuse, alors elle est considérée comme étant au sommeil. A son réveil, elle ne peut transmettre à un débit supérieur à ICR, car elle ne dispose plus d'estimation correcte du débit disponible dans le réseau. ACR est réduit implicitement à ACR := ACR.CDF, si aucune cellule RM "en amont" n'est reçue, pendant que plus de CRM cellules RM "en aval" sont envoyées (pour plus de détail voir [60, 61]).

Une autre fonction de la source est la génération des cellules RM. Le flux des cellules RM doit être proportionnel au flux de données, c'est pourquoi le choix d'une cellule RM toutes les

$N_{rm}$ cellules de données a été préféré à la génération périodique. Il se peut malheureusement que le débit de la source soit nul, et aucune cellule RM ne peut être envoyée, engendrant ainsi une méconnaissance totale de l'état du réseau qui pourrait complètement changé entre temps. Cette situation est appelée situation de famine, en y remédie en envoyant une cellule RM à faible priorité appelée *out-of-rate*, si aucune cellule n'a été transmise pendant $T_{rm}$ secondes. $T_{rm}$ est typiquement 100 ms, pour obtenir un débit de 10 cellules par seconde comme débit minimal.

### 5.4.4 Comportement d'une destination

La destination doit retourner les cellules RM qu'elle reçoit d'une source en changeant le bit de direction à "en amont" (Backward). Elle met également le champs CI d'une cellule RM à la valeur EFCI de la dernière cellule de données reçue, de la source en question.

### 5.4.5 Comportement général d'un commutateur

Après avoir défini tous les champs nécessaires au contrôle de flux dans l'ABR, voyons maintenant le comportement des commutateurs. Comme mentionné précédemment, il n'existe aucune spécification précise sur ce dernier et chaque constructeur peut implémenter tel ou tel algorithme. Beaucoup d'algorithmes ont été proposés, la liste est assez longue pour les énumérer tous. Toutefois, on peut distinguer deux catégories : les commutateurs basés sur l'indication de congestion (EFCI-based switch) (ou la taille exacte des files d'attente au commutateurs), et ceux basés sur le débit explicite qui peuvent utiliser également la taille des files comme paramètre de contrôle (voir la figure 5.2).

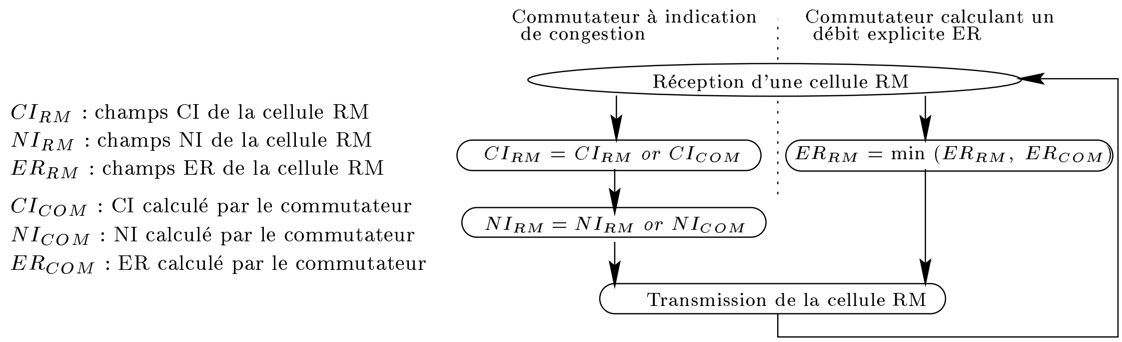



FIG. 5.2: Fonctionnement d'un commutateur à débit explicite et à indication de congestion (EFCI)

#### Commutateurs basés EFCI

Dans cette catégorie de commutateurs, selon le degré de congestion, le commutateur peut altérer le contenu d'une cellule RM dans les deux directions ("en aval" ou "en amont"). Les valeurs de CI et NI sont mises à jour en fonction de la congestion estimée au niveau du commutateur. Les commutateurs basés sur l'indication de congestion que nous allons considérer dans cette thèse fonctionnent de la façon suivante :

Si la taille de la file d'attente dans le commutateur dépasse un certain seuil $Q^H$ (appelé seuil de congestion), le commutateur met à un le bit EFCI dans les cellules de données, jusqu'à ce que la file soit en dessous de ce seuil. Si par contre, la taille de la file est entre le seuil $Q^H$ et un autre seuil $Q^L$, alors le commutateur se considère être dans son point de fonctionnement optimal, et averti ainsi les sources de garder leurs débits inchangés. Ceci est fait en positionnant le bit $NI$ des cellules RM à un (le bit EFCI des cellules de données est laissé à zéro). Quand la destination reçoit une cellule RM ("en aval"), elle met le bit CI à la valeur du bit EFCI de la dernière cellule de données reçue. Le commutateur peut lui même mettre à jour le bit CI d'une cellule RM ("en amont") si congestion y a.

Ces algorithmes sont beaucoup plus simples à implémenter, cependant les performances obtenues sont assez limitées ; elles dépendent de nombreux paramètres négociables. Ainsi, il est extrêmement difficile de faire le choix optimal de ces derniers dans un environnement très variable. Les performances de ces algorithmes dépendent également, comme on le verra plus en détail dans le chapitre suivant, du nombre de commutateurs traversés par une connexion.

#### Commutateurs basés ER

Cette famille de commutateur a la capacité d'estimer la bande passante disponible pour le service ABR, ce qui lui permet de notifier de manière quasi exacte la bande passante allouée à chaque utilisateur, en mettant à jour le champs ER des cellules RM ("en aval" et/ou "en amont").

Les commutateur à débit explicite peuvent cohabiter avec les commutateur à indication de congestion. Seulement, le débit explicite dans ce cas aura peut de chance d'être effectif, et c'est l'indication de congestion qui l'emporte. Quand seuls des commutateurs à débit explicite sont traversés par une source, celle-ci négocie un $AIR$ (Additive Increase Rate) égal à PCR, pour que le champs ER des cellules RM retournant à la source soit pris en compte, dans toutes les situations (voir section 5.4.3) . En effet, dans ce cas les commutateurs basés ER n'affectent pas le champ CI des cellules RM. Par conséquent, à la réception de la première cellule RM par la source, celle ci augmente son débit de $N_{rm} * PCR$, qui résulte d'un ACR calculé au moins égale à PCR. Ainsi, en prenant le minimum entre le $ACR$ calculé et la valeur du champ ER de la cellule RM, ACR sera dans tous les cas mis à jour à la valeur ER, du moment que celle-ci est inférieur à PCR.

Ce type de commutateurs, bien qu'ils soient d'une complexité relative, permettent une meilleure utilisation et répartition de la bande passante tout en minimisant les pertes. Tout au long de cette partie nous décrirons différents algorithmes basés sur le débit explicite, au même temps que leur évaluation est faite, cela nous évitera des redondances.

## 5.5 Le service ABT

Ce service (ATM Block Transfer) a été proposé par le CNET (Centre National D'études en Télécommunications) [2], il est à l'étude à l'UIT (Union International des Télécommunications), mais pas à l'ATM Forum. Il est basé sur la notion de bloc ATM [70] délimité par deux cellules RM (Ressource Management). Ces cellules RM servent à faire la demande de réservation de ressources pour les prochains blocs, notamment le paramètre de trafic BCR (Block Cell Rate). Toutefois, certains paramètres sont négociés dès le départ, et la source tant qu'elle ne les viole pas est assurée d'avoir la qualité de service associée. Ces paramètres sont :

- Le débit crête $PCR_{RM}$ qui limite la fréquence des demandes de réservations (renégociation de BCR), et $CDVT_{RM}$ associé à ce débit crête.

- Le débit crête PCR, pour l'ensemble des cellules (CLP = 0+1), ainsi que le CDVT qui lui est associé.

- Le débit soutenable SCR (Sustainable Cell Rate) et BT (Burst Tolerance), également pour l'ensemble des cellules (CLP =0+1), et le CDVT associé. SCR peut être nul, à ce moment là si un bloc est transmis en l'absence de ressources dans le réseau, celui-ci peut être rejeté.

Deux versions de ce service sont définies :

### ABT/DT (Delayed Transmission)

Dans cette version, la source attend d'avoir un acquittement positif pour transmettre un bloc au débit négocié BCR. Dans ce cas, le bloc en question aura la même qualité de service que celle du service CBR. De toute façon elle peut toujours envoyer au débit SCR, si celui-ci n'est pas nul. Plusieurs modes de fonctionnement existent quant aux réponses du réseau :

- Le mode rigide : Le réseau répond positivement ou négativement à une requête, à l'aide du bit CI de la cellule RM. Dans le cas d'une réponse négative, la source ne peut que faire une deuxième demande.

- Le mode élastique : Dans ce mode, le réseau peut spécifier un débit inférieur à la demande, dans le cas où celle-ci ne peut pas être satisfaite. Dans tous les cas, si les ressources dans le réseau sont insuffisantes, un bloc est soit totalement détruit ou accepté, il ne peut y avoir d'acceptation partielle [70]

Les décisions de conformité sont prises séparément pour le flux de cellules RM et de données. Les cellules RM satisfaisant la fonction GCRA($PCR_{RM}^{-1}$, $CDVT_{RM}$) sont conformes. Le test de conformité du paramètre BCR pour le flux de données, est effectué à l'aide d'un GCRA dynamique. Tant que les blocs sont conformes au paramètre SCR, les requêtes doivent être acceptées et dans tous les cas la qualité de service offerte pour des blocs acceptés doit être pareille à celle du service CBR, en termes de probabilité de perte et de variation de délai. Ceci dit, que l'ABT est comparable au service CBR négocié bloc par bloc. Une proposition se rapprochant de cette idée (ABT) a été faite dans [45] (R-CBR).

Des traits de ressemblances entre ce service et le service ABR, peuvent être visibles à partir de l'utilisation des cellules RM et de l'ajustement du débit de la source. Surtout pour le mode élastique, qui lui a tendance à récupérer la bande passante disponible. Toutefois, des différences fondamentales existent, à savoir que le service ABR ne fait aucune réservation de ressources si le MCR est nul, et ne peut avoir des garanties sur les délais de transfert.

L'inconvénient de l'ABT/DT est le délai de négociation qui est au moins égal au délai d'aller retour entre la source et la destination. Par conséquent, lors d'un refus d'allocation (en mode rigide), la source doit attendre un certain nombre de délais d'aller retour pour pouvoir transmettre, ce qui risque de nuire à la qualité de service qu'une connexion s'attend à avoir. Le mode élastique apporte des éléments de réponse à ce problème et diminue ainsi l'attente. Une autre version de l'ABT (ABT/IT) visant à résoudre ce problème est proposée.

### ABT/IT (Immediate Transmission)

Dans cette version, la source n'attend pas confirmation pour transmettre. Un bloc est transmis juste après la cellule RM. Si les ressources dans le réseau sont insuffisantes, le bloc est alors détruit. Dans cette version, une partie d'un bloc peut être stockée dans le réseau en modifiant le BCR contrairement à l'ABT/DT [82].

## 5.6 Le service UBR et UBR+ (ou GFR)

Le service UBR (Unspecified Bit Rate) [1] est le correspondant de l'Internet dans l'ATM, où aucun contrôle n'est effectué par le réseau. La source envoie ses données à leurs risques et périls. C'est le service le moins coûteux et le plus simple. Il est parfois plus adéquat pour des applications utilisant TCP pour le transfert de données, dans le cas où certains mécanismes tels que EPD (Early Packet Discard) [43] lui sont associés. Il est destiné essentiellement aux sources ne pouvant pas disposer d'un mécanisme leur permettant d'envoyer à un débit variable, et ne peuvent par conséquent avoir tous les bénéfices du service ABR en particulier, aussi celles ne pouvant spécifier aucun paramètre de trafic tels que SCR ou MBS.

Une amélioration a été proposée pour ce service qui vise à minimiser les pertes. C'est le service UBR+ ou GFR (Minimum Garanteed Rate) [46, 47]. Il garantit un taux de perte minimum et un débit de transmission minimal de MCR, si les rafales n'excédent pas deux fois la taille maximale d'un paquet (SDU) : MBS = 2*CPCS-SDU en cellules. La conformité des cellules de ce service est testée à l'aide d'un GCRA(PCR$^{-1}$, CDVT). De plus pour déterminer si un paquet est éligible (dans la limite du débit minimum garanti), il doit être conforme au GCRA(1/MCR, Burst_Tolerance(MBS)+CDVT) modifié comme suit :
Si à l'arrivée de la première cellule d'un paquet, le nombre de jetons = $TOKEN\_COUNT \geq MBS/2$ = CPCS-SDU size, alors toutes les cellules du paquet sont déclarées éligibles, et chacune d'elles consomme un jeton à sa transmission. Si par contre $TOKEN\_COUNT < MBS/2$, alors toutes les cellules du paquet sont déclarées inéligibles et aucune d'elles ne consomme de jetons lors de sa transmission. Ceci assure la transmission avec une faible probabilité de perte, au pire de toutes les cellules déclarées éligibles, et au mieux de toutes les cellules transmises par la source.

Pour des besoins d'équité entre différentes sources, la politique de WFQ (Weighed Fair Queueing) peut être utilisée lors de l'insertion des paquets dans la file GFR au niveau d'un commutateur donné. Elle fonctionne comme suit [47] :

- Un paquet éligible sera rejeté de la file ((toutes les cellules du même paquet seront détruites), si la première cellule du paquet trouve une taille de la file d'attente excédant un seuil $HBO$ et qu'au même temps, le nombre de cellules insérées dans la file appartenant à ce même flux $i$ dépasse un seuil $T_i = MBS_i$. Il est également rejeté si la taille de la file d'attente est à son point maximum. Dans tous les autres cas il sera inséré (toutes les cellules du même paquet seront insérées).

- Un paquet marqué inéligible est rejeté si la taille de la file dépasse un certain seuil $LBO$. Il est inséré dans les autres cas.

## 5.7 Équité "max-min" et équité proportionnelle

Le problème d'équité a suscité beaucoup d'interrogations et de recherche. En effet, il est très difficile de définir un système équitable, dans la mesure où certaines sources peuvent être restreintes : passant par des commutateurs où la congestion est très forte et par d'autres où la bande passante est suffisamment grande. Ajouter à cela qu'une connexion passant par plusieurs commutateurs peut dégrader le rendement du réseau, si une bande passante égale à celle attribuée aux autres connexions lui est attribuée [67]. C'est l'inconvénient reproché à l'équité au sens "max-min" que nous allons décrire dans ce qui suit :

### Équité au sens max-min

La majorité des algorithmes conçus pour l'ABR, comme on le verra par la suite, reposent sur ce principe d'équité [59, 83]. La définition de cette équité peut être donnée comme suit [78] : Soient $\mathcal{R}$ l'ensemble des chemins sur un réseau ou l'ensemble des flux (connexions), $R$ le nombre de tels chemins, et $\mathcal{L}$ l'ensemble des liens du réseau (ressources), où chaque lien $l$ à la capacité $C_l > 0$ . On note $l \in r$ où $r$ est un chemin, si le chemin $r$ traverse le lien $l$. Un partage du débit sur l'ensemble des chemins $(x_1, \cdots, x_R)$ est "max-min" équitable, si pour tout chemin $r$, il existe au moins un lien $l \in r$ tel que

$$\sum_{r' \ni l} x_{r'} = C_l \quad \text{et} \quad x_r = \max(x_{r'}, r' \ni l)$$

Bien évidemment avec la contrainte

$$\sum_{l \in r} x_r \leq C_l \quad l \in \mathcal{L}, \tag{5.1}$$

qui signifie que la somme des débits alloués aux connexions partageant le même lien est inférieure ou égale à la capacité du lien en question. Si le nombre de liens et de chemins sont finis, alors il existe un seul partage du débit satisfaisant le critère "max-min".

### Équité proportionnelle

Récemment F. P. Kelly [67] a remis en cause l'efficacité de l'équité au sens "max-min", et a proposé une notion alternative à cette dernière. Cette notion suppose qu'un partage de débit sur un ensemble d'utilisateurs est équitable, s'il maximise la fonction $\sum_{\mathcal{R}} \log x_r$ sous la contrainte (5.1), où $\mathcal{R}$ et $\mathcal{L}$ sont définis précédemment. Dans le cas d'un nombre de chemins et de liens finis, le vecteur de partage $(x_1, \cdots, x_R)$ ainsi défini est unique. Il est caractérisé par le fait que, pour tout autre partage $(x'_1, \cdots, x'_R)$ la relation suivante est vérifiée :

$$\sum_{\mathcal{R}} \frac{x'_r - x_r}{x_r} \leq 0.$$

Dans [78], les auteurs confirment l'observation que les protocoles classiques de congestion réalisent plutôt cette équité que le "max-min". Quelques exemples d'algorithmes ont été considérés (fenêtre fixe, Metropolis, algorithmes à croissance linéaire et décroissance exponentielle du débit). Dans [37], un algorithme pour l'ABR, incluant la tarification et prenant compte de l'équité proportionnelle a été proposé. Le champ de recheche de ce coté est très ouvert.

D'autres critères d'équité pour l'ABR, prenant en compte des connexions ayant des MCRs différents existent. Ces critères se rapprochent plus de l'équité au sens "max-min" que de l'équité proportionnelle, bien que le débit soit alloué de manière proportionnelle à MCR. Comme exemples [1] :

- MCR plus partage équitable : la bande passante allouée à une connexion est son MCR, plus une fraction de la bande passante totale après avoir soustrait la somme de tous les MCRs de celle-ci. $ER_i = MCR_i + (B - M)/N$ où $M$ est la somme de tous les MCRs.
- Maximum entre MCR et le partage "max-min".
- Allocation proportionnelle à MCR : la bande passante allouée à une connexion $i$ est proportionnelle au MCR de celle-ci. $ER_i = B * (MCR_i/M)$.
- Allocation pondérée : à chaque connexion $i$ est associé un poids $w(i)$. Les allocations de bandes passantes sont proportionnelles à ces poids. $ER_i = B * \Big(w(i)/\sum_j w(j)\Big)$.

# Chapitre 6

# Analyse de l'indication de congestion

Plusieurs paramètres ont une influence certaine sur les performances et la qualité de service (QoS) des connexions ABR. Quelques uns de ces paramètres sont fixes. C'est le cas par exemple de la capacité maximale des files d'attente, de la bande passante disponible pour le service ABR, et du délai d'aller retour. D'autres, peuvent en revanche, être négociés à l'établissement d'une connexion. Il est par conséquent, crucial d'avoir des méthodes qui permettent au réseau de choisir ces paramètres en fonction des paramètres fixes et de la qualité de service à laquelle il souhaite répondre.

Dans le but de proposer de telles méthodes, des modèles de files d'attente simples ont étés développés et étudiés durant ces dernières années. En particulier, ont été développés des modèles basés sur l'hypothèse que le réseau peut se modéliser par une seule file d'attente [21, 28, 50, 85, 92, 103].

Le but essentiel du présent chapitre est d'étudier le cas, le plus réaliste, où une connexion passe à travers plusieurs commutateurs ; nous examinons alors la validité de l'approche simplifiée où le réseau est résumé en une seule file (goulot d'étranglement).

Il est tout à fait évident que si le premier commutateur a une bande passante inférieure à celles des autres, l'empilement des cellules ne se produira pas dans les commutateurs en aval ; et par conséquent, l'approche d'une seule file peut être utilisée. A ce propos, nous allons donc restreindre notre étude au cas contraire ; nous considérons $N$ commutateurs le long du circuit virtuel, où la bande passante disponible pour l'ABR dans le $i$ème commutateur est supérieure à celle du $(i+1)$ème.

Nous considérons les commutateurs basés EFCI et ER. Deux objectifs sont visés par cette étude [8, 11] :

- Le premier objectif est de mieux comprendre le phénomène du goulot d'étranglement. Nous montrons en particulier, que ce n'est pas nécessairement le commutateur qui a la bande passante la plus petite qui est "responsable" de la congestion, ou qui se construit le premier.
- Le deuxième est de calculer des bornes sur les tailles des files d'attente en fonction des paramètres fixes (délai d'aller retour, les capacités maximales des files ...etc). Ceci permettra en particulier, de calculer les paramètres de contrôle qui permettent de n'avoir aucune perte dans le cas où les tailles maximales des files sont connues à priori. Ces

résultats nous permettrons également, de comparer la taille maximale de la file d'attente obtenue sous l'hypothèse que le réseau peut se simplifier en un nœud, aux tailles maximales des files dans le modèle à plusieurs commutateurs en tandem.

Le chapitre est structuré comme suit : dans la section 6.1, une description des algorithmes à analyser est donnée, nous mettrons l'accent sur l'algorithme EPRCA que nous considérons comme un algorithme basé sur le débit explicite (ER). L'autre algorithme analysé est l'algorithme basé sur l'indication de congestion (EFCI), dont la description a été donnée dans le chapitre précédent. Le modèle considéré est décrit dans la section 6.2. Les résultats analytiques obtenus, aussi bien pour la phase transitoire que la phase stationnaire, sont donnés dans la section 6.3. Dans la section 6.4, des exemples numériques sont considérés et les résultats obtenus par simulation sont comparés à ceux obtenus analytiquement. Enfin, nous concluons ce chapitre avec quelques remarques dans la section 6.5.

## 6.1 Description

Dans le chapitre précédent, une brève description du comportement de la source (*Source End System* (SES) [60]), de la destination (*Destination End System* (DES)) et du cheminement des informations de contrôle vers la source a été donnée. Nous nous contentons donc, de présenter uniquement les algorithmes implémentés au niveau des commutateurs que nous avons considérés dans la présente étude.

Dans ce chapitre nous allons évaluer les performances des commutateurs basés EFCI, déjà décrits dans le chapitre précédent (commutateurs à indication de congestion). Nous considérons que chaque commutateur a son propre seuil de congestion ($Q_i = Q_i^H = Q_i^L$). Ainsi, une fois que la taille de la file d'attente excède ce seuil, le commutateur est considéré comme congestionné, sinon des indications de non congestion sont envoyées à la source.

L'autre algorithme considéré est un algorithme très proche de EPRCA [49, 50, 93, 97] (Enhanced Proportional Rate Control Algorithm) :

### EPRCA

C'est un algorithme issu de PRCA [93] qui ne fonctionne qui ne fonctionne qu'à indication de congestion. Ce qui est apporté de nouveau par EPRCA, par rapport à PRCA est l'estimation de la bande passante disponible. Ceci est fait, en estimant le débit moyen général ($MACR$ ; Mean CellRate) de toutes les sources. On peut s'attendre ainsi à ce que dans le cas où les sources envoient au débit MACR, non seulement l'équité serait garantie, mais aussi une utilisation maximale de la bande passante, sans que la file d'attente ne déborde.

En effet, EPRCA utilise la taille de la file d'attente comme seuil de contrôle. Il utilise également deux seuils de congestion : QT (Queue Threshold) et DQT pour déterminer le degré de congestion. En cas de congestion, EPRCA réduit le débit de toutes les sources ayant un débit $CCR$ supérieur à $MACR$. Rappelons que ce dernier représente la moyenne générale de tous les $ACR$. Pour estimer sa valeur, EPRCA récupère le débit courant $CCR$ de chaque cellule RM "en aval" (forward) et effectue le calcul suivant :

$$MACR = MACR(1 - AV) + CCR_{cell}AV$$

où $AV$ est un paramètre constant égal à 1/16.

A noter que le débit moyen qu'on obtient avec cette estimation n'est pas la moyenne "réelle" de tous les débits, mais une moyenne où la source ayant un débit plus fort a plus d'effet. Car le nombre de cellules RM reçues en provenance d'une connexion donnée est proportionnel au débit de cette connexion (une cellule RM toutes les $N_{rm}$ cellules de données). En effet, on peut constater cela dans le chapitre 8 (section 8.1.2) dans le cas de l'algorithme UT (Uniform Tracking).

A la réception des cellules RM ("en amont" de préférence), EPRCA agit selon la taille de la file d'attente :

- Si la taille de la file d'attente est inférieure au seuil $QT$, le commutateur considère qu'il n'est pas congestionné, et la cellule RM n'est pas affectée (dans ce cas l'algorithme est équivalent à EFCI).
- Si la taille de la file d'attente est comprise entre $QT$ et $DQT$, le commutateur se considère en état de congestion, et met à jour le champs $ER_{cell}$ de la cellule RM comme suit : si le débit courant de la source $CCR_{cell}$ est supérieur à $MACR.DPF$, il estimera que le débit de la source est trop élevé et doit être réduit à

  $$ER = MACR.ERF.$$

  Les facteurs $DPF$ (Down Pressure Factor) et $ERF$ (Explicit Reduction Factor) ont une valeur de 7/8. Les sources ayant un débit courant inférieur à $MACR.DPF$ ne seront pas affectées.
- Si par contre, la taille de la file d'attente excède $DQT$, le commutateur se considère dans un état de forte congestion et réduit de manière sélective le débit de toutes les sources à

  $$ER = MACR.MRF.$$

  Où la valeur $MRF = 1/4 < ERF$ (Major Reduction Factor) est recommandée.

Les commutateurs basés ER que nous avons considérés dans ce chapitre s'inspirent de ceux utilisés dans [92]. Ces commutateurs sont très proche de EPRCA. En effet, considérons les paramètres suivants : $ERF = MRF$, $QT = DQT = Q_i$ où $Q_i$ est le seuil de congestion du commutateur $i$. Si on considère également, qu'au lieu d'évaluer le MACR, qui est l'estimation de l'ACR moyen de toutes les sources, le commutateur a connaissance de la bande passante réelle, alors l'algorithme basé sur le débit explicite évalué ci-dessous n'est autre qu'EPRCA.

On obtient alors un algorithme fonctionnant de la façon suivante :

En l'absence de congestion, ACR est augmenté de la même façon que dans les commutateurs basés EFCI. Dans le cas où la congestion est détectée (la taille de la file d'attente excède $Q_i$), le commutateur calcule alors le FS (Fair Share) en fonction de la bande passante disponible et les MCRs des différentes connexions actives. Dans notre cas, en présence de congestion, chaque commutateur calcule son $FS_i$ $(i = 1, \cdots, N)$. Si d'autres commutateurs sont congestionnés, c'est le minimum des FSs qui est considéré $(min_{i\geq 1}(FS_i) \stackrel{\Delta}{=} FS_N)$. $FSi$ est donné par $FSi = \mu_i - MCR_i$ où $\mu_i$ est la bande passante disponible au niveau du commutateur $i$. Cette valeur est multipliée par un facteur de réduction (*Explicit Reduction Factor* $(ERF_i = ERF, \ \forall i)$) plus petit que 1, puis ajoutée au MCR et le résultat est finalement signalé aux sources.

$$ERi = MCR + ERF * FSi \tag{6.1}$$

On définit alors, après avoir considéré $MCR_i = 0 \quad \forall i$

$$ER_i \stackrel{\Delta}{=} ERF * \mu_i \quad i = 1, \cdots, N$$

Des temporisateurs pour pallier aux situations de famine, où ACR est nul pour un long moment (100 ms) existent. Il est permis à la source dans ce cas, d'émettre des cellules RM appelées *out-of-rate*, à un débit ne dépassant pas $TCR = 10$ *cells/s* avec une priorité faible (voir [60] pour plus de détails). Ces mécanismes ne sont pas pris en compte dans le présent travail.

Dans le présent chapitre, aussi bien que dans les suivants, nous considérons que les cellules RM "en amont" et les cellules RM "en aval" sont mises à jour. Le cas où uniquement les cellules RM "en aval" sont mises à jour, pour les commutateurs EFCI aussi bien que ER, peut être vu comme le cas le pire (worst case), du moment que les informations sur la congestion prennent beaucoup plus de temps pour arriver aux sources. Les tailles des files d'attente sont alors beaucoup plus importantes au moment où l'étude qualitative demeure la même.

## 6.2 Modèle pour l'approche analytique

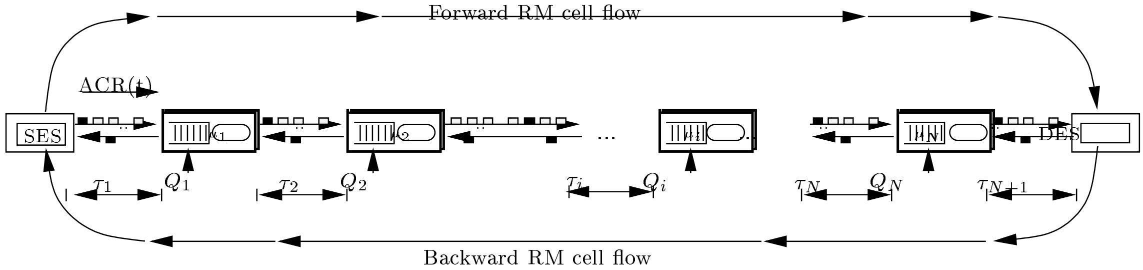



FIG. 6.1: Modèle à multiples commutateurs en tandem

Considérons une source ABR saturée (i.e. ayant des cellules à envoyer à tout instant) envoyant vers une destination (DES). La source et la destination sont séparées par de multiples ($N$) commutateurs, avec une bande passante disponible de $\mu_1, \cdots, \mu_N$, respectivement. On suppose que $PCR \stackrel{\Delta}{=} \mu_0 > \mu_1 > \mu_2 > \cdots > \mu_N > MCR$. On considère également que les tailles maximales des files sont suffisamment grandes pour qu'il n'y ait pas de pertes (on calculera ci-dessous les tailles garantissant zéro pertes). Les cellules RM et les cellules de données passent à travers les $N$ commutateurs avant d'arriver à la destination (Figure 6.1). Le temps nécessaire à une cellule RM dans un système vide, pour aller de la source vers la destination, et de retourner à la source est $\tau$ (temps de service non compris). Elle met $\tau_1$ entre la source et le premier commutateur, et généralement $\tau_i$ entre le $(i-1)$ème et le $i$ème commutateur, comme montré dans la figure 6.1. Le temps que met une cellule RM pour aller du dernier commutateur à la destination, puis retourner à la source dans un système vide est $\tau' \stackrel{\Delta}{=} \tau - \sum_{\ell=1}^{N} \tau_\ell$.

La variation du débit alloué à la source ($ACR(t)$) est cyclique. Le cycle étant défini comme le temps qui sépare les deux instants où $ACR(t)$ est croissant et égal à $\mu_N$, ces instants sont référés dans ce qui suit par $T_N^n - \sum_{\ell=1}^{N} \tau_\ell$ et $T_N^{n+1} - \sum_{\ell=1}^{N} \tau_\ell$ (voir ci-dessous la définition de $T_N^n$ et $T_N^{n+1}$ et la figure 6.2).

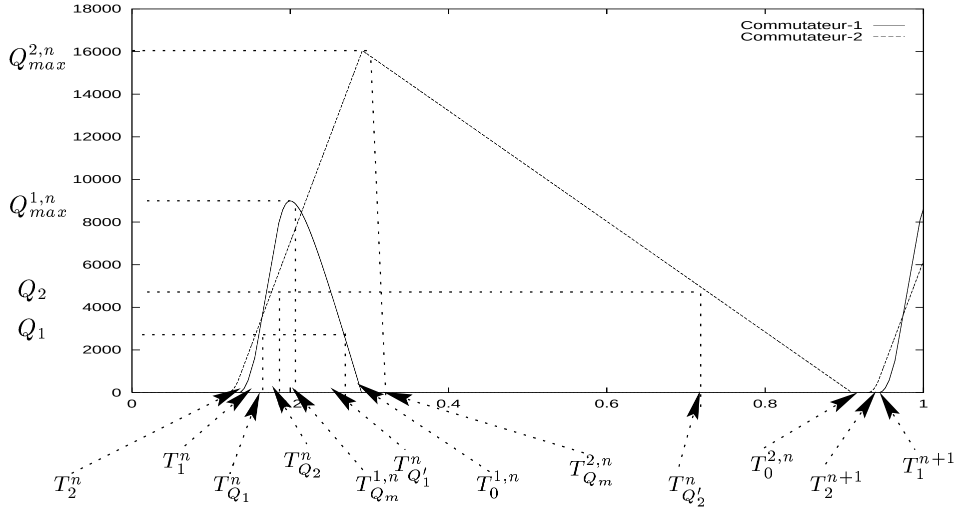



FIG. 6.2: Les tailles des files d'attente dans un modèle à deux commutateurs

Définissons (Figure 6.2) :

- $Q_i$ : Le seuil de congestion au niveau du $i$ème commutateur.
- $T_i^n \triangleq$ Instant où la file $i$ commence à se remplir, dans le $n$ème cycle (i.e. $ACR(T_i^n - \sum_{\ell=1}^{i} \tau_\ell) = \mu_i$).
- $Q_i^n \triangleq$ La taille de la file $i$ à l'instant $T_i^n$, $Q_i^0 = 0\ \forall i$.
- $Q_{max}^{i,n} \triangleq$ La taille maximale de la file $i$, dans le $n$ème cycle.
- $Q_i(t) \triangleq$ La taille de la file $i$ à l'instant $t$.

Par convention, on considère que la bande passante au niveau de la source est $PCR$ (la source ne peut pas émettre à un débit supérieur à PCR), et on définit ainsi : $ACR(T_0^n) = \mu_0 = PCR$.

On considère qu'après le premier cycle, la $N$ème file ne se vide jamais (i.e. utilisation maximale). Nous établirons les conditions pour que cette hypothèse soit vérifiée (voir équations (6.4) et (6.10)). Nous considérons également, que l'indication de congestion arrive à la source $\sum_{\ell=1}^{i} \tau_\ell$, après qu'elle soit détectée au niveau du $i$ème commutateur (i.e. le temps entre deux cellules RM et le temps de service d'une cellule ("en amont") sont négligés). En pratique, l'indication peut prendre beaucoup plus de temps de l'instant où elle a lieu jusqu'à son arrivée à la source ; ce surplus de temps est borné par le temps entre deux cellules RM, i.e. $(N_{rm} - 1)/\mu_N$ plus les temps de service dans les commutateurs en aval $(1/\mu_{i-1} + \cdot + 1/\mu_1$ où $C_i$ est le débit dans le sens "amont" du commutateur $i$, qu'on suppose très large) . Pour l'analyse transitoire, nous supposons que $ICR = \mu_N$ (il n'est pas raisonnable de négocier un ICR plus large que $\mu_N$ ; $ICR = \mu_N$ correspond alors à la rafale (burst) maximale initiale).

## 6.3 Résultats essentiels

Dans cette section, les résultats obtenus pour la phase transitoire, aussi bien que stationnaire, sont présentés. La phase transitoire nous permet d'avoir des bornes sur la taille maximale de la file d'attente $Q_{tr}^i$, dans le commutateur $i$, qui sera uniforme pour tout instant. Ceci dit, si la capacité de la file d'attente est plus large que la taille maximale calculée, il n'y aura aucune perte à aucun moment. La phase stationnaire, quant à elle, nous permettra d'avoir des bornes pour le cas où la file est initialement à l'état stationnaire. Nous obtenons ainsi, des pertes négligeables pour un temps de communication suffisamment long, pour des files d'attente de capacité plus large que les tailles maximales $Q_{st}^i$ calculées (les tailles dans ce cas sont plus petites que celles calculées pour la phase transitoire).

Nous appuyant sur des simulations, nous supposons que l'état stationnaire existe en effet, et est atteint en un temps fini. La taille maximale de file d'attente $i$ à l'état stationnaire est alors $Q_{st}^i \triangleq \overline{\lim}_{n\to\infty}(Q_{max}^{i,n})$, $i = 1, \cdots, N$.

**Lemme 6.3.1** **P1)** *Si la file $i$ n'est pas vide à l'instant $t$, alors la file $i+1$ n'est pas vide à l'instant $t + \tau_{i+1}$.*

**P2)** *Si la taille de la file $i$ excède le seuil de congestion $Q_i$, dans le nème cycle, alors la file $i-1$ sera vide à la fin de ce cycle (en fait toutes les files $l \leq i-1$ seront vides à la fin de ce cycle).*

La première propriété est triviale, puisque si la file $i$ n'est pas vide à l'instant $t$, alors le débit d'entrée à la file $i+1$ à l'instant $t + \tau_{i+1}$ est égal a $\mu_i > \mu_{i+1}$.

Pour la deuxième propriété, si la taille de la file $i$ excède le seuil $Q_i$ et la file $i-1$ est non vide, alors la taille de la file $i$ continue de croître. Par conséquent, elle décroît (à l'instant $t$) seulement si la file $i-1$ est vide (à l'instant $t - \tau_i$), ce qui devrait arriver (après un certain temps) puisque $ACR(t)$ décroît tant que la file $i$ excède le seuil $Q_i$. Du Lemme 6.3.1 (**P1**), il découle que toutes les files $l < i-1$ seront vides à la fin de ce cycle. On s'intéressera essentiellement au cas où la file $N$ est le *goulot essentiel* (voir définition 6.3.1).

**Définition 6.3.1** i*) La file $i$ est un goulot au sens strict si et seulement si la file $j$ $(j \neq i)$ reste tout le temps vide.*
ii*) La file $i$ est le goulot essentiel si et seulement si la file $j$ $(j \neq i)$ n'atteint jamais son seuil de congestion $Q_j$.*
iii*) On dira que la congestion est tout le temps due au commutateur $i$ si et seulement si la file $i$ atteint son seuil $Q_i$, pas plus tard que $\sum_{\ell=1}^{i} \tau_\ell - \sum_{\ell=1}^{j} \tau_\ell$ avant que la file $j$ $(\forall j \neq i)$ ait atteint le sien, quel que soit le cycle (la première cellule RM qui indique la congestion est mise à jour par le ième commutateur : $T_{Q_i}^n + \sum_{\ell=1}^{i} \tau_\ell \leq T_{Q_j}^n + \sum_{\ell=1}^{j} \tau_\ell$; $\forall n\ \forall j \neq i$).*

**Remarque 6.3.1** *La file une ne peut pas être un goulot au sens strict.*

**Lemme 6.3.2** *Si $Q_i^n \geq Q_i^m$ pour certains $n$ et $m$, alors $Q_{max}^{i,n} \leq Q_{max}^{i,m}, i = 1, \cdots, N$.*

**Théorème 6.3.1** *Quand la congestion est tout le temps due au $N$ème commutateur, les tailles maximales des files d'attente sont obtenues durant le premier cycle :*

$$Q_{tr}^i \triangleq \max_{n\geq 0}(Q_{max}^{i,n}) = Q_{max}^{i,0}$$

**Preuve:** Notons, tout d'abord, que $Q_i^n = 0\ \forall n$ pour $i < N$. Ceci est dû au Lemme 6.3.1 **P2**. Pour tout $n$, on a $Q_i^n \geq Q_i^0$, puisqu'on considère que la file $i$ est initialement vide. A présent, nous allons utiliser la propriété monotone suivante : Supposons que $Q_i^n \geq Q_i^m$ pour certains $n$ et $m$. Alors, le temps que prend la file $i$ pour atteindre $Q_i$, dans le $n$ème cycle, est plus court que le temps qu'elle met pour atteindre $Q_i$ dans le $m$ème cycle. Il découle, que $ACR$ dans le $n$ème cycle à l'instant $T_{Q_i}^n - \sum_{\ell=1}^{i} \tau_\ell + t$ est plus petit que le $ACR$ dans le $m$ème cycle à l'instant $T_{Q_i}^m - \sum_{\ell=1}^{i} \tau_\ell + t$ pour $t \geq 0$. La preuve suit en prenant $m = 0$ et un $n$ arbitraire.

**Remarque 6.3.2** *Dans la preuve du Théorème 6.3.1, nous avons également établi la preuve du Lemme 6.3.2.*

Soient $A_{i,n}$, $B_{i,n}$, $C_{i,n}$ définis comme :

$$A_{i,n} \triangleq \begin{cases} \sqrt{\dfrac{2(Q_i - Q_i^n)}{AIR * \mu_N}} + \dfrac{\mu_i - \mu_N}{AIR * \mu_N} & \text{si } \dfrac{\mu_{i-1} - \mu_i}{AIR * \mu_N} \geq \sqrt{\dfrac{2(Q_i - Q_i^n)}{AIR * \mu_N}} \\ \dfrac{Q_i - Q_i^n}{\mu_{i-1} - \mu_i} + \dfrac{\mu_{i-1} + \mu_i - 2\mu_N}{2 * AIR * \mu_N} & \text{ailleurs} \end{cases}$$

$$B_{i,n} \triangleq \left( 2 \sum_{\ell=1}^{N} \tau_\ell + A_{N,n} - \frac{\mu_{i-1} - \mu_N}{AIR * \mu_N} \right)^+$$

$$C_{i,n} \triangleq min \left( 2 \sum_{\ell=1}^{N} \tau_\ell + A_{N,n} - \frac{\mu_{i-1} - \mu_N}{AIR * \mu_N}, \ \frac{\mu_{i-1} - \mu_i}{AIR * \mu_N} \right)$$

où $(x)^+ \triangleq max(0, x)$. $Q_i^n$ est donnée dans le Théorème 6.3.3 et Théorème 6.3.4.

### 6.3.1 Résultats pour la phase transitoire

**Théorème 6.3.2** *Pour les commutateurs basés EFCI aussi bien que pour ceux basés ER, on a*
*i) La file i commence à se remplir la première, si et seulement si*

$$\frac{\mu_i - \mu_j}{AIR * \mu_N} \leq \sum_{\ell=i+1}^{N} \tau_\ell - \sum_{\ell=j+1}^{N} \tau_\ell \ , \ \forall j \neq i.$$

*Dans le cas particulier :* $\tau_i = \tau_1 \ \forall i$, *on a : la file i commence à se remplir la première si et seulement si*

$$\frac{\mu_i - \mu_j}{AIR * \mu_N} \leq (j - i)\, \tau_1, \quad \forall j \neq i$$

*ii) Les propriétés suivantes sont équivalentes :*

- *(ii.a) La congestion est tout le temps due au ième commutateur.*
- *(ii.b)* $\displaystyle\sum_{\ell=j+1}^{N} \tau_\ell + \sum_{\ell=1}^{i} \tau_\ell - \sum_{\ell=1}^{j} \tau_\ell - \sum_{\ell=i+1}^{N} \tau_\ell \leq A_{j,n} - A_{i,n}, \ \forall n, \quad j < N.$

*Cas particulier : pour le Nème commutateur, les points suivant sont équivalents :*

- *(ii.a) La congestion est tout le temps due au Nème commutateur.*
- *(ii.b.1)* $\displaystyle 2 \sum_{\ell=j+1}^{N} \tau_\ell \leq A_{j,n} - A_{N,n}, \ \forall n, \quad j < N,$
- *(ii.c.1)* $\displaystyle 2 \sum_{\ell=j+1}^{N} \tau_\ell \leq A_{j,0} - A_{N,0}, \quad \forall j < N.$

**Théorème 6.3.3** *Si la congestion est tout le temps due au Nème commutateur, alors on a*
*i) Pour les commutateurs basés EFCI aussi bien que pour ceux basés ER, les points suivants sont équivalents :*

- *(i.a) La file deux est un* goulot au sens strict.
- *(i.b)* $2\sum_{i=1}^{N}\tau_\ell \leq \frac{\mu_i - \mu_N}{AIR * \mu_N} - A_{N,n}\ ;\ \forall n,\ i < N.$
- *(i.c)* $2\sum_{i=1}^{N}\tau_\ell \leq \frac{\mu_{N-1} - \mu_N}{AIR * \mu_N} - A_{N,0}$

*i.1) Les points suivants sont également équivalents*

- *(i.1.a) Aucune file ne reste vide à tout moment.*
- *(i.1.b)* $2\sum_{i=1}^{N}\tau_\ell > \frac{\mu_1 - \mu_N}{AIR * \mu_N} - A_{N,0}$

*ii) Pour les commutateurs basés EFCI, si la file N n'est pas un* goulot au sens strict, *alors la taille maximale de la file d'attente, au ième commutateur ($Q^i_{tr}$ qui est obtenue dans le premier cycle, $Q^0_i = 0\ \forall i$), est donnée par*

$$\begin{aligned} Q^t_1 &= \frac{AIR * \mu_N}{2}\, {C_{1,0}}^2 + (\mu_0 - \mu_1)\, B_{1,0} + RDF * AIR * C_{1,0} \\ &\quad - \frac{RDF * \mu_1}{\mu_N} \log\left(1 + \frac{AIR * \mu_N}{\mu_1} C_{1,0}\right) \end{aligned} \tag{6.2}$$

*et pour $i > 1$, on a*

$$\begin{aligned} Q^i_{tr} &\approxeq \frac{AIR * \mu_N}{2} {C_{i,0}}^2 + (\mu_{i-1} - \mu_i)\left[B_{i,0} + \frac{RDF}{\mu_N}\log\left(1 + \frac{AIR * \mu_N}{\mu_1} C_{1,0}\right)\right. \\ &\quad \left. + \sum_{j=1}^{i-1}\left(\frac{Q^t_j}{\mu_j} + \frac{RDF(\mu_j - \mu_{j+1})}{\mu_N \mu_j}\right)\right] \end{aligned} \tag{6.3}$$

*Les bornes suivantes sont vérifiées (pour le nombre de cellules restantes dans la file N à la fin d'un cycle) :*

$$\begin{aligned} \left(Q_N - \mu_N\left(\tau + \frac{1}{2 * AIR}\right)\right)^+ \leq Q^n_N \leq &\left(Q_N + RDF\left(1 - e^{-\frac{\mu_N}{RDF}\tau}\right) - \frac{\mu_N}{2 * AIR} \times \right. \\ &\left.\left(2 * AIR * \tau + \left(1 - e^{-\frac{\mu_N}{RDF}\tau}\right)^2\right)\right)^+ \end{aligned} \tag{6.4}$$

*iii) Pour les commutateurs basés ER, la file N est le* goulot essentiel *si*

$$\sum_{\ell=1}^{i-1} \frac{Q^t_\ell}{\mu_\ell - ER_N} < \sqrt{\frac{2Q_i}{AIR * \mu_N}} - B_{i+1,0} \quad \forall i < N$$

*iv) Pour les commutateurs basés ER, si la file N n'est pas* goulot au sens strict, *alors la taille maximale de la file i ($Q^i_{tr}$) est donnée par*

$$Q^i_{tr} = \frac{AIR * \mu_N}{2}\, {C_{i,0}}^2 + (\mu_{i-1} - \mu_i)\left(\sum_{\ell=1}^{i-1}\frac{Q^t_\ell}{\mu_\ell - ER_N} + B_{i,0}\right) \tag{6.5}$$

Les preuves des deux théorèmes précédents sont données dans l'annexe B (section B.1.2).

### 6.3.2 Résultats pour la phase stationnaire

Dans ce qui suit (phase stationnaire) on se restreindra uniquement au cas où *la congestion est tout le temps due* au $N$ème commutateur. Puisqu'on considère l'état stationnaire, ceci est moins restrictif que l'hypothèse correspondante pour la phase transitoire. En effet, le comportement à l'état stationnaire est un régime périodique obtenu après une période de transition, ce qui fait que *la congestion est tout le temps due* au $N$ème commutateur si et seulement si $T^n_{Q_2} + \tau_2 \leq T^n_{Q_1}$ pour un certain $n \geq 1$. L'hypothèse ci-dessous s'applique à toutes les simulations qu'on a pu observer, dans le cas où la $N$ème file ne se vide jamais.

**Théorème 6.3.4** *Si la file $N$ n'est pas le* goulot au sens strict *et* la congestion est tout le temps due *au $N$ème commutateur, alors*
*i) Pour les commutateurs basés EFCI, la taille maximale de la file i ($Q^i_{st}$), est donnée par*

$$\begin{aligned} Q^s_1 &= Q^\infty_1 + \frac{AIR * \mu_N}{2}\, {C_{1,\infty}}^2 + (\mu_0 - \mu_1)\, B_{1,\infty} + RDF * AIR * C_{1,\infty} \\ &\quad - \frac{RDF * \mu_1}{\mu_N} \log\left(1 + \frac{AIR * \mu_N}{\mu_1} C_{1,\infty}\right) \end{aligned} \tag{6.6}$$

*et pour $i > 1$ on a*

$$\begin{aligned} Q^i_{st} &\approxeq Q^\infty_i + \frac{AIR * \mu_N}{2} {C_{i,\infty}}^2 + (\mu_{i-1} - \mu_i) \left[ B_{i,\infty} + \frac{RDF}{\mu_N} \log\left(1 + \frac{AIR * \mu_N}{\mu_1} C_{1,\infty}\right) \right. \\ &\quad \left. + \sum_{j=1}^{i-1} \left( \frac{Q^s_j}{\mu_j} + \frac{RDF(\mu_j - \mu_{j+1})}{\mu_N \mu_j} \right) \right] \end{aligned} \tag{6.7}$$

$$\text{Où} \quad Q^\infty_i = 0 \; \forall i < N \quad \text{et} \quad Q^\infty_N = \left( Q_N - \mu_N \left( \tau + \frac{1}{2 * AIR} \right) \right)^+ \tag{6.8}$$

*ii) Pour les commutateurs basés ER, la taille maximale de la file i ($Q^i_{st}$), est donnée par*

$$Q^i_{st} = Q^\infty_i + \frac{AIR * \mu_N}{2}\, {C_{i,\infty}}^2 + (\mu_{i-1} - \mu_i) \left( \sum_{\ell=1}^{i-1} \frac{Q^s_\ell}{\mu_\ell - ER_N} + B_{i,\infty} \right) \tag{6.9}$$

$$\text{Où } Q^\infty_i = 0 \; \forall i < N, \quad \text{et } Q^\infty_N = \left( Q_N - (\mu_N - ER_N) \left( \tau + \frac{\mu_N - ER_N}{2 * AIR * \mu_N} \right) \right)^+ \tag{6.10}$$

La preuve de ce théorème est donnée dans l'annexe B (section B.1.3).

## 6.4 Résultats numériques et simulations

Dans la section suivante, quelques exemples numériques sont présentés dans le but de valider les différentes hypothèses considérées dans l'approche analytique par des simulations. Les simulations présentées sont obtenues à l'aide du simulateur REAL, qu'on a modifié en introduisant un module permettant de simuler le trafic ABR. Dans ces simulations, nous avons considéré qu'une source a la capacité de transmettre des cellules l'une après l'autre (comme si chaque cellule est un paquet en soi).

Une comparaison entre les résultats obtenus pour notre modèle, et ceux obtenus sous l'hypothèse qui consiste à simplifier le réseau en un seul nœud (exactement les mêmes paramètres que ceux de notre modèle avec les $(N-1)$ premiers commutateurs effacés) est également faite. Seule la phase transitoire est considérée (premier cycle), du moment que les tailles maximales des files qui nous permettent de n'avoir aucune perte sont obtenues durant cette phase (voir Théorème 6.3.1). Nous calculons les tailles des files $(Q^i_{tr},\ i \geq 1)$ dans chaque commutateur en fonction de $\sum_{\ell=1}^{i} \tau_\ell$.

Comme on l'avait vu précédemment (Théorème 6.3.3), les tailles maximales des files aux commutateurs ne sont pas fonction du délai d'aller retour $(\tau)$. Ceci est dû au fait que les commutateurs ont la capacité d'altérer le contenu des cellules RM "en amont" (i.e. quand la congestion est détectée, les cellules RM dans la direction "en amont" sont aussi mises à jour). Cependant, à l'état stationnaire elles sont fonction et de $\tau$ et de $\sum_{\ell=1}^{i} \tau_\ell$, puisque, la file $N$ peut ne pas être vide à la fin d'un cycle ($Q^n_N > 0$, voir équations (6.7) et (6.8) ; (6.9) et (6.10)).

**Remarque 6.4.1** *Déjà mentionné dans la section 6.2, le temps entre deux cellules RM est négligé dans ce chapitre. Cela, a un effet dans le calcul des bornes sur les files d'attente, par conséquent, dans ce qui suit nous avons pris en compte ce temps (borne supérieure) ; ainsi, la quantité $(N_{rm} - 1)/\mu_N$ est ajoutée à $A_{i,0}$, $i \geq 0$ quand le calcul des bornes sur les files d'attente (ana-qi dans les figures ci-dessous) est fait. Ceci est justifié par le fait, que quand la congestion est détectée par le ième commutateur, l'information de congestion arrive à la source au plus tard $\sum_{\ell=1}^{i} \tau_\ell + (N_{rm} - 1)/\mu_N$ après.*

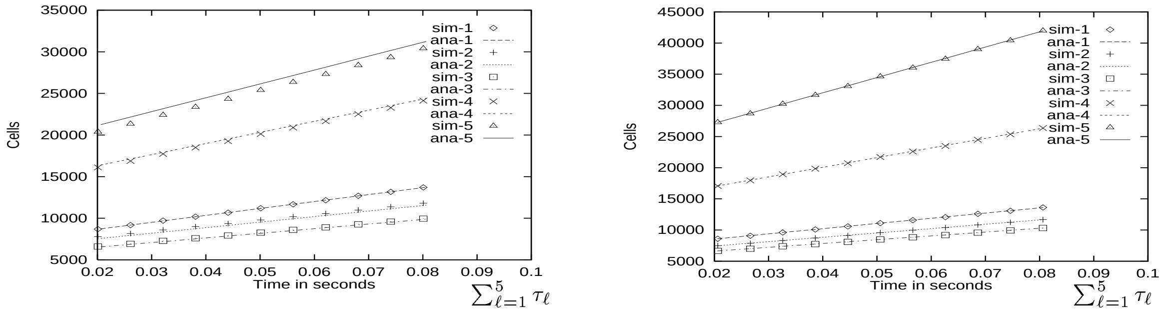
Dans toutes les simulations qui suivent, nous avons considéré les paramètres suivants : $PCR = 134.78\ Mbps$ (temps de transmission d'une cellule est de 3 $\mu sec$), $MCR = 0.0\ Mbps$ $AIR = 0.1\ Mbps \approxeq 250\ cells/sec$, $RDF = 512$ et $ERF = 0.8$. Nous allons prendre des seuils différents pour chaque file, dans le but de montrer l'effet des commutateurs non homogènes sur la taille maximale de la file d'attente au niveau du $N$ème commutateur.

Dans le premier exemple, nous avons considéré cinq commutateurs sur le lien, avec les seuils de congestion suivants : $Q_1 = 20000$, $Q_2 = 15000$, $Q_3 = 10000$, $Q_4 = 10000$ et $Q_5 = 5000$. Les bandes passantes disponibles au niveau des commutateurs sont respectivement : $\mu_1 = 106\ Mbps = 250000\ cells/sec$, $\mu_2 = 84.8\ Mbps = 200000\ cells/sec$, $\mu_3 = 70.66\ Mbps = 166666.67\ cells$ sec, $\mu_4 = 42.4\ Mbps = 100000\ cells/sec$ et $\mu_5 = 21.2\ Mbps = 50000\ cells/sec$, ce qui nous donne, alors $A_{1,0} = 0.2593$, $A_{2,0} = 0.314$, $A_{3,0} = 0.3106$, $A_{4,0} = 0.1566$ et $A_{5,0} = 0.102$. Du Théorème 6.3.2 on a : *la congestion est tout le temps due* au commutateur $N$ si et seulement si $2\sum_{\ell=j+1}^{5} \tau_\ell \leq A_{j,0} - A_{N,0}$, $\forall j < 5$, qui est équivalent à

$$\tau_5 \leq 0.0273 sec\,, \quad \tau_4 \leq 0.1043 - \tau_5 sec\,, \quad \tau_3 \leq 0.106 - \tau_4 - \tau_5 sec\,, \quad \tau_2 \leq 0.078667 - \tau_3 - \tau_4 - \tau_5 sec$$

D'autre part, on doit avoir (Théorème 6.3.3 *(i.1)*) $\sum_{\ell=1}^{5} \tau_\ell > -0.086$ ce qui signifie que toutes les files se construisent. Nous avons choisi $\tau_i$ $i = 2 \cdots 5$ : $\tau_5 = 2\ ms$, $\tau_4 = 2\ ms$, $\tau_3 = 2\ ms$ et $\tau_2 = 1 ms$ comme valeurs initiales. En comparant les résultats analytiques aux résultats de simulation (en fonction du délai d'aller retour), on augmente uniquement $\tau_1$ (de 3 $ms$ à 30 $ms$ avec un pas de 3 $ms$, voir figure 6.3), puisque les tailles des files d'attente sont fonction de $\sum_{\ell=1}^{N} \tau_\ell$. $\tau_6$ est égal à 5 $ms$, il n'a aucune influence, pour le cas transitoire, du moment que les cellules RM "en amont" sont mises à jour.

Les Figures 6.3 illustrent la taille maximale de chaque file d'attente en fonction du délai d'aller retour $2\sum_{i=1}^{N} \tau_i$ pour l'exemple à cinq commutateurs ci-dessus, pour les commutateurs basés EFCI (a) aussi bien que ceux basés ER (b). On peut constater que les résultats analytiques (*ana*) sont très proches de ceux obtenus par simulation (*sim*), et l'erreur est presque




(a) Commutateurs basés EFCI (b) Commutateurs basés ER

FIG. 6.3: Tailles des files d'attente en fonction du délai d'aller retour

négligeable essentiellement pour les commutateur basés ER.

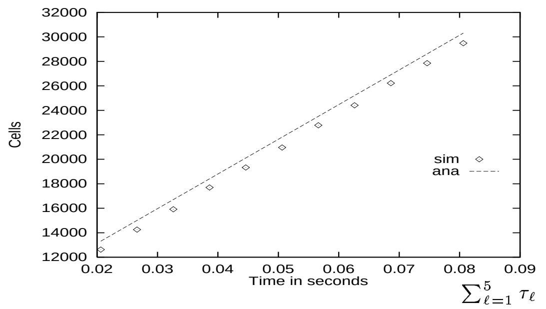
En général, il est connu que les tailles des files d'attente dans les commutateurs basés EFCI sont plus importantes que celles constatées dans les commutateurs basés ER. Cependant, dans cet exemple on constate l'inverse. C'est première fausse conclusion tirée quant on cinsidère un modèle à un seul goulot d'étranglement. Ceci, est simplement dû au fait que lorsque une congestion est détectée dans les commutateurs basés ER, la source réduit sont débit à un débit constant ($ER_5 = 0.8\mu_5 = 40000 >> 0$). Ce débit peut être très grand par rapport à un débit nul, et la différence $\mu_{i-1} - ER_5$ n'est pas aussi grande, en particulier pour $i = 4, 5$ dans notre exemple. Par conséquent, les files se vident très lentement (voir équation 6.5) contrairement aux commutateurs basés EFCI où le débit de la source atteint un débit nul (ou MCR) en un instant assez court, après que la congestion soit détectée.

Pour avoir la taille maximale de la file d'attente dans le modèle simplifié, on peut considérer le même modèle décrit dans ce chapitre (même si on avait supposé que le nombre de commutateur est supérieur à un, $N > 1$), et utiliser les résultats des Théorèmes précédents. En effet, en considérant le modèle décrit dans la section 6.2 avec $\mu_0 = \mu_1 = \cdots = \mu_{N-1} = PCR$, on obtient ainsi les formules nous permettant de calculer la taille maximale de la file d'attente dans le modèle à une seule file.

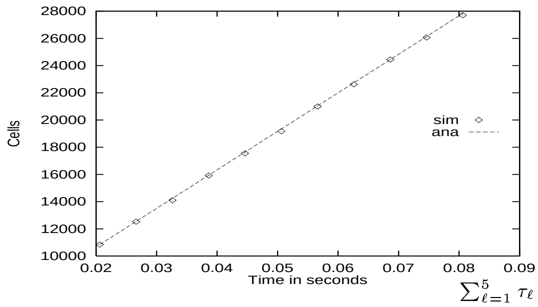
La figure 6.4 illustre la différence entre les résultats analytiques obtenus en suivant la procédure décrite précédemment, et les résultats de simulations pour le modèle simplifié. On peut voir que dans cet exemple par contre, la taille de la file dans le commutateur basé EFCI est plus importante que dans le commutateur basé ER. On peut constater ainsi, que le nombre de commutateur sur le lien agit négativement sur les performances des commutateurs basés ER.

#### 6.4.0.1 Tailles maximales des files d'attente dans notre modèle et le modèle à une seule file

Dans le but de montrer la différence entre les tailles maximales des files d'attente dans un réseau "réel", et celles obtenues en simplifiant le réseau en un seul commutateur (une seule file qui correspond au commutateur le plus lent), nous avons considéré un autre exemple. Dans cet exemple, seuls deux commutateurs sont pris en considération. Les bandes passantes disponibles sont respectivement $\mu_1 = 40.435791\ Mbps$, $\mu_2 = 4.043\ Mbps$ et les seuils de congestion $Q_1 = Q_2 = 5000\ cellules$. Du Théorème 6.3.2 et Théorème 6.3.3 on obtient les résultats suivants :




(a) Commutateurs basés EFCI




(b) Commutateurs basés ER

FIG. 6.4: Taille maximale de la file d'attente en fonction du délai d'aller retour pour le modèle à une seule file

Pour les commutateurs basés EFCI, aussi bien que ceux basés ER, la file deux commence à se remplir avant la file une si et seulement si $\tau_2 < 0.036$ *sec* et *la congestion est tout le temps due* au second commutateur si et seulement si $\tau_2 \leq 0.0128$ *sec*. La file deux est *goulot au sens strict* si et seulement si $2(\tau_1 + \tau_2) \leq -0.037556$ *sec*. Ceci signifie que la a file deux ne peut pas être le *goulot au sens strict*. Cependant, elle est le *goulot essentiel* pour les commutateurs basés EFCI (*resp.* basés ER) si $\tau_1 + \tau_2 < 0.002165$ (*resp.* $\tau_1 + \tau_2 < 0.012845$).

La différence dans la taille maximale des files d'attente entre le modèle simplifié et le notre (la seconde file) est de l'ordre du seuil de congestion $Q_1 = 5000$, ce qui est relativement énorme quand on considère des tailles de 18000 cellules. Ceci est illustré dans la Figure 6.5 (diff-5000), où la différence entre la taille maximale de la file d'attente au niveau du second commutateur et celle dans le modèle simplifié (à un seul commutateur) est donnée en fonction du délai d'aller retour $2(\tau_1 + \tau_2)$. Deux valeurs différentes du seuil de congestion au niveau du premier commutateur $Q_1$ sont considérées (5000 and 15000) ($Q_2 = 5000$ *cellules* dans tous les exemples de cette section). La taille maximale obtenue dans le modèle simplifié est également retracée (single dans la Figure 6.5).

Dans cet exemple seul $\tau_1 + \tau_2$ varie. Nous avons considéré comme valeurs initiales : $\tau_1 = 0.263$ $ms$ et $\tau_2 = 0.35$ $ms$. Chaque fois, la même valeur est ajoutée à la fois à $\tau_1$ et $\tau_2$, jusqu'à ce que $\tau_2 = 0.012$ $ms$. Notons, qu'on obtient les mêmes résultats si on fait varier uniquement $\tau_1$ ou $\tau_2$.

A partir des formules de la section 6.3, on peut constater que $Q_2^t$ augmente linéairement en fonction de la taille $Q_1^t$. $Q_1^t$ est en particulier fonction de $\mu_1$ ; plus $\mu 1$ est proche de $\mu_2$, plus grande est la taille de la file d'attente au niveau du premier commutateur. Par conséquent, en diminuant $\mu_1$, on peut obtenir des cas où la file d'attente dans le deuxième commutateur est significativement plus large que celle qu'on obtient dans un modèle à un seul commutateur (simplifié), pendant que $\mu_2$ est le même pour les deux modèles. Cependant, les conditions pour que la congestion soit tout le temps due au second commutateur changent. Ainsi, une valeur adéquate de $Q_1$ doit être choisie pour les satisfaire quand $\mu_1$ est diminué. En fait, on souhaiterait avoir les mêmes propriétés que pour $\mu_1 = 40.43$ $Mbps$, et montrer comment la diminution de $\mu_1$ et l'augmentation de $Q1$ affectent la taille maximale de la file d'attente au niveau du second commutateur. Notons également que de larges tailles de file d'attente au second commutateur sont obtenues quand *la congestion est tout le temps due* à celui ci.

Comme illustration, considérons l'exemple précédent avec $\mu_1 = 26.957194$ $Mbps$. Notre objectif est de déterminer $Q_1$ tel que *la congestion est tout le temps due* au second commutateur si et seulement si $\tau_2 \leq 0.0128$ *sec*. On a

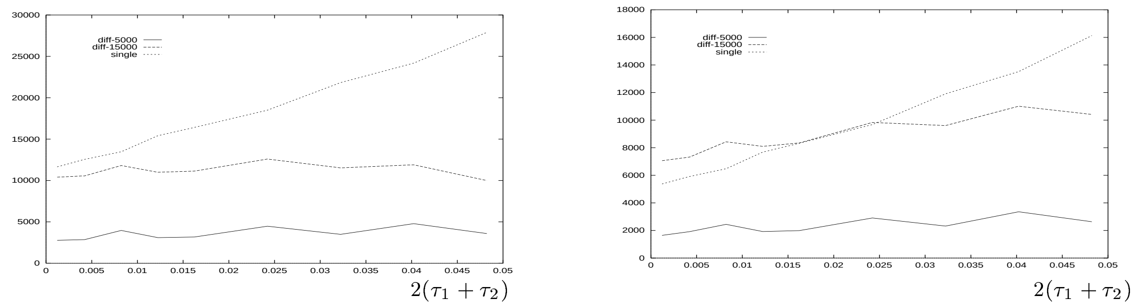
$$\frac{\mu 1 - \mu_2}{AIR * \mu_2} = 0.0226 \leq \sqrt{\frac{2Q_2}{AIR * \mu_2}} = 0.0632 \Longrightarrow A_{2,0} = 0.07693 \ sec.$$

Supposons $\dfrac{PCR - \mu_1}{AIR * \mu_2} \leq \sqrt{\dfrac{2Q_1}{AIR * \mu_2}}$. Alors $Q_1 \leq 15308.88$, et $A_{1,0} = \dfrac{1}{500\sqrt{5}}\sqrt{Q_1}$.

Du Théorème 6.3.2 on a : *la congestion est tout le temps due* au second commutateur si et seulement si

$$0.0296 \leq \frac{1}{500\sqrt{5}}\sqrt{Q_1} - 0.07693 \quad \text{i.e.} \quad Q_1 \geq 14185.801$$

La Figure 6.5 montre la différence dans les tailles de file d'attente entre notre modèle et le




(a) Commutateurs basés EFCI (b) Commutateurs basés ER

FIG. 6.5: Différence en taille maximale de file d'attente entre notre modèle et le modèle simplifié (une seule file)

modèle simplifié (diff-5000, diff-15000), ainsi que la taille maximale obtenue dans le modèle simplifié (single dans la figure). La différence dans les tailles est encore plus grande pour le deuxième cas (diff-15000) quand $Q_1 = 15000$ *cellules* et $\mu_1 = 26.957194$ *Mbps* sont considérés. Dans certains cas, spécialement pour les commutateurs basés ER, elle est égale à la taille maximale obtenue dans le modèle simplifié (c'est à dire que la taille maximale de la file d'attente au niveau du second commutateur dans notre modèle est deux fois plus grande que la taille maximale dans le modèle à un seul commutateur).

La décroissance dans la différence qu'on peut constater dans la figure 6.5 (diff-15000) après l'instant $(\tau_1 + \tau_2) = 0.048$ *sec* est due au fait que l'ACR atteint PCR dans le modèle à deux commutateurs pendant qu'il ne l'atteint pas dans le modèle à un seul commutateur.

Les résultats numériques présentés dans cette section concernent seulement la phase transitoire, cependant la même analyse peut être faite pour la phase stationnaire et les mêmes résultats qualitatifs (les tailles des files d'attente dans ce cas sont plus petites dans les deux modèles) peuvent être obtenus.

## 6.5 Remarques et Conclusions

Généralement, une fraction de la capacité maximale des files d'attente est recommandée pour les seuils de congestion. La situation où les commutateurs ont des seuils différents peut ainsi être courante en pratique. Ceci implique en particulier, la situation générale qu'on a étudiée, avec des seuils différents dans des commutateurs différents.

Comme montré dans l'exemple précédent, il est possible de choisir des paramètres ($Q_i$, $\tau_i$ et $\mu_i$) tels que la taille maximale de la file d'attente au $N$ème (second dans l'exemple précédent) commutateur soit deux fois, voire même plus, la taille maximale dans le cas d'un unique commutateur (modèle simplifié). Par conséquent, le modèle qui consiste à simplifier le réseau en un seul commutateur ne permet pas de déterminer les "vraies" bornes pour les tailles maximales des files d'attente au niveau des commutateurs. Ces tailles maximales dépendent en définitive du nombre de commutateurs traversés par la connexion.

Ceci implique que les capacités des files d'attente requises par des connexions traversant plusieurs commutateurs peuvent être extrêmement larges. C'est un important inconvénient de l'*Indication de Congestion* (par bit unique) révélé par notre analyse.

En présence de plusieurs commutateurs sur un circuit virtuel, sur lequel le contrôle de flux est effectué, les tailles maximales des files obtenues dans le cas de EPRCA (algorithme basé ER) peuvent être plus larges que celles obtenues sous les commutateurs basés EFCI. La dynamique des files d'attente s'est révélée également être beaucoup plus complexe que dans le cas d'un seul nœud. Un comportement non intuitif a été constaté, dans lequel la congestion a lieu dans un commutateur qui n'est apparemment pas le plus lent sur le circuit. Ce fait, avec les bornes analytiques que nous avons obtenues, suggèrent d'être très prudent quant à l'utilisation des approximations basées sur un seul goulot d'étranglement (correspondant au commutateur le plus lent)

Nous avons considéré le cas où le $i$ème commutateur est plus rapide que $i+1$ème, puisque dans le cas contraire, le lien le plus rapide peut être ignoré : il aura un débit d'entrée plus petit que le débit de sortie, et sera par conséquent toujours vide. Dans notre cas cependant, les points suivants peuvent avoir lieu

– La file dans le nœud le plus rapide peut commencer à se remplir avant celle dans le nœud le plus lent (Théorème 6.3.2 (i)),
– Quand l'empilement des cellules se fait dans deux nœuds consécutifs, le débit auquel la taille de la file augmente peut être *plus large dans le nœud le plus rapide que dans le plus lent*!. On peut constater ce comportement dans la figure 6.3, où la taille de la file d'attente dans le commutateur 3 est plus petite que celle dans le commutateur 2 (ou 1). La raison est que le débit de transmission peut augmenter rapidement à l'entrée du commutateur rapide, pendant que le débit d'entrée dans le commutateur le plus lent peut être considérablement petit du moment que ce débit d'entrée est généré par le débit de sortie du commutateur précédent (le plus rapide).

  Le débit d'entrée dans le commutateur le plus lent est alors borné par le débit de sortie du commutateur le plus rapide, au moment où le débit d'entrée au commutateur le plus rapide peut seulement être borné par le mécanisme de contrôle lui même (premier commutateur par exemple), et peut être beaucoup plus grand. On obtient ainsi, un phénomène où le commutateur le plus rapide protège le plus lent.
– Un autre phénomène important constaté concerne la congestion dans le commutateur le plus lent (nœud $N$). Celle ci peut prendre fin longtemps après que la source reçoive des indications de congestion.

  En fait, la file d'attente dans le nœud le plus lent continue d'*augmenter* après que la congestion ait été signalée, jusqu'à ce que les files dans les commutateurs précédents soient vides (voir Figure 6.2 et Lemme 6.3.1 **P2**). Ce phénomène n'est pas constaté dans le modèle à un seul commutateur.

Nous avons présenté une analyse, et pour la phase transitoire, et pour la phase stationnaire. Dans la phase transitoire, nous avons calculé des bornes supérieures sur les tailles maximales des files d'attente qui sont uniformes pour tout instant. En utilisant ces formules pour désigner les paramètres de contrôle, on peut garantir qu'il n'y ait aucune perte. Cette analyse peut paraître conservative, si les pertes dans la phase transitoire peuvent être tolérées ; à ce moment, les bornes les plus optimistes obtenues pour l'état stationnaire peuvent être utilisées.

# Chapitre 7

# Analyse de stabilité des algorithmes pour le contrôle de congestion dans l'ABR

Dans le chapitre précédent, la méthode de calcul directe a été utilisée pour calculer des bornes sur les tailles des files d'attente pour les algorithmes basés EFCI, et EPRCA. Cette méthode est très difficile d'usage (à cause du délai d'aller retour), pour d'autres algorithmes, en particulier pour les algorithmes "complètement" basés ER, ou basés sur les tailles exactes ou relatives des files d'attente aux commutateurs intermédiaires.

Dans [15], ont été considérés des algorithmes où la source envoie à un débit de la forme $x_{n+1} = \max(x_n + \alpha(\hat{\mu}_n - x_n) - \beta(\hat{q}_n - Q))$, où $\hat{\mu}$ et $\hat{q}$ sont les estimations respectives de la bande passante disponible et de la taille de la file d'attente. $\alpha$ et $\beta$ deux réels positifs. Il a été montré que le "pure-rate matching" n'est pas stable ($\beta = 0$). Le système par contre est stable pour $0 < \beta < \alpha < 2 + \beta/2 < 4$, et ne l'est pas si la relation n'est pas satisfaite (bien que la taille de la file d'attente soit bornée, le système oscille en permanence). Ainsi, si la taille de la file d'attente n'est pas utilisée comme paramètre de contrôle, alors le système n'est pas stable pour une utilisation maximale de la bande passante. En plus du délai d'aller retour qui n'a pas été pris en compte dans ce travail, ce genre d'algorithmes comme on le verra dans le prochain chapitre, ne permet malheureusement pas une utilisation maximale de la bande passante, pour certaines configurations du réseau.

Dans [9], nous avons considéré une version modifiée de l'algorithme ci-dessus, $x_{n+1} = \alpha x_n + \beta\hat{\mu}_n - \epsilon$, où $0 < \alpha < 1$, $\beta = 1 - \alpha$ et le drift $\epsilon > 0$ (Full matching with drift), ou $0 < \alpha < 1$, $\beta < 1 - \alpha$ et le drift $\epsilon \geq 0$ arbitraire (Partial matching. $\epsilon$ peut être nul). Nous avons montré en tenant compte du délai, que le système est stable. Par conséquent, les algorithmes ER sont stables pour une utilisation inférieure à 100% de la bande passante.

Dans le présent chapitre, la méthode de stabilité des solutions aux équations différentielles (à retard) est utilisée. Cette méthode nous permet de vérifier si un algorithme est stable ou ne l'est pas en prenant compte du délai d'aller retour (du retard). La stabilité dans ce cas peut avoir deux sens :

- Le premier, est le débordement de la file d'attente. En effet, on verra par la suite que pour des paramètres bien définis (paramètres fixes : délai d'aller retour ou capacité maximale de la file d'attente), la file d'attente dans certains algorithmes (Point Fixe) déborde et les pertes ne peuvent être empêchées, sans pour autant qu'elle converge vers l'infini.

- Le deuxième, sont les oscillations de la taille de la file d'attente. Il est souhaitable que la taille de la file d'attente soit le plus proche possible d'un certain seuil de congestion, cela permettrait en particulier d'avoir des délais d'aller retour dont la variance est minime, et d'espérer ainsi utiliser le service ABR pour la vidéo sur demande par exemple.

Ainsi, les performances des algorithmes les plus connus ("intéressant") sont étudiées. Nous montrons, en particulier, que pour de larges bandes passantes, il peut toujours y avoir des pertes dans le cas des algorithmes *ERAQLES* et le *Point Fixe*. Les domaines de stabilité sont calculés pour les derniers ainsi que pour l'algorithme *ERICA+* et des exemples d'instabilité sont donnés.

Ce chapitre est structuré comme suit : Dans la section 7.1 nous énoncerons les différents outils mathématiques utilisés le long du travail (la définition de la stabilité et quelques théorèmes liés à la stabilité). Les algorithmes basés-EFCI ainsi que ceux basés sur la taille de la file d'attente (Point Fixe) sont présentés dans la section 7.2. Dans la section 7.3, ERAQLES et ERICA sont, brièvement décrits et analysés. Finalement, nous concluons par quelques remarques dans la section 7.4.

## 7.1 Définitions

Dans ce qui suit nous utiliserons la théorie de stabilité des solutions aux équations différentielles à retard (aux arguments retardés). L'intérêt de cette théorie est qu'en l'absence de solutions exactes à la quasi totalité de ce genre d'équations, elle permet cependant d'avoir un aperçu sur le comportement de ces solutions. Cette méthode est souvent efficace dans l'analyse de stabilité des algorithmes de contrôle de flux (basé sur le débit : rate-based) pour le service ABR (*ERICA+* [59], *ERAQLES* [83], Fixed-Point [69], ...etc).

Nous commençons par énoncer la définition de la *stabilité d'une solution* (**stabilité locale**). Nous illustrerons ceci avec l'équation différentielle à retard suivante

$$\dot{x}\ (t) = f(x(t), x(t-\tau), t), \qquad \dot{x}\ (t) \triangleq \frac{dx(t)}{dt}. \tag{7.1}$$

**Définition 7.1.1** *Soit $w(t)$ une fonction, continue pour $t > 0$, satisfaisant l'équation (7.1) pour $t > \tau$. Cette solution est dite* **stable** *quand $t \to \infty$ si, étant donnés deux nombres positifs $t_0$ et $\epsilon$, il existe un nombre positif $\delta$ correspondant, tel que toute solution continue $x(t)$ de l'équation (7.1) qui satisfait*

$$\max_{t_0 \le t \le t_0 + \tau} |x(t) - w(t)| \le \delta \tag{7.2}$$

*satisfait aussi*

$$\max_{t_0 \le t} |x(t) - w(t)| \le \epsilon \tag{7.3}$$

*La solution est dite* **uniformément stable** *si, étant donné $\epsilon$, il existe un $\delta$ tel que pour tout $t_0 > 0$ et toute solution $x(t)$ qui satisfait (7.2), satisfait aussi (7.3).*

*La solution $w(t)$ est dite* **asymptotiquement stable** *si*
*(a) elle est stable ;*

*(b) pour chaque $t_0 \geq 0$ il existe un $\delta$ tel que toute solution $x(t)$ qui satisfait (7.2), satisfait également*

$$\lim_{t\to\infty} [x(t) - w(t)] = 0. \tag{7.4}$$

**Théorème 7.1.1** *Considérons l'équation différentielle suivante*

$$\frac{dx(t)}{dt} + \alpha\frac{dx(t-\tau)}{dt} + \beta x(t) + \gamma x(t-\tau) = 0$$

*Supposons que $|\alpha| \neq 1$, alors les points suivants sont vrais :*

1. *Si $|\alpha| > 1$, alors $x(t)$ est instable pour tout délai positif $\tau$.*
2. *Si $|\alpha| < 1$, $\gamma^2 < \beta^2$, ou $\gamma = \beta \neq 0$, alors augmenter $\tau$ ne change pas la stabilité de $x(t)$.*
3. *Si $|\alpha| < 1$, $\gamma^2 > \beta^2$, et*
   **(i)** *$\gamma + \beta < 0$, alors $x(t)$ est instable pour tout délai positif $\tau$ ;*
   **(ii)** *$\gamma + \beta > 0$, alors $x(t)$ est uniformément asymptotiquement stable quand $\tau < \tau_0$ et instable quand $\tau > \tau_0$, où $\tau_0 = \theta/w$, et*

$$\begin{aligned} w &= \left(\left(\gamma^2 - \beta^2\right)\left(1 - \alpha^2\right)^{-1}\right)^{1/2}, \\ \theta &= arccot\left(-\frac{\alpha w^2 + \beta\gamma}{w\left(\gamma - \beta\alpha\right)}\right) \end{aligned}$$

Dans ce qui suit nous énonçons la généralisation du théorème de Poincaré-Lyapunov [20] aux équations différentielles à retard.

**Théorème 7.1.2** *Considérons l'équation non linéaire suivante*

$$a_0 u'(t) + b_0 u(t) + b_1 u(t-\tau) = f(u(t), u(t-\tau)), \quad t > \tau; \quad a_0 \neq 0,\ b_0,\ b_1 \text{ constantes} \tag{7.5}$$

*avec les conditions initiales*

$$u(t) = g(t), \qquad 0 \leq t \leq \tau. \tag{7.6}$$

*Supposons que*
*(a) Toute solution continue de la partie linéaire*

$$a_0 u'(t) + b_0 u(t) + b_1 u(t-\tau) = 0; \tag{7.7}$$

*approche zéro quand $t \to \infty$ ;*
*(b) $f(u, v)$ est une fonction continue de $u$ et $v$ au voisinage de l'origine $|u| + |v| \leq c_1$ ; et*
*(c)*

$$\lim_{|u|+|v|\to 0} \frac{|f(u,v)|}{|u| + |v|} = 0. \tag{7.8}$$

*Alors, si $\max_{0\leq t\leq\tau} |g(t)|$ est suffisamment petit (selon $c_1, a_0, b_0$ et $b_1$), toute solution de l'équation non linéaire (7.5), peut être prolongée par continuité sur l'intervalle $0 \leq t < \infty$, et satisfait*

$$\lim_{t\to\infty} |u(t)| = 0. \tag{7.9}$$

Les preuves des théorèmes précédents peuvent être trouvées dans [71] *resp.* [20] (Théorème 3.2, pp. 82 *resp.* Théorème 11.2, pp. 336).

**Remarque 7.1.1** *Dans tout le reste du chapitre présent, nous parlerons de la stabilité de la solution-zéro. Nous rencontrerons en effet des équations où $x(t) = 0$ est solution triviale (équations différentielles homogènes), nous analyserons alors la stabilité d'une telle solution. Si la solution-zéro est uniformément asymptotiquement stable, cela veut dire que toute solution converge vers zéro ($\lim_{t\to\infty} x(t) = 0$).*

*Nous allons ignorer le délai entre les cellules RM, et considérer que le débit $ACR(t)$ est continuellement mis à jour. Cette hypothèse a un effet très négligeable sur la précision dans le calcul des zones de stabilité, puisque non seulement le délai entre deux cellules RM est très petit, mais aussi $ACR(t)$ est constant entre deux cellules RM.*

### 7.1.1 Modèle pour l'approche analytique

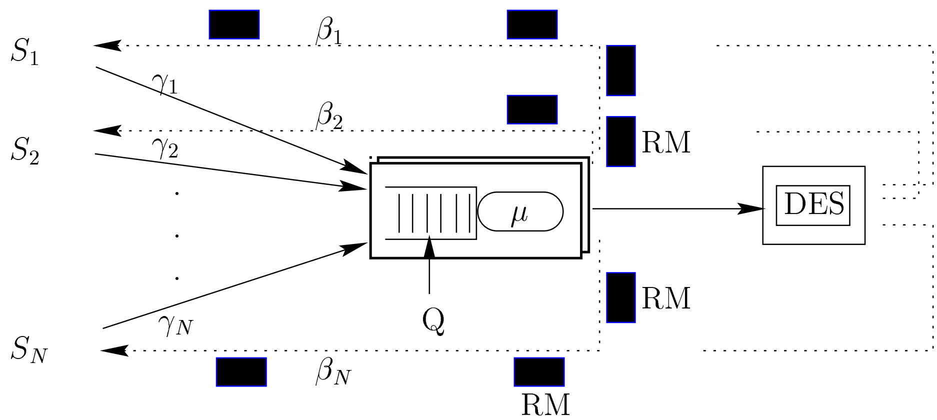



FIG. 7.1: Modèle à plusieurs sources partageant le même goulot

Pour notre approche analytique, nous considérons plusieurs sources équivalentes transmettant vers une même destination. Les sources partagent un goulot d'étranglement de capacité $\mu$, et d'une file d'attente de capacité maximale de $B$ cellules (voir figure 7.1).

Soient :
$\beta_i \triangleq$ Le temps nécessaire à une cellule pour aller du goulot (file) vers la source $i$.
$\gamma_i \triangleq$ Le temps nécessaire à une cellule pour aller de la source $i$ vers la file.
$\tau_i \triangleq \beta_i + \gamma_i$.
$y(t) \triangleq$ La taille de la file à l'instant $t$.
$x_i(t) \triangleq$ Le débit de la source $i$ à l'instant $t$. $x(t)$ le débit total de toutes les sources.
$\bar{y} \triangleq$ La taille de la file quand le système se stabilise.
$\dot{x}\ (t) \triangleq \frac{dx(t)}{dt}$, et $\ddot{x}\ (t) \triangleq \frac{d^2x(t)}{dt^2}$.

**Remarque 7.1.2** *Dans tout le reste du chapitre, nous allons considérer la stabilité des algorithmes dans un système où seules des sources équivalentes sont considérées, bien que la modélisation soit donnée pour le cas général. Ceci nous permettra d'avoir une idée sur la zone de stabilité d'un algorithme donné. En effet,* généralement *si dans un système où des sources équivalentes sont considérées, un algorithme donné est instable pour tout $\tau > \tau_0$* resp. *stable* *pour tout* $\tau < \tau_1$ *; alors dans un système où des sources non équivalentes existent, l'algorithme en question est instable si* $\min_{i\geq 1}(\tau_i) > \tau_0$ resp. *stable si* $\max_{i\geq 1}(\tau_i) < \tau_1$.

## 7.2 Algorithmes basés EFCI et basés taille des files d'attente

Plusieurs algorithmes basés sur la notification de congestion ont été proposés, comme PRCA (voir [28, 85, 86, 97]). Un grand nombre d'entre eux translatent la congestion à l'occupation des files d'attente intermédiaires, de telle sorte que la congestion est signalée selon que la taille de la file est plus grande ou plus petite qu'un certain seuil donné, appelé *seuil de congestion*. Les approches théoriques dans l'étude de ce genre d'algorithmes, utilisent souvent un modèle simplifiant tout le réseau en un seul nœud.

Comme on l'a vu dans le chapitre précédent (voir également [11, 8]), les tailles maximales des files d'attente obtenues en utilisant des algorithmes basés EFCI dépendent fortement du nombre de commutateurs traversés par une connexion. Elles peuvent être beaucoup plus importantes dans le cas de plusieurs connexions sur le même chemin. Ce phénomène est essentiellement du au fait que la notion de congestion diffère d'un commutateur à un autre, selon le seuil de congestion de chacun, et à la nature binaire de l'information sur la congestion. Aussi, il est clair qu'une information continue sur l'occupation des files d'attente permettrait de pallier aux inconvénients de l'information binaire.

S'appuyant sur cette dernière idée, d'autres algorithmes également basés sur l'occupation des files d'attente ont été présentés (Ex. donné dans [51]). L'information émise par le commutateur vers les sources est la taille de la file d'attente ABR aux instants où les cellules RM "en amont" sont reçues. Les conditions de stabilité et de convergence sont également étudiées dans ces articles. Notons que les tailles maximales des files d'attente dans de tels algorithmes dépendent souvent du nombre de sources traversant les commutateurs, et parfois de la bande passante disponible pour le service ABR.

L'inconvénient majeur de ces algorithmes est la difficulté du choix des différents paramètres, puisque ces schémas sont de la forme $x(t) = f(x(t), y_1(t-\beta_1), \cdots, y_n(t-\beta_n))$, où $x(t)$ est le débit de la source à l'instant $t$ et $y_i(t-\beta_i)$ est la taille de la file d'attente au commutateur $i$ distant de $\beta_i$ de la source. Ainsi, dans un environnement où la bande passante pour le service ABR est d'une grande variance, ces algorithmes perdent d'une manière substantielle de leurs performances, comme on le verra dans ce qui suit.

Dans [69], un schéma assez simple appelé algorithme du *Point Fixe*, basé sur la variation de l'occupation de la file d'attente est proposé. Le schéma montre des traits intéressants, notamment un débit assez lisse (smooth) aux sources. Cependant, ni l'analyse de stabilité ni une évaluation par des simulations variées n'ont été faites. Dans ce qui suit, nous présentons cet algorithme comme un exemple des algorithmes basés sur les tailles des files d'attente, et nous analyserons ensuite sa stabilité. Nous montrons que pour certaines bandes passantes (très petites), le schéma n'est pas stable quel que soit le délai d'aller retour.

### 7.2.1 Algorithme du point fixe

Cet algorithme est essentiellement basé sur les tailles des files d'attente aux commutateurs intermédiaires. Plus précisément, il est basé sur le degré de saturation des commutateurs (i.e. la fraction de la taille de la file sur un certain seuil, qui peut être la capacité maximale de la file). Le but est de garder la taille de la file proche du seuil de congestion, en variant le débit de la source d'une manière inversement proportionnelle à l'occupation de la file.

L'amélioration des performances par cet algorithme, par rapport à ceux basés sur l'augmentation linéaire et la réduction exponentielle du débit, à savoir les algorithmes basés EFCI, n'est plus à démontrer. En effet, il a la même complexité que ces derniers, tout en permettant aux différents utilisateurs d'avoir la même bande passante (équitable).

L'algorithme du *point fixe* fonctionne comme suit : les cellules RM sont envoyées suivant les recommandations de l'*ATM Forum* [1] (voir chapitre 5). La source peut envoyer des données à un débit n'excédant pas $PCR_i$. Au début de la transmission, la source émet au débit $ICR_i$. Dans notre analyse, nous n'avons pas pris en compte les conditions initiales (elles ont un effet tout à fait négligeable). Le débit de la source peut augmenter ou diminuer selon l'information se trouvant dans les cellules RM "en amont" arrivant à la source ($TQ \triangleq y(t)/Q$). Chaque fois qu'une cellule RM est reçue par la source, celle-ci ajuste son débit comme suit :

$$ACR := ACR + AIR * N_{rm} \left(1 - \frac{ACR}{PCR}\right) - TQ * ACR \tag{7.10}$$

où $TQ$ est un champ dans la cellule RM reçue, mis à jour par les commutateurs.
Quand une cellule RM "en amont" est reçue par un commutateur, le champ $TQ$ est mis au maximum entre la valeur $TQ$ se trouvant dans la cellule, et la taille de la file au niveau du commutateur divisée par le seuil de congestion $Q_H$ (qui peut être la capacité maximale de la file) :

$$TQ := \max\left(TQ, \frac{\text{taille de la file } y(t)}{\text{seuil de congestion } Q_H}\right) \tag{7.11}$$

### Approche analytique

Notre but est d'analyser cet algorithme et de trouver ces insuffisances dans le cas où elles existent. Nous utilisons des équations différentielles pour la modélisation. Les équations (7.10) et (7.11) permettent d'écrire pour notre modèle, ce qui suit :

$$\begin{cases} x_i(t) = x_i(t^-) + AIR * N_{rm} \left(1 - \dfrac{x_i(t^-)}{PCR}\right) - \dfrac{y(t-\beta_i)}{Q} x_i(t^-) \\ \\ y(t+\delta t) = \left(y(t) + \left(\displaystyle\sum_{i=1}^{N} x_i(t-\gamma_i) - \mu\right) \delta t\right)^+ \end{cases} \tag{7.12}$$

Où $(x)^+ = max(x, 0)$, et $t^-$ est le dernier instant où $ACR$ est modifié (l'instant d'arrivée de la précédente cellule RM). Puisque la file est supposée être tout le temps non vide (le débit des cellules RM arrivant aux sources est constant), et si le délai entre deux cellules RM est négligé, $ACR$ peut effectivement, être supposé comme le point fixe de l'équation ci-dessus ($x(t) = x(t^-)$) (le débit réagit immédiatement aux changements de la taille de la file d'attente). Par conséquent, le débit d'une source a un comportement très proche du débit obtenu en considérant $x(t) = x(t^-)$. De l'équation (7.12) on obtient

$$\begin{cases} x_i(t) \cong AIR * N_{rm} \left( \dfrac{PCR * Q}{AIR * N_{rm} * Q + PCR * y(t-\beta_i)} \right) \\ \\ \dot{y}\,(t) = \displaystyle\sum_{i=1}^{N} x_i(t-\gamma_i) - \mu \end{cases} \tag{7.13}$$

De (7.13) on a

$$\begin{aligned} \dot{y}\,(t) &= \sum_{i=1}^{N} x_i(t-\gamma_i) - \mu = \sum_{i=1}^{N} x_i(t-\gamma_i) + \sum_{i=1}^{N} AIR * N_{rm} - \frac{AIR * N_{RM}}{PCR} \sum_{i=1}^{N} x_i(t-\gamma_i) \\ &\quad - \frac{1}{Q} \sum_{i=1}^{N} \left( y(t-\tau_i) x_i(t-\gamma_i) \right) - \mu \\ &= \left( 1 - \frac{AIR * N_{RM}}{PCR} \right) \dot{y}\,(t) - \frac{1}{Q} \sum_{i=1}^{N} \left( y(t-\tau_i) \left( x_i(t-\gamma_i) - \frac{\mu}{N} + \frac{\mu}{N} \right) \right) \\ &\quad + \frac{(N * PCR - \mu) AIR * N_{rm}}{PCR} - \frac{AIR * N_{RM}}{PCR} \left( \sum_{i=1}^{N} x_i(t-\gamma_i) - \mu \right) \end{aligned} \tag{7.14}$$

Soient

$$\begin{aligned} H(t) &\triangleq y(t) - \frac{(N * PCR - \mu) AIR * N_{rm} * Q}{\mu * PCR} \\ \text{et} \qquad \dot{H}_i\,(t) &\triangleq x_i(t-\gamma_i) - \frac{\mu}{N} \Longrightarrow \sum_{i=1}^{N} \dot{H}_i\,(t) = \dot{H}\,(t) \end{aligned}$$

On obtient alors de l'équation (7.14)

$$\begin{aligned} \dot{H}\,(t) \quad + \quad & \frac{\mu^2}{N^2 * AIR * N_{rm} * Q} \sum_{i=1}^{N} H(t-\tau_i) = \\ & \qquad - \frac{\mu}{N * Q * AIR * N_{rm}} \sum_{i=1}^{N} H(t-\tau_i)\, \dot{H}_i\,(t) \end{aligned} \tag{7.15}$$

**Proposition 7.2.1** *Pour des sources équivalentes ($\tau_i = \tau\ \forall i$), si*

$$\tau < \tau_0 \triangleq \frac{\pi}{2} \frac{N * Q * AIR * N_{rm}}{\mu^2}$$

*alors la solution-zéro de l'équation (7.15)* $\left( H(t) = 0 \Longleftrightarrow y(t) = \frac{(N*PCR-\mu)AIR*N_{rm}*Q}{\mu*PCR} \right)$ *est uniformément asymptotiquement stable.*

**Preuve:** La preuve découle des Théorèmes 7.1.1 et 7.1.2 après avoir remarqué que pour des sources équivalentes, on a

$$\begin{aligned} (7.15) \quad &\Longleftrightarrow \quad \dot{H}\,(t) = - \frac{\mu^2}{N * AIR * N_{rm} * Q} H(t-\tau) \left( \frac{N * AIR * N_{rm} * Q}{N * AIR * N_{rm} * Q + \mu H(t-\tau)} \right) \\ &\Longleftrightarrow \quad \dot{H}\,(t) + \frac{\mu^2}{N * AIR * N_{rm} * Q} H(t-\tau) = \frac{1}{N * AIR * N_{rm} * Q} \times \\ &\qquad \frac{\mu^3 \left( H(t-\tau) \right)^2}{N * AIR * N_{rm} * Q + \mu H(t-\tau)} \triangleq f(H(t-\tau)) \end{aligned} \tag{7.16}$$

et puisque,

$$\lim_{|u|\to 0} \frac{|f(u)|}{|u|} = 0$$

alors, d'après le Théorème 7.1.2, la stabilité de la solution-zéro de la partie linéaire de l'équation (7.16) implique la stabilité de la solution-zéro de toute l'équation (7.16). La condition nécessaire et suffisante pour la stabilité de la solution-zéro de la partie linéaire de (7.16) découle par application directe du Théorème 7.1.2. ∎

La Figure 7.2 montre que la zone d'instabilité pour l'algorithme du *point fixe* est très proche de $\tau_0$ ($\tau > \tau_0$). Les paramètres considérés dans cet exemple sont les suivants : Une seule source est considérée, traversant un commutateur à bande passante $\mu = 5000$ *cells/sec* (1.272 Mbps). $AIR = 128$, $PCR = 33333.3250$ *cells/sec* $= 14.13$ *Mbps* $Q = 200$ et $ICR = 2000$ *Cells/sec*. De la Proposition 7.2.1, on obtient $\tau_0 = 0.05147$ *sec*. Par conséquent, la solution triviale ($H(t) = 0$) de (7.16) est stable pour $\tau \leq \tau_0$. Dans la Figure 7.2, on peut constater que pour l'algorithme du *point fixe* le système n'est pas stable pour cette valeur ($\tau = 0.050\,\text{sec}$). Cependant, il est stable pour des valeurs plus petites ($\tau = 0.044$ *sec* $> \tau_0 - 1/AIR$).

En fait, la valeur obtenue dans la Proposition 7.2.1 est un peu optimiste. En effet, la région d'instabilité pour le *point fixe* est très proche de cette valeur. Plus exactement, l'heuristique suivante est généralement vraie : pour le *point fixe*, le système est stable pour $\tau < \tau_1$ et instable pour $\tau > \tau_1$ où $\tau_1$ satisfait $\tau_0 > \tau_1 > \tau_0 - 1/AIR$.

Maintenant, supposons que le système est en effet stable (le débit aux sources se stabilise, aussi bien que la taille de la file), alors on a $\dot{y} = 0$, donc $x_i = \mu/N$. De (7.12) on obtient (en posant $Q = \xi B$, $\xi < 1$)

$$\bar{y} = \frac{(N * PCR - \mu) AIR * N_{rm} * Q}{\mu * PCR} = \frac{(N * PCR - \mu) AIR * N_{rm} * \xi B}{\mu * PCR} \tag{7.17}$$

Ce résultat peut être trouvé dans [69]. Il est treś facile de remarquer que la taille de la file quand elle se stabilise, peut dépasser la taille maximale de la file d'attente pour certaines valeurs de $\mu$. En d'autre termes, on peut énoncer le Lemme suivant :

**Lemme 7.2.1** *Il y aura toujours des pertes (pas de stabilité) pour*

$$\mu < \frac{N * PCR}{1 + \frac{PCR}{AIR * N_{rm} * \xi}}$$

Ce résultat est obtenu à partir de l'équation (7.17) en posant $\bar{y} > B$. Les limitations dues aux bandes passantes relativement petites, constituent un majeur (et grand) inconvénient de l'algorithme du *point fixe*, même si le seuil de congestion est pris comme une fraction de la capacité maximale de la file. Il faut rappeler par ailleurs, que pour des bande passantes assez larges l'algorithme n'est pas non plus stable (voir la Proposition 7.2.1).

## 7.3 Algorithmes basés ER

### 7.3.1 ERAQLES

L'algorithme *ERAQLES* [83] (**E**xplicit **R**ate **A**lgorithm using **Q**ueue **LE**ngth **S**tates) propose d'évaluer la bande passante disponible pour le service ABR d'une autre façon que l'algorithme *ERICA* (dont la description est donnée dans la prochaine section), et ce en utilisant la variation de la taille de la file d'attente destinée au service ABR. Dans ce qui suit, on considère que la bande passante estimée est bel et bien la bande passante disponible (comme dans *ERICA*). Le cas où l'algorithme estime la bande passante en fonction des variations de la taille de la file d'attente est présenté dans l'annexe C. L'algorithme peut être résumé brièvement comme suit :

À la source aucun changement, le débit est mis à jour en fonction du champ ER dans les cellules RM reçues par la source (la source négocie un AIR égal à PCR).

### Au commutateur

Le débit explicite pour la source $i$ est calculé en utilisant

$$ER_i(t) = ER(t).s_i(t)$$

où $s_i(t)$ est la fonction du partage équitable (Fair Share), elle est calculée comme suit, en considérant des sources équivalentes de mêmes distances par rapport au commutateur :

$$s_i(t) = (1 - s_0)\frac{CCR_i(t-T)}{CCR(t-T)} + s_0$$

où $CCR_i(t)$ est le débit de la source $i$ à l'instant $t$, et $CCR(t)$ est le débit total du trafic ABR ($0 < s_0 < 1$). $T$ est le temps entre une source est le commutateur (en fait, c'est le temps maximum entre n'importe quelle source et le commutateur en question, selon notre modèle $T \triangleq \max_{i\geq 1}(\beta_i)$).

$ER(t)$ est la bande passante non utilisée par les autres trafics à fortes priorités (CBR, VBR), ajoutée à la bande passante susceptible de remplir la file durant un délai d'aller retour ($T$). Finalement, ER est mis au minimum entre, $ER$ se trouvant dans la cellule RM, et le $ER_i(t)$ calculée. $ER(t)$ est donné par :

$$ER(t) = C^* + h\frac{Q - y(t)}{T}$$

où $C^*$ est la bande passante non utilisée par les trafics à fortes priorités (CBR, VBR), et $y(t)$ est la taille de la file d'attente ABR à l'instant $t$. $Q$ est un seuil donné ($Q = \xi B$, $\xi < 1$ où $B$ est la capacité maximale de la file d'attente).

Dans [83], une approche analytique est donnée ainsi qu'une preuve pour la convergence et l'équité. Il a été ainsi montré que l'équité au sens "max-min" est satisfaite, ainsi que l'utilisation maximale de la bande passante. Cependant, la stabilité de l'algorithme n'a pas été abordée. Dans cette section, nous essayons d'identifier les paramètres adéquats pour assurer la stabilité. Nous montrons que l'algorithme ERAQLES n'est pas stable pour certains paramètres, et les performances de ce schéma sont limitées par la bande passante disponible pour le service ABR. On montrera qu'à l'inverse de l'algorithme du *point fixe*, plus la bande passante est grande, moins les performances d'*ERAQLES* sont bonnes.

### Approche analytique

Considérons le même modèle et les mêmes notations que précédemment, alors l'évolution de $x(t)$ et $y(t)$ est donnée par :

$$\begin{cases} x_i(t) = \left((1-s_0)\dfrac{x_i(t-\tau_i)}{\sum_{\ell=1}^{N} x_\ell(t-\gamma_\ell-\beta_i)} + s_0\right)\left(\mu + h\dfrac{Q-y(t-\beta_i)}{T}\right) \\ y(t+\delta t) = \left(y(t) + \left(\displaystyle\sum_{i=1}^{N} x_i(t-\gamma_i) - \mu\right)\delta t\right)^+ \end{cases} \tag{7.18}$$

$$\text{Soit} \qquad H(t) \triangleq y(t) - Q - \frac{(N-1)s_0\mu T}{((N-1)s_0+1)h} \tag{7.19}$$

Après quelques simplifications, avec l'hypothèse que la file ne se vide jamais et les sources sont équivalentes, et suivant les mêmes étapes que la section précédente, on obtient

$$\dot{H}(t) + h\frac{(N-1)s_0+1}{T}H(t-\tau) = 0 \tag{7.20}$$

**Proposition 7.3.1** *la solution-zéro de l'équation précédente :* $y(t) = Q + \dfrac{(N-1)s_0\mu T}{((N-1)s_0+1)h}$ *est uniformément asymptotiquement stable, si et seulement si*

$$\tau < \tau_0 = \frac{\pi T}{2h((N-1)s_0+1)}$$

**Preuve:** La preuve découle de l'application directe du Théorème 7.1.1.

Par conséquent le schéma n'est pas stable pour tous les paramètres. Sa stabilité dépend essentiellement des paramètres $h$ et $s_0$. Si $h$ satisfait ($s_0 = 1/N$, $2T = \tau$)

$$\frac{\pi}{4h((N-1)s_0+1)} < 1 \Longrightarrow \frac{\pi}{4((N-1)s_0+1)} \leq 0.785 \leq h \tag{7.21}$$

alors le schéma n'est pas stable. Toutefois, les paramètres recommandés dans [83] ($s_0 = 1/N$ et $h = 0.1839$) assurent la stabilité dans tous les cas. La Figure 7.3 illustre le cas d'une seule source pour $h = 0.785$ (instable) et $h = 0.70$ (stable). Les paramètres considérés sont les mêmes que ceux de l'exemple de la précédente section (algorithme du *Point Fixe*, cas stable $T = 0.05$ *sec*, et seule une source est considérée).

Tout comme l'algorithme du *point fixe*, il est facile de constater à partir de l'équation(7.19), que quand le système se stabilise, la taille de la file d'attente dépend de la bande passante disponible pour le service ABR. Après quelques calculs, le Lemme suivant est aisément prouvé (en utilisant l'équation (7.21)) :

**Lemme 7.3.1** *Il y aura toujours des pertes (pas de stabilité) pour*

$$\frac{\tau}{2} \geq \frac{(1-\xi)B((N-1)s_0+1)h}{(N-1)s_0\mu} \Longrightarrow \tau \geq \frac{\pi}{2}\frac{(1-\xi)B}{(N-1)s_0\mu}$$

On peut constater également qu'en considérant $s_0 = 0$, l'algorithme est toujours stable; toutefois, la preuve d'équité [83] (même si elle concerne uniquement le cas où les sources sont équidistantes) n'est plus valide à ce moment là. En fait, le constat suivant est fait : pour $s_0$ très petit, l'équité est garantie dans un temps relativement long, et pour $s_0$ assez large la stabilité peut ne pas être garantie, et la taille de la file d'attente peut également dépasser la taille maximale (voir le Lemme 7.3.1). Ces points constituent les inconvénients majeurs de cet algorithme.

L'étude faite dans cette section, suppose qu'ERAQLES a connaissance de la bande passante exacte, et ne se charge pas de l'estimer. Or, la méthode d'estimation de la bande passante utilisée peut élargir en effet le domaine de stabilité de cet algorithme. Pour plus de détails, voir annexe C où la stabilité est étudiée en tenant compte de cette dernière remarque.

### 7.3.2 ERICA(+)

ERICA+ [59] (**E**xplicit **R**ate **I**ndication **C**ongestion **A**lgorithm), proposé par Jain et al, est un algorithme basé sur le débit explicite ensemble avec l'occupation des files d'attente aux commutateurs intermédiaires. Il est apparemment, l'algorithme le plus référencé dans la littérature sur le contrôle dans l'ABR. De nombreuses simulations dans différents environnements ont été faites, et plusieurs améliorations ont été apportées. Il faut tout de même rappeler que ces améliorations apportent une complexité supplémentaire. L'équité et la stabilité du schéma ont été abordées dans plusieurs contributions, en utilisant essentiellement des simulations. Dans la section présente, notre objectif est d'étudier la stabilité de cet algorithme, en considérant différents paramètres.

*ERICA* fonctionne de la façon suivante (voir [59] pour plus de détails) :

Le commutateur détermine d'une manière périodique sur chaque lien le facteur de charge $z$, la bande passante disponible pour le service ABR (Capacité ABR), et le nombre de circuits virtuels (VCs) actifs. Le facteur de charge est calculé comme suit :

$$z \leftarrow \frac{\text{Débit ABR Entrant}}{\text{Capacité ABR}}$$

$$\text{où} \quad \text{Capacité ABR} \leftarrow \text{Taux d'Utilisation} \times \text{Capacité du Lien} - \text{Usage VBR} - \text{Usage CBR}$$

Dans le but d'atteindre 100 % d'utilisation de la bande passante, *ERICA* introduit la taille de la file d'attente comme paramètre de contrôle (*ERICA+*), et le *Taux d'Utilisation* est alors mis à un. Ainsi, la *Capacité ABR* est révisée comme suit (on considère, dans ce qui suit, un *seuil de congestion* constant ($Q$)) :

$$\text{Capacité ABR} \leftarrow \text{Capacité ABR} \times \frac{b * Q}{(b-1) \times \text{Taille de de file} + Q}$$

Les valeurs recommandées pour $b$ sont

$$b = 1.15 \text{ si Taille de la file} > Q; \quad b = 1.05 \quad \text{ailleurs}$$

Finalement, le débit explicite signalé à chaque source est mis à jour comme suit (à noter que le *Facteur de Charge* est aussi changé) :

$$ER\_Calculated \leftarrow \text{Max}\left(\frac{\text{Capacité ABR}}{\text{Nombre de VCs Actifs}}, \frac{CCR[VC]}{z}\right)$$

$CCR[VC]$ est le débit entrant du VC ; il est, soit estimé par le commutateur (Nombre de cellule Reçue/Intervalle d'Estimation), soit relevé dans la cellule RM "en aval" du VC (si le comptage par VC n'est pas supporté par le commutateur)

### Approche analytique

Pour analyser la stabilité de l'algorithme *ERICA+*, on a considéré le cas où *ERICA* tente l'utilisation à 100 % de la bande passante disponible, tout en utilisant un *seuil de congestion* constant ($Q$). En considérant le même modèle et les mêmes notations que précédemment, l'évolution du débit et de la taille de la file d'attente dans *ERICA+* peuvent être résumés par les équations suivantes (le délai entre deux cellules RM est négligé) :

$$\begin{cases} x_i(t) = max\left(\frac{1}{N}, \frac{x_i(t-\tau_i)}{\sum_{\ell=1}^{N} x_\ell(t-\gamma_\ell-\beta_i)}\right) * ER\frac{b*Q}{(b-1)y(t-\beta_i)+Q} \\ \\ y(t+\delta t) = \left(y(t) + \left(\sum_{i=1}^{N} x_i(t-\gamma_i) - \mu\right)\delta t\right)^+ \end{cases} \tag{7.22}$$

$b$ prend deux valeurs, comme mentionné ci-dessus, selon que la taille de la file ABR est plus grande ou plus petite que $Q$ (les valeurs recommandées sont respectivement 1.15 et 1.05). Dans l'analyse qui suit, nous considérons que la valeur est unique ; et nous identifions la zone d'instabilité pour le cas le meilleurs (qui correspondra intuitivement à la valeur la plus petite de $b$). Soit

$$H(t) \triangleq y(t) - Q.$$

En procédant de la même manière que dans les sections précédentes, et après quelques calculs et simplifications, pour le cas des sources équivalentes, on obtient de (7.22)

$$\dot{H}(t) + \frac{(b-1)\mu}{bQ} H(t-\tau) = -\frac{(b-1)}{bQ} H(t-\tau)\,\dot{H}(t) \tag{7.23}$$

**Proposition 7.3.2**

$$Si \qquad \tau < \tau_0 = \frac{\pi}{2}\frac{bQ}{(b-1)\mu},$$

*alors la solution-zéro de l'équation (7.23) ($H(t) = 0 \Longleftrightarrow y(t) = Q$) est uniformément asymptotiquement stable.*

**Preuve:** La preuve est similaire à celle de la Proposition 7.2.1, en remarquant que

$$(7.23) \Longleftrightarrow \dot{H}(t) = -\frac{(b-1)\mu}{bQ} H(t-\tau)\left(\frac{bQ}{bQ + (b-1)H(t-\tau)}\right)$$

$$\Longleftrightarrow \dot{H}(t) + \frac{(b-1)\mu}{bQ} H(t-\tau) = \frac{\mu((b-1)H(t-\tau))^2}{(bQ)^2 + bQ(b-1)H(t-\tau)} \triangleq f(H(t), H(t-\tau)).$$

∎

La Figure 7.4 (a) illustre un exemple où l'algorithme *ERICA+* est instable. Les paramètres considérés dans cet exemple sont les suivants : la bande passante disponible pour le service ABR est $\mu = 100000\ Cells/sec$, le *seuil de congestion* $Q$ est mis à 200 $Cells$ et $\tau = 0.064\ sec$. De la Proposition 7.3.2 on a $\tau_0 = 0.06597$ pour $b = 1.05$, et $\tau_0 = 0.02408$ pour $b = 1.15$. On déduit alors, que la zone d'instabilité est entre ces deux valeurs de $\tau_o$. En effet, dans plusieurs exemple on a vérifié que *ERICA+* est instable si $\tau > \tau_0$, où $\tau_0$ est la valeur obtenue pour la plus petite valeur de $b$ ($b = 1.05$). Ce qui est confirmé en particulier, dans l'exemple illustré par la Figure 7.4 (a).

S'appuyant sur la Proposition 7.3.2, on peut choisir le paramètre $b$ de telle façon que la stabilité soit assurée dans tous les cas. Par exemple, au lieu que la fonction $fn()$ soit définie comme $fn(y(t)) \triangleq \dfrac{bQ}{(b-1)y(t)+Q}$, on la définit comme : $fn(y(t)) \triangleq \dfrac{Q}{by(t)+(1-b)Q}$.

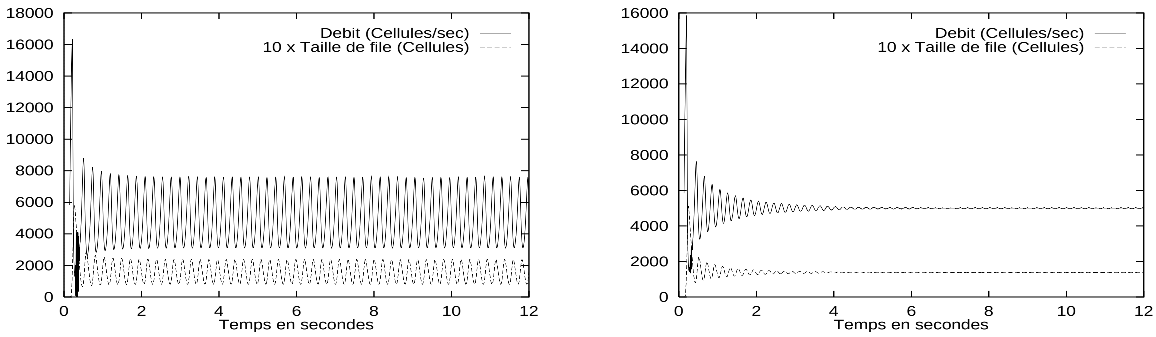
$$\text{Où} \quad b \triangleq \begin{cases} Min\left(0.05, \dfrac{\pi Q}{6 * ER * \tau_{max}}\right) & \text{si} \quad \text{Taille de la file} < Q \\ Min\left(0.15, \dfrac{\pi Q}{2 * ER * \tau_{max}}\right) & \text{Ailleurs} \end{cases}$$

La Figure 7.4 (b) montre les effets de la nouvelle fonction $fn()$. Le débit, aussi bien que la taille de la file d'attente sont constants après un temps très court, pendant qu'ils étaient oscillatoires dans le cas de *ERICA+* (Figure 7.4 (a)).

## 7.4 Conclusion

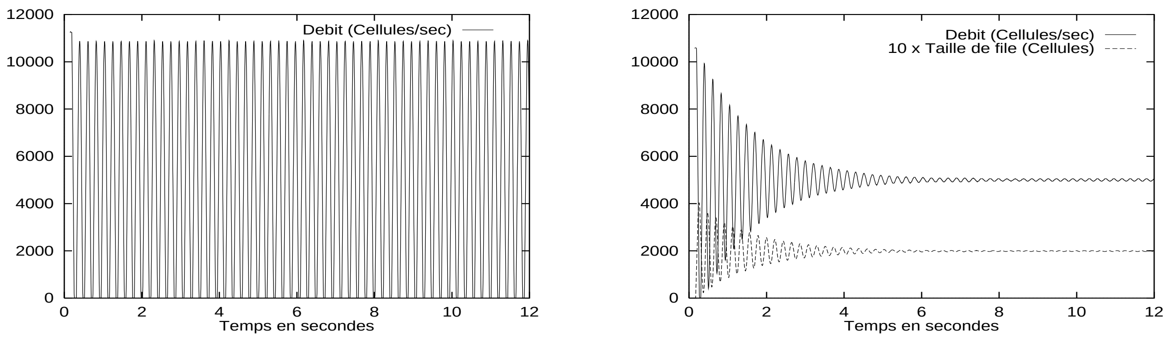
Dans le présent travail, la stabilité de différents algorithmes de contrôle de congestion pour l'ABR a été étudiée, et le domaine de stabilité de chacun a été calculé. Bien que l'algorithme du *Point Fixe* soit équitable et très simple à implémenter, il ne garantit malheureusement pas de stabilité sous certaines configurations. *ERAQLES* permet une convergence assez rapide, cependant les tailles des files d'attente au niveau des commutateurs intermédiaires peuvent être très larges (ERAQLES est très agressif). Finalement, nous avons montré que même *ERICA+* n'est pas stable pour certains paramètres, et nous avons proposé des valeurs pour le paramètre $b$ permettant d'avoir de meilleures performances.

Nous avons montré dans ce chapitre également, la force de la méthode de stabilité des solutions aux équations différentielles à retard quant à l'analyse de stabilité des algorithmes pour l'ABR. Nous nous sommes restreints à l'étude de quelques algorithmes ; cependant de nombreux autres basés sur le débit explicite et/ou la taille de la file d'attente peuvent être étudiés de la même façon.



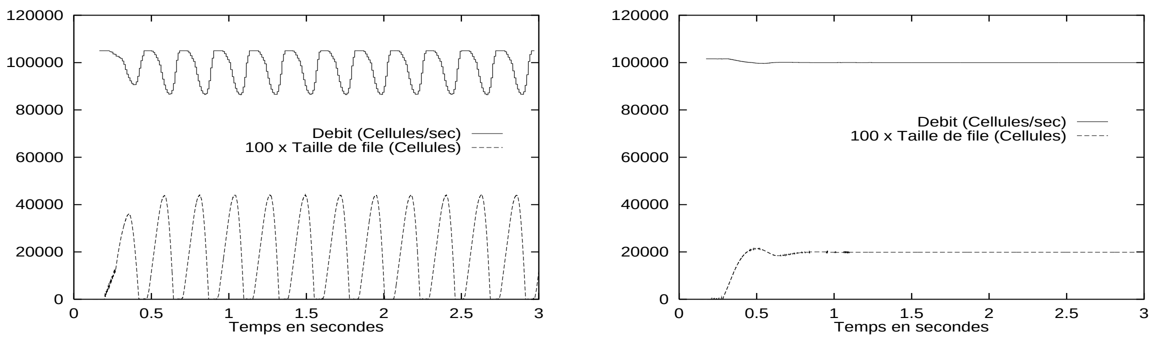

(a) $\tau = 0.050$ (b) $\tau = 0.044$

FIG. 7.2: Taille de la file d'attente (cellules) et débit de transmission (cellules/sec), pour $\tau = 0.044$ $sec$ (stable) et $\tau = 0.050$ $sec$ (instable), sous l'algorithme du Point Fixe.




(a) $h = 0.785$ (b) $h = 0.700$

FIG. 7.3: Débit de transmission (cellules/sec), pour $h = 0.70$ (stable) et $h = 0.785$ (instable), sous l'algorithme ERAQLES.




(a) *ERICA+* (b) enhanced ERICA+

FIG. 7.4: Taille de la file d'attente (cellules) et débit de transmission (cellules/sec), pour $\tau = 0.064$ $sec$, $\mu = 100000$ $Cellules/sec$ et $Q = 200$ $Cellules$ sous l'algorithme *ERICA+* (avant et après l'introduction de la nouvelle fonction $fn(Queue)$).

# Chapitre 8

# Analyse et conception d'algorithmes ABR

Dans le Chapitre 6, une méthode directe est utilisée pour calculer des bornes supérieures sur les tailles des files d'attente durant la phase transitoire, aussi bien qu'à l'état stationnaire, pour les algorithmes basés EFCI ainsi qu'EPRCA. Le cas général de plusieurs commutateurs en tandem a été étudié. Dans le chapitre précédent, la méthode des équations différentielles à retard a été utilisée dans le but d'analyser la stabilité de différents algorithmes (les algorithmes basés sur la taille des files d'attente, débit explicite ...etc). Ainsi, les domaines de stabilité de l'algorithme du *Point Fixe*, *ERAQLES* et *ERICA+* ont été calculés.

Nous basant sur les études précédentes, nous proposons dans le présent chapitre une méthode plus générale pour évaluer analytiquement un algorithme de contrôle de flux basé sur le débit explicite (et/ou la taille de la file d'attente), pour le service ABR. La stabilité, l'équité et l'utilisation maximale de la bande passante sont spécialement considérées. Ces dernières, constituent les critères essentiels qu'un algorithme de contrôle de flux, basé sur le débit explicite, pour le service ABR devrait satisfaire. Des exemples où au moins l'un des critères cités ci-dessus n'est pas satisfait, sont décrits et analysés en s'appuyant sur une approche analytique aussi bien que sur des simulations.

## 8.1 Que doit satisfaire un algorithme pour l'ABR ?

Dans tout le reste de ce chapitre, nous utiliserons le même modèle décrit dans le chapitre précédent. Nous établirons ainsi les conditions nécessaires, que doit satisfaire un algorithme basé sur le débit explicite, pour le service ABR.

Nous considérons uniquement des commutateurs basés sur le débit explicite (la taille de la file d'attente peut être un paramètre de contrôle au moment du calcul de ce débit). Dans ce genre de commutateurs, le débit explicite est calculé et transmis par la suite, via une cellule RM, aux sources. La source met à jour son débit en se conformant au champs $ER$ de la cellule RM ("en amont") reçue ($ACR(t) = ER_i(t - \beta_i)$). Le débit attribué à la source $i(ER_i(t))$ par le commutateur est calculé en général en se basant sur le nombre de connexions $(N)$, le débit total d'entrée du trafic ABR ($\sum_{i=1}^{N} CCR_i(t - \gamma_i)$), le débit de la connexion $i$ ($CCR_i(t - \gamma_i)$), la taille de la file d'attente au niveau du commutateur ($y(t)$) et la bande passante disponible pour le service ABR ($\mu$), que nous supposons constante. De ce fait, il est tout à fait possible de résumer le comportement du système décrit, dans les équations suivantes ($x_i(t) \triangleq CCR_i(t)$) :

$$\begin{cases} x_i(t) = f(x_1(t-\beta_i-\gamma_1), x_2(t-\beta_i-\gamma_2), \cdots, x_N(t-\beta_i-\gamma_N), y(t-\beta_i), N, \mu) \\ \dot{y}\ (t) = \sum_{i=1}^{N} x_i(t-\gamma_i) - \mu \end{cases} \tag{8.1}$$

Note : Les algorithmes n'utilisant pas la taille de la file d'attente, comme paramètre de contrôle, ne sont pas stables s'ils visent à utiliser 100% de la bande passante [9, 15](ERICA par exemple).

### 8.1.1 Stabilité

Pour des sources équivalentes, De l'équation ci-dessus, il est toujours possible d'exprimer $y(t)$ comme fonction d'une équation différentielle à retard de premier ou second ordre. Pour des connexions équivalentes, l'équation obtenue peut toujours se mettre sous la forme suivante :

$$\begin{aligned} a\,\ddot{H}\ (t) + b\,\dot{H}\ (t-\tau) + c\,\dot{H}\ (t) &+ \alpha H(t-\tau) + \beta H(t) \\ &= G(\dot{H}\ (t-\tau), \dot{H}\ (t), H(t-\tau), H(t)) \end{aligned} \tag{8.2}$$

où $H(t) \triangleq y(t) - \bar{y}$, $G(0,0,0,0) = 0$ et $H(t) = 0$ (équivalent à $y(t) = \bar{y}$) est une solution triviale. Ceci dit s'il y a stabilité, alors la taille de la file d'attente tentera de se stabiliser à la taille $\bar{y}$ (voir chapitre précédent pour la definition des différents paramètres du modèle. $\bar{y}$ en général est la valeur de $y(t)$ pour laquelle $\dot{y}\ (t) = 0$). Deux genres d'analyses sont alors possibles :

La première est de savoir si on pourrait avoir $\bar{y} > B$, où $B$ est la taille maximale de la file d'attente. Si c'est le cas, alors des pertes pourraient avoir lieu, et par conséquent l'algorithme ne peut être stable pour les paramètres satisfaisant l'inégalité précédente. L'algorithme du *Point Fixe* et *ERAQLES* sont des exemples de ce genre d'algorithmes, pour lesquels une étude détaillée est donnée dans le précédent chapitre.

La deuxième est d'utiliser la théorie de stabilité des solutions aux équations différentielles à retard [5, 20, 51, 71]. Si la solution $y(t) = \bar{y}$ est uniformément asymptotiquement stable, alors quels que soient les paramètres initiaux[1], la taille de la file d'attente va se stabiliser à $\bar{y}$ avec de minimes oscillations au départ. Les exemples d'*ERICA+*, *ERAQLES* et l'algorithme du *Point Fixe*, ont été analysés dans le précédent chapitre, et les conditions de stabilité pour ces derniers ont été établies.

Dans ce qui suit, nous considérons uniquement des algorithmes où le débit de chaque source devient constant, pour une exécution relativement longue, si la bande passante totale disponible est constante (le débit de chaque source se stabilise, le cas échéant avec de minimes oscillations au départ).

[1] Les paramètres doivent être les mêmes pour des connexions équivalentes, dans le service ABR, et sont choisis tels que le débit soit le plus proche possible du débit disponible.

### 8.1.2 Utilisation maximale

Nous nous référons à l'utilisation maximale parfois par la convergence de l'algorithme. Le débit attribué à chaque source doit être une fonction croissante, si toute la bande passante n'est pas utilisée. Ceci pour assurer que dans le cas où des sources restreintes ou silencieuses existent, la bande passante non utilisée par ces dernières est récupérée pour être partagée sur les sources non restreintes s'il en existe. C'est équivalent à augmenter le débit tant que la file d'attente est vide.

Supposons que le débit de chaque source peut s'écrire sous la forme de l'équation (8.1). Alors, la condition nécessaire (puisque le débit de chaque source converge vers une certaine valeur constante) et suffisante pour assurer la convergence est la suivante :

**Lemme 8.1.1** *Sous les hypothèses ci-dessus, un algorithme ABR basé sur le débit explicite est convergent (i.e. il garantit l'utilisation maximale), si et seulement si*

$$\begin{aligned} \forall\, \mu_{max} \leq \mu, \quad \exists\, t_0 : Si \sum_{k=1,k\neq i}^{N} x_k(t) = \mu_{min} = \mu - \mu_{max}\ \forall t, \\ alors, \quad x_i(t) \geq \mu_{max} \forall\, t > t_0 \end{aligned} \tag{8.3}$$

**Preuve:** La preuve est triviale, du moment que c'est la définition même de l'utilisation maximale. Il est facile de vérifier, que si la condition ci-dessus est satisfaite, alors l'utilisation maximale est garantit. En effet, on aura $\forall t > t_o$, $\sum_{k=1}^{N} x_k(t-\gamma_k) \geq \mu$. D'autre part, si $\exists \mu_{max} \leq \mu$ : $\forall t\ x_i(t) < \mu_{max}$, alors l'utilisation maximale n'est pas garantit. En effet, prenons le modèle décrit précédemment, et supposons que $N-1$ sources sont restreintes ainsi $\sum_{k=1,k\neq i}^{N} x_k(t) = \mu - \mu_{max}$, et seule la source $i$ est non restreinte. Puisque $x_i(t) < \mu_{max}\ \forall t$, il en découle que $\sum_{k=1}^{N} x_k(t) < \mu\ \forall t$, et par conséquent l'algorithme n'est pas convergent.

Un exemple d'algorithmes non convergents est le "pure-rate matching" [9, 15]. C'est en fait un "pure rate matching " corrigé par la taille de la file d'attente, qu'on appelera dans ce chapitre simplement "pure rate matching". Dans cet algorithme, le débit est mis à jour de la façon suivante : Supposons que le nombre de connexions actives est $N$, et que la bande passante disponible au niveau du commutateur est $\mu$, alors le débit affecté à chaque source est donné par :

$$ER_i(t) = (1-\alpha)x_i(t-\gamma_i) + \alpha\frac{\mu}{N} + \beta(Q_H - y(t)) = x_i(t+\beta_i)$$

où $y(t)$ est la taille de la file d'attente à l'instant $t$. Il est facile de vérifier que le débit maximale $(x_i(t))$, attribué à une source $i$, est obtenu pour $y(t-\beta_i) = 0$, par conséquent on a

$$\begin{aligned} x_i(t) &\leq (1-\alpha)x_i(t-\tau_i) + \alpha\frac{\mu}{N} + \beta Q_H \\ \Longrightarrow x_i(t) - x_i(t-\tau_i) &\leq \alpha\left(\frac{\mu}{N} - x_i(t-\tau_i) + \frac{\beta}{\alpha}Q_H\right) \\ \Longrightarrow x_i(t) < x_i(t-\tau_i) &\quad \text{si} \quad x_i(t-\tau_i) > \frac{\mu}{N} + \frac{\beta}{\alpha}Q_H \end{aligned}$$

Récursivement, on peut prouver que $\forall t\ \ x_i(t) \leq \frac{\mu}{N} + \frac{\beta}{\alpha}Q_H \stackrel{\Delta}{=} \mu_{max}$. De ce fait $\exists\ \alpha, \beta, N$ : $\mu_{max} \leq \mu$ et $\forall t\ \ x_i(t) \leq \mu_{max}$. Alors, en vertu du Lemme 8.1.1, il découle que le schéma n'est pas convergent. En d'autres termes, si la bande passante disponible pouvant être affectée à la source $i$, à cause des sources restreintes ou silencieuses est plus large que $\frac{\mu}{N} + \frac{\beta}{\alpha}Q_H$, alors l'utilisation maximale ne sera pas garantit, sous certaines configurations du réseau.

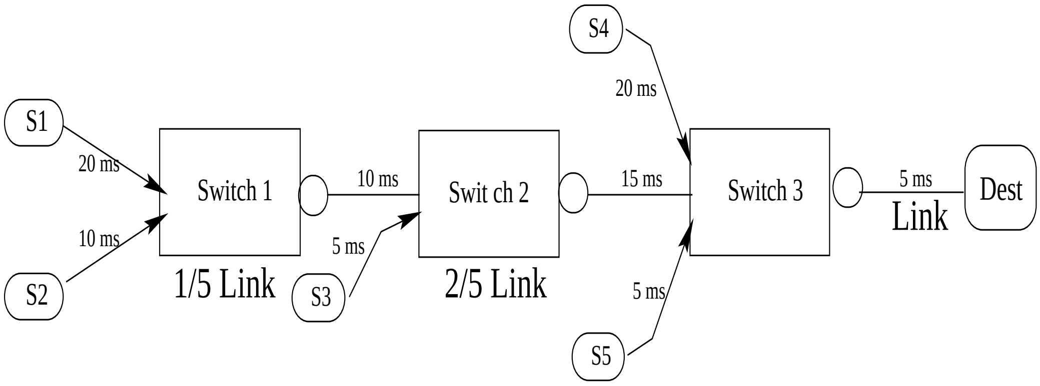



FIG. 8.1: Scénario des simulations pour la convergence et l'équité.

Le scénario dans la figure 8.1 est considéré dans le but de vérifier la convergence et l'équité des différents exemples présentés dans ce chapitre. Le scénario est le suivant : toutes les sources ont la même destination (Dest). La bande passante disponible au niveau du troisième commutateur est $Link = 20000\ cell/sec$, et $1/5\ Link = 4000\ cell/sec$, $2/5\ Link = 8000\ cell/sec$, respectivement au niveau du premier et second commutateur. Les sources 2, 3, 5 commencent à transmettre deux secondes avant les sources 1, 4. Avant ces deux secondes de simulation, les sources, idéalement (si convergence et équité sont assurées), doivent transmettre avec les débits suivants : S1, S3 : 4000 cellules/sec et S5 : 12000 cellules/sec. Après les deux secondes de simulation, on devrait avoir dans le cas idéal, la configuration suivante (débit de chaque source) : S1, S2 : 2000 cellules/sec, S3 : 4000 cellules/sec, et S4, S5 : 6000 cellules/sec.

La Figure 8.2 *(a)* montre le débit de chaque source entre l'instant 0 et 2 secondes, quand l'algorithme ci-dessus est considéré. On voit que la source 5, transmet à un débit (6500 $cell/sec$) nettement inférieur à celui dont elle peut en disposer (12000 $cell/sec$). L'utilisation pour les deux secondes de simulation, dans cet exemple est de 72.5%.

Considérons un autre algorithme, l'algorithme $UT$ (Uniform Tracking Algorithm) [44]. Dans cet algorithme, le débit pour la source $i$ est mis à jour comme suit :

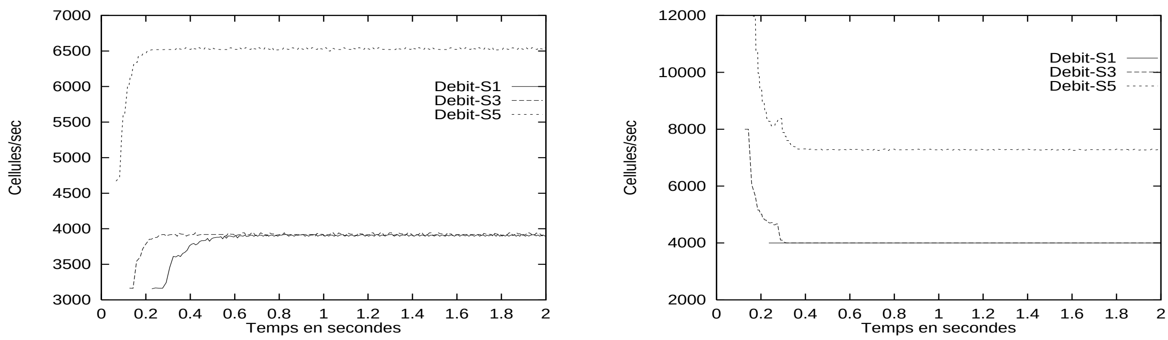
$$x_i(t) = FS(t) = \frac{\mu}{N_{eff}} \quad \text{où} \quad N_{eff} = max\left\{\frac{\sum_{k=1}^{N} x_k(t-\gamma_i)}{\widetilde{FS(t)}}, 1\right\}$$

où $\widetilde{FS(t)}$ est la moyenne sur le débit ABR entrant calculé périodiquement, tout les délai d'aller retour maximum $T$. Quand une cellule RM, d'une source donnée $\ell$ est reçue au niveau du commutateur, $\widetilde{FS(t)}$ est mis à jour comme suit : $\widetilde{FS} := (1-\alpha)\widetilde{FS} + \alpha RM_\ell -> CCR = (1-\alpha)\widetilde{FS} + \alpha x_\ell(t-\gamma_i))$.

Supposons que $\bar{x}_i(t-\gamma_i)$ est le débit moyen pris sur $T$ pour la source $i$. Alors, il est facile de montrer que

**Proposition 8.1.1**

$$N_{eff} \approx \frac{\left(\sum_{i=1}^{N} \bar{x}_k(t-\gamma_k)\right)^2}{\sum_{i=1}^{N} \bar{x}_k(t-\gamma_k)^2} \tag{8.4}$$




(a) Pure Rate matching

(b) Uniform Tracking

FIG. 8.2: Débits en cellules/sec pour les algorithmes "pure-rate matching" et $UT$ (Uniform Tracking).

**Preuve:** Durant un intervalle de temps $T$, $FS$ est mis à jour chaque fois qu'une cellule RM d'une source donnée $i$ est reçue :

$$FS := CCR_i \quad \text{si } FS = 0; \quad \text{sinon} \quad FS := (1-\alpha)FS + \alpha CCR_i$$

Durant ce temps, en utilisant le débit moyen de la source sur cet intervalle, on reçoit $b_i$ cellules RM en provenance de la source $i$, où $b_i = \frac{\overline{CCR_i * T}}{N_{rm}}$. D'autre part, le nombre total de cellules RM reçues durant cette période est donné par

$$b = \frac{\sum_{i=1}^{N} \overline{CCR}_i * T}{N_{rm}}.$$

Par conséquent, la probabilité qu'une cellule RM reçue appartienne à la source $i$ est donnée par

$$p_i = \frac{b_i}{b} = \frac{\overline{CCR}_i}{\sum_{k=1}^{N} \overline{CCR}_k}.$$

Considérons, maintenant, $FS$ aux instants d'arrivée des cellules RM. $FS_n$ à l'arrivée de la $n$ème cellule RM ($n \in \{1, \cdots, b\}$), dans la même période $T$ est donné par

$$FS_{n+1} = (1-\alpha)FS_n + \alpha t_{n+1}, \tag{8.5}$$

où $t_{n+1}$ est le débit moyen $\overline{CCR}$ de la connexion d'où provient la $n+1$ème cellule. On a alors $t_{n+1} = \overline{CCR}_i$ avec la probabilité $p_i$. A partir de l'équation (8.5), on déduit

$$FS_n = (1-\alpha)^{n-1} t_1 + \sum_{j=2}^{n} \alpha (1-\alpha)^{n-j} t_j \tag{8.6}$$

$$\text{Ainsi} \qquad \mathbb{E}[FS_n] = (1-\alpha)^{n-1} \mathbb{E}[t_1] + \sum_{j=2}^{n} \alpha (1-\alpha)^{n-j} \mathbb{E}[t_j] \tag{8.7}$$

D'autre part, on a

$$\mathbb{E}[t_1] = \sum_{k=1}^{N} p_k \overline{CCR}_k = \sum_{k=1}^{N} \frac{b_k}{b} \overline{CCR}_k$$

$$\begin{aligned}
\text{et} \quad \mathbb{E}[t_2] &= \sum_{j=1}^{N} \frac{b_j}{b} \left( \sum_{k=1,k\neq j}^{N} \frac{b_j}{b-1}\overline{CCR}_k + \frac{b_j-1}{b-1}\overline{CCR}_j \right) \\
&= \frac{1}{b(b-1)} \sum_{j=1}^{N} b_j \left( \sum_{k=1}^{N} b_k \overline{CCR}_k - \overline{CCR}_j \right) \\
&= \frac{1}{b(b-1)} \left( \sum_{k=1}^{N} b \times b_k \overline{CCR}_k - \sum_{j=1}^{N} b_j \overline{CCR}_j \right) \\
&= \sum_{k=1}^{N} \frac{b_k}{b} \overline{CCR}_k = \mathbb{E}[t_1] \qquad (8.8)
\end{aligned}$$

En suivant le raisonnement, on peut prouver que $\mathbb{E}[t_i] = \mathbb{E}[t_j] \quad \forall\ i, j$. Par conséquent, de l'équation (8.7), on obtient finalement ($b$ large)

$$\begin{aligned}
\widetilde{FS} &\approx \mathbb{E}[FS_n] = \sum_{k=1}^{N} \frac{b_k}{b}\overline{CCR}_k = \frac{\sum_{k=1}^{N} \overline{CCR}_k^{\,2}}{\sum_{j=1}^{N} \overline{CCR}_j} \\
\Longrightarrow N_{eff} &= \frac{\sum_{j=1}^{N} \overline{CCR}_j}{\widetilde{FS}} \approx \frac{\left(\sum_{j=1}^{N} \overline{CCR}_j\right)^2}{\sum_{j=1}^{N} \left(\overline{CCR}_j\right)^2} \qquad (8.9)
\end{aligned}$$

Notons que, puisque le commutateur est distant de $\gamma_i$ secondes de la source $i$, l'équation (8.9) peut s'écrire alors comme suit :

$$N_{eff} \approx \frac{\sum_{j=1}^{N} \bar{x}_j(t-\gamma_j)}{\widetilde{FS}} = \frac{\left(\sum_{j=1}^{N} \bar{x}_j(t-\gamma_j)\right)^2}{\sum_{j=1}^{N} \bar{x}_j(t-\gamma_j)^2}$$

■

Supposons que les débits des sources se stabilisent pour un temps relativement long (stabilité), alors $\bar{x}_i(t) = x_i(t)\ \forall t$. Soit $x_\ell(t-\gamma_\ell) \triangleq \max_{k\geq 1} x_k(t-\gamma_k)$, et supposons que $\sum_{k=1,k\neq \ell}^{N} x_k(t-\gamma_k) = \mu_{min} << \frac{N-1}{N}\mu\ \forall t$ (ceci est vérifié s'il existe une seule source $i$ restreinte telle que $x_i(t-\gamma_i) < \frac{\mu}{N}$). Si toute la bande passante est utilisée [44] et sans depassement, alors on a $x_\ell \leq \mu - \mu_{min}$. Ainsi,

$$\begin{aligned}
&N_{eff} = \frac{\left(\sum_{k=1}^{N} x_k(t-\gamma_k)\right)^2}{\sum_{k=1}^{N} x_k(t-\gamma_k)^2} > \frac{(\mu_{min}+x_\ell)^2}{\mu_{min}^2 + x_\ell^2} \\
&\Longrightarrow x_\ell(t) = \frac{\mu}{N_{eff}} < \frac{\mu_{min}^2 + x_\ell^2}{\mu_{min}^2 + x_\ell^2 + 2x_\ell\mu_{min}}\mu \leq \mu - 2\frac{\mu-\mu_{min}}{\mu}\mu_{min} \\
&\Longrightarrow \text{si} \quad \mu_{min} < (2-\alpha)\mu \quad \text{, alors} \quad x_\ell(t) < \mu - \alpha\mu_{min} < \mu - \mu_{min} \quad \forall t, \alpha > 1
\end{aligned}$$

Il en découle que $x_\ell(t) < \mu - \mu_{min}\ \forall\ t\ (\alpha = 1)$, par conséquent $\sum_{k=1}^{N} x_k(t) < \mu\ \forall\ t > t_0$ ($t_0$ peut être quelques délais d'aller retour). Le schéma n'est donc pas convergent.

La Figure 8.2 *(b)*, illustre le débit de chaque connexion, en considérant le même scénario que précédemment (figure 8.1). On peut constater, qu'à la source 5 est affecté un débit plus petit que $7300 cell/sec$, pendant qu'elle peut transmettre sans saturer le réseau au débit de

12000 $cell/sec$. L'utilisation dans cet exemple pour les deux secondes de simulation est de 76.5%, et elle peut être moins, si la bande passante que peut utiliser la source 5 est plus large. Remarquons que les débits des sources 1 et 3 sont les mêmes et l'équité est atteinte rapidement.

### 8.1.3 Équité

La condition suffisante pour l'équité, en utilisant le modèle du chapitre précédent est la suivante : on dira qu'un algorithme est équitable si

$$\sum_{k=1}^{N} x_k(t-\gamma_k) = \mu \Longrightarrow |x_i(t+\beta_i) - x_j(t+\beta_j)| < |x_i(t-\gamma_i) - x_j(t-\gamma_j)| \ \forall i \neq j$$
$$\text{ou} \quad x_i(t+\beta_i) - x_j(t+\beta_j) = 0 \ \forall t \tag{8.10}$$

Ceci signifie, que le débit alloué pour tout couple de connexions, au même instant est le même, ou tend à être le même pour un temps d'exécution relativement long, du moment que la différence entre les débits alloués à deux sources $(i, j)$, à l'instant présent est inférieur à la différence entre leur débit de transmission (débit entrant au commutateur).

D'autre part, on dira qu'un algorithme n'est pas équitable

$$\text{si} \quad \exists i, j \ : \sum_{k=1}^{N} x_k(t) = \mu \quad \text{et} \quad |x_i(t) - x_j(t)| \geq C^{st} > 0 \ \forall t$$

Un exemple d'algorithmes non équitables est le suivant (algorithme ERICA première version [1, 62]) :

$$ER_i(t) = max\left(\frac{1}{N}, \frac{x_i(t-\gamma_i)}{\sum_{k=1}^{N} x_k(t-\gamma_k)}\right)\mu_1 \quad \text{où} \quad \mu_1 \triangleq \mu * \text{Target\_Utilisation}.$$

ERICA tente d'utiliser uniquement une fraction de la bande passante. Ainsi, nous considérons que la bande passante disponible vue par ERICA est $\mu_1$ au lieu de $\mu$, car elle vise à avoir $\sum_{k=1}^{N} x_k(t) = \mu_1$. Ceci donne, pour des sources non restreintes ayant un débit plus grand que $\mu_1/N$ :

$$x_i(t) = \frac{x_i(t-\tau_i)}{\sum_{k=1}^{N} x_k(t-\beta_i-\gamma_k)}\mu_1$$

Par conséquent,

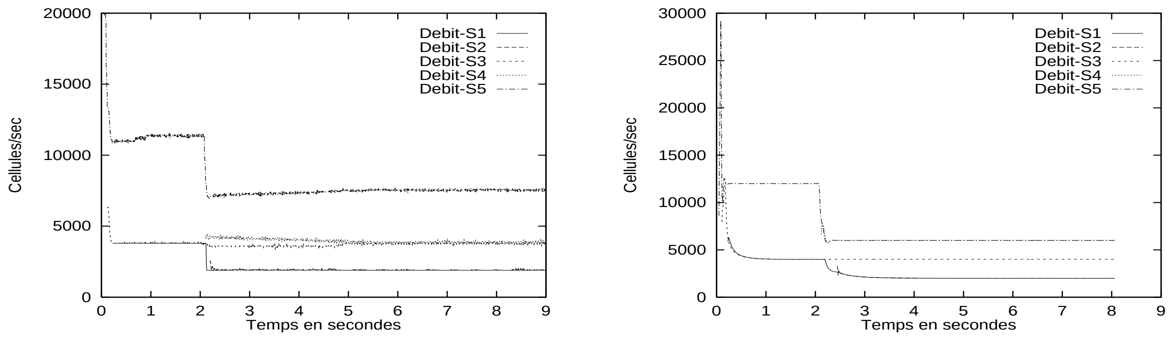
$$|x_i(t) - x_j(t)| = \left|\frac{x_i(t-\tau_i)}{\sum_{k=1}^{N} x_k(t-\beta_i-\gamma_k)} - \frac{x_j(t-\tau_j)}{\sum_{k=1}^{N} x_k(t-\beta_j-\gamma_k)}\right|\mu_1 \tag{8.11}$$

En fait, l'équité dans ce schéma (quand elle a lieu) est causée par le fait de prendre le maximum entre le débit proportionnel et le partage équitable $\mu_1/N$, ainsi que de diviser par la quantité $\sum_{k=1}^{N} x_k(t-\gamma_k)$. Ceci est vrai, dans le cas des sources commençant relativement à un même niveau $(\mu_1/N)$. Mais quand une source commence à transmettre pendant que d'autres sont dans leurs état stationnaire, on peut avoir la situation où l'utilisation maximale est atteinte $(\sum_{k=1}^{N} x_k(t-\beta_j-\gamma_k) = \mu \ \forall t, j)$, sans pour autant que les sources aient le même débit de transmission. En effet, de l'équation (8.11), on a

$$\sum_{k=1}^{N} x_k(t) = \mu_1 \ \forall t \Longrightarrow |x_i(t) - x_j(t)| = |x_i(t-\tau_i) - x_j(t-\tau_j)|$$

Il en découle, qu'il existe des partages où le débit total alloué égale à la bande passante, pendant que les sources peuvent avoir des débit différents, par conséquent l'équité n'est pas garantit (même pour des sources équivalentes $\tau_i = \tau_j$).

Comme illustration, la Figure 8.3 (a) montre le débit de chaque source, en considérant le scénario de la Figure 8.1, où ERICA tente d'utiliser 95% de la bande passante ; soit 19000 *cellules/sec*. On peut constater que la bande passante totale est effectivement consommée, mais avec des partages non équitables. En effet, à la source 5 est attribué un débit de 7500 cellules/sec, et les sources 3 et 4 ont reçu moins de 3800 cellules/sec chacune, enfin les sources 1 et 2 ont reçu 1900 cellules/sec chacune. La source 4 (non restreinte dans cet exemple) en particulier reçoit prèsque la moitié de ce que reçoit la source 6, au moment où elle devait recevoir le même débit si l'algorithme était équitable. A noter également que le point d'équilibre (le partage) dépend du débit initial de la source 4. Plus ce débit est proche du débit optimal plus l'équité est atteinte, et inversement (débit initial dans ERICA est $\mu_1/N$).




(a) Ancien ERICA

(b) Point Fixe

FIG. 8.3: Débits en cellules/sec pour l'algorithme ERICA (ancien) et l'algorithme du Point Fixe.

Considérons l'algorithme du *Point Fixe*, décrit dans la section 7.2.1 du chapitre précédent, comme un autre exemple, pour lequel on tentera d'évaluer l'équité. On a

$$x_i(t) = x_i(t^-) + AIR * N_m \left(1 - \frac{x_i(t^-)}{PCR}\right) - \frac{y(t-\beta_i)}{Q} x_i(t^-) \tag{8.12}$$

Par conséquent,

$$|x_i(t) - x_j(t)| = \left| \left(1 - \frac{AIR * N_{rm}}{PCR}\right)(x_i(t^-) - x_j(t^-)) - \frac{y(t-\beta_i)x_i(t^-) - y(t-\beta_j)x_j(t^-)}{Q} \right|$$

Si le système est stable, alors $y(t) = \bar{y}$ (constante ou avec des variations négligeables pendant $|\beta_j - \beta_i|$), ce qui nous permet de déduire

$$\begin{aligned} |x_i(t) - x_j(t)| &= \left| \left(1 - \frac{AIR * N_{rm}}{PCR}\right)(x_i(t^-) - x_j(t^-)) - \frac{y(t-\beta_i)}{Q}(x_i(t^-) - x_j(t^-)) \right| \\ &= \left| \left(1 - \frac{AIR * N_{rm}}{PCR} + \frac{\bar{y}}{Q}\right)(x_i(t^-) - x_j(t^-)) \right| \end{aligned}$$

On avait vu que, quand la taille de la file se stabilise dans l'algorithme du point fixe ($\sum_{i=1}^{N} x_i(t) = \mu \; \forall t$), sa taille est égale à

$$\bar{y} = \frac{(N * PCR - \mu) AIR * N_{rm} * Q}{\mu * PCR}$$

Ainsi, on a

$$\begin{aligned} |x_i(t) - x_j(t)| &= \left| \left( 1 - \frac{AIR * N_{rm}}{PCR} + \frac{(N * PCR - \mu) AIR * N_{rm}}{\mu * PCR} \right) (x_i(t^-) - x_j(t^-)) \right| \\ &= \left| \left( 1 - \frac{N * AIR * N_{rm}}{\mu} \right) (x_i(t^-) - x_j(t^-)) \right| \\ &= \left| \frac{\mu - N * AIR * N_{rm}}{\mu} \right| |x_i(t^-) - x_j(t^-)| \end{aligned}$$

Finalement, on déduit que l'algorithme n'est pas équitable si

$$\left| \frac{\mu - N * AIR * N_{rm}}{\mu} \right| \geq 1 \Longrightarrow \mu \leq \frac{N * AIR * N_{rm}}{2} \tag{8.13}$$

Toutefois, il est facile de vérifier que le débit moyen d'une source avec cet algorithme est au moins égal à $\frac{AIR * N_{rm}}{2}$. Ceci est dû au fait que, quand ACR est zéro après la réception d'une cellule RM, à la réception de la prochaine cellule RM quel que soit le contenu de celle-ci, le débit de la source sera égal à $AIR * N_{rm}$. Ainsi, le débit de la source est au moins égal à $AIR * N_{rm}$ une fois sur deux cellules RM reçues, et si la relation (8.13) est satisfaite, alors la taille de la file tendra vers l'infini, du moment que le débit moyen d'entrée est supérieur à la bande passante disponible (le débit des sources oscille entre zéro (MCR $= 0$) et $AIR * N_{rm}$). Par conséquent, pour ces valeurs le schéma n'est pas stable. Il découle ainsi, que si l'algorithme est stable, alors il est équitable pour tous les paramètres (MCR $= 0$).

La Figure 8.3 (b) montre le débit alloué à chaque source dans l'exemple précédent, en utilisant l'algorithme du point fixe. Dans cet exemple, tous les commutateurs ont le même seuil de congestion $Q_i = Q = 1000$ *cellules*, $AIR = 125 \leq min(8000/(2 * 32), 16000/(3 * 32), 40000/(5 * 32) = 125$ et le débit minimal des sources est également le même pour toutes les sources $MCR = 10$ cellules/sec. L'algorithme est stable pour les délais d'aller retour de l'exemple en question. On peut constater que toute la bande passante est utilisée en un temps très court, et que le partage du débit sur l'ensemble des sources est presque idéal, aussi bien pour les deux premières secondes de simulation que pour le reste.

En suivant les mêmes étapes décrites dans cette section, pour l'équité, on peut facilement prouver qu'ERAQLES est également équitable pour $1 > s_0 > 0$, aussi bien que l'algorithme "pure rate matching", décrit dans la section précédente.

**Remarque 8.1.1** *Si un algorithme est équitable (satisfait (8.10)) et convergent (satisfait (8.3)), alors il est "max-min" équitable. En effet, s'il est équitable, alors tout couple de connexions non restreintes auront le même débit, par conséquent toutes les connexions non restreintes auront le même débit. D'autre part, si l'algorithme est convergent, alors la bande passante non utilisée par les sources restreintes ou silencieuses sera attribuée équitablement (puisque le schéma est équitable) sur l'ensemble des sources non restreintes.*

## 8.2 Conclusion et travaux futures

Dans le présent chapitre, nous avons présenté une méthode générale pour l'évaluation d'un algorithme basé sur le débit explicite et/ou la taille exacte de la file d'attente. Dans ce genre d'algorithmes, comme on l'a vu à travers les chapitres précédents de cette partie, il est très difficile d'estimer par calcul direct le temps de séjour, la taille de la file d'attente, et d'autres critères de performance. Aussi, d'autres critères essentiels que doit satisfaire tout algorithme ABR ont été présentés. Ces critères une fois satisfaits, permettent d'avoir un algorithme à la fois stable, permettant une utilisation maximale de la bande passante pour toutes les configurations du réseau, et enfin équitable, et par conséquent "max-min" équitable.

Nous avons par la même occasion analysé quelques algorithmes pour l'ABR, qui ne satisfont pas l'un des critères de performance cités ci-dessus. Par des simulations, nous avons validé les hypothèses considérées à travers ce chapitre, en particulier le fait de négliger le temps entre deux cellules RM.

Tous les algorithmes pour l'ABR vu jusqu'ici reposent sur l'équité au sens "max-min". Comme on l'a déjà souligné, l'équité au sens "max-min" est un choix très discutable. En effet, on peut voir une injustice dans le fait qu'une source d'un autre continent, se voit allouer la même bande passante qu'une source ne traversant qu'un seul commutateur, probablement à "deux pas" de sa destination. Cela dégraderait de façon dramatique le rendement du réseau en général, et par conséquent son revenu. Pour le revenu, il pourait être amorti, par la facturation en fonction de la localisation, mais le nombre d'utilisateurs satisfaits se verrait restreint (le rendement du réseau restera faible).

D'autre part, les réseaux de demain doivent avoir une grande flexibilité et offrir une variété de choix. En effet, il pourrait y avoir par exemple des utilisateurs qui souhaiteraient payer plus, pour plus de bande passante que d'autres utilisateurs "ordinaires", et avoir ainsi une garantie implicite d'un certain débit. Ceci peut arriver si un utilisateur souhaitait utiliser l'ABR pour la vidéo sur demande par exemple, ou d'autres "urgences".

Ainsi, les algorithmes ABR doivent inclure en même temps ces considérations de coût et de distance [37, 66, 67, 78]. Pour le coût, ils doivent trouver le juste milieu comme réponse aux deux questions suivantes : doit-on payer pour ce qu'on obtient ? ou doit-on obtenir pour ce qu'on paye ?. La réponse doit être les deux en même temps : on doit payer pour ce qu'on obtient, avec l'option d'avoir plus (que les autres, du moment qu'on ne peut rien garantir quant à la quantité) si on paye plus (que les autres). La distance quant à elle, peut être prise en compte lors d'allocation de débit, soit en fonction du nombre de commutateurs traversés, soit du délai d'aller retour de la source au commutateur (ou à sa destination). Pour ce dernier point, il nous semble plus pratique et plus prudent de considérer le nombre de commutateurs traversés, au lieu de la distance de la source au commutateur (la proposition de [37] par exemple).

Dans cette optique, nous pensons que l'équité proportionnelle [66, 67] prendra le dessus sur l'équité au sens "max-min". Les points de vues discutés dans les derniers paragraphes, constituent une orientation de nos axes de recherches dans un futur très proche avec la considération du multi-points également.

# Chapitre 9

# Conclusion Générale

Nous avons abordé au cours de cette thèse des problème liés au contrôle de flux dans les réseaux. Différentes méthodes ont été utilisées pour évaluer les performances de l'existant. Les constatations à travers des simulations nous ont permis de dégager quelques problèmes liés au fonctionnement de TCP. En considérant également des modèles fluides, nous avons pu évaluer les performances des différentes versions de TCP, ainsi que le contrôle de flux dans l'ABR.

Dans la première partie, nous nous sommes intéressés à l'évaluation du protocole TCP. A l'aide des traces obtenues par simulation, nous avons pu déceler quelques problèmes dans ce dernier, et proposer des solutions pour y pallier. Nous avons aussi proposé une méthode permettant de recouvrir des pertes de paquets ayant lieu plusieurs fois. Dans un deuxième temps, nous avons proposé un modèle simple utilisé dans beaucoup d'articles [16, 22, 72, 73], pour une approche analytique de TCP Reno et TCP Vegas.

En utilisant des approximations fluides, nous avons déterminé les critères de performance de chaque version, à savoir l'évolution de la fenêtre, le débit moyen, et le délai d'aller retour moyen. Ceci nous a permis en particulier de comparer les deux versions citées ci-dessus. Nous avons également montré que Vegas a le même comportement que Reno sous de fortes congestions, ou en présence de plusieurs connexions utilisant Vegas. Nous avons analysé le *Slow Start* de Vegas en le comparant à celui de Reno, et nous avons proposé une autre façon d'accroître la fenêtre durant cette phase. Nous avons également proposé des améliorations pour la détection de la bande passante disponible dans Vegas, qui permettent d'avoir moins de paquets dans les routeurs intermédiaires. Ces propositions nécessitent plus d'investigations et d'évaluation que nous essayons de faire.

Dans l'approche analytique de TCP, seule une connexion est prise en compte. Le cas de plusieurs connexions équivalentes (ayant le même délai d'aller retour) peut être résolu, en généralisant les résultats obtenus. Actuellement, nous essayons d'évaluer les performances de chaque version, en considérant le modèle décrit dans la section 4.4.1, avec de multiples connexions n'ayant pas forcément le même délai d'aller retour. Un autre problème qui nécessite beaucoup d'investigations est l'évaluation des performances de TCP Vegas au dessus du service ABR. Nous pensons que Vegas pourrait fonctionner en parfaite harmonie au dessus ce service, si les seuils de congestion de Vegas ($\alpha_{th}$ et $\beta_{th}$) sont choisis essentiellement en fonction du nombre de connexions partageant le même lien.

Dans la deuxième partie, nous avons analysé plusieurs algorithmes pour le service ABR dans les réseaux ATM (EFCI, EPRCA, Point Fixe, ERAQLES, ERICA+, OSU (l'algorithme à débit proportionnel), Uniform Tracking, Pure Rate matching), et d'autres peuvent être analysés en suivant les étapes décrites essentiellement dans le chapitre 8. Une méthode directe a été utilisée dans le chapitre 6, pour calculer les bornes sur les tailles des files d'attente. Nous avons ainsi montré que l'indication de congestion peut provoquer des files d'attente de grandes tailles si les connexions passent à travers plusieurs commutateurs.

La méthode de stabilité des solutions aux équations différentielles a été utilisée, pour étudier la stabilité de certains algorithmes basés sur le débit explicite et les tailles des files d'attentes. Ces algorithmes sont à priori connus comme stables, dans la mesure où la taille de la file d'attente est bornée (sauf dans certains cas pour l'algorithme du point fixe). Nous nous sommes alors intéressés à d'éventuelles oscillations de celle ci. Nous avons pu montrer en particulier, qu'ERICA+, ERAQLES, et les algorithmes basés sur la taille de la file d'attente ne sont pas stables pour de grands délais d'aller retour.

Ces mêmes délais constituent la difficulté majeure pour désigner un algorithme stable et efficace. On déduit ainsi, qu'un algorithme optimal doit impérativement prendre en compte ce paramètre. C'est ce qu'a tenté de faire ERAQLES en ajoutant (*resp.* amputant), à chaque allocation du débit, le débit susceptible de remplir (*resp.* vider) la file pendant un RTT. On a vu que cet algorithme n'est pas stable pour des bandes passantes assez grandes. Toutefois, d'après l'annexe C, si l'estimation de la bande passante se fait à des intervalles assez grands ($AN_{rm}$ grand), ERALQES devient alors stable. Seulement à ce moment là, la taille de la file d'attente aux instants initiaux risque d'être très large, et on risque également de sous-estimer la bande passante disponible, si une des connexions a un délai d'aller retour très grand.

Dans de nombreux travaux, ERICA+ est rapporté comme stable. Dans ces travaux, essentiellement des simulations ont été considérées. Notre étude nous a montré en particulier, que les exemples pris dans ces travaux ne permettaient apparemment pas de voir l'instabilité de cet algorithme, pour des délais assez larges. Elle nous a permis également de proposer des changements garantissant une stabilité pour tous les paramètres.

Dans le chapitre 8, une méthode pour évaluer l'équité et la convergence des algorithmes basés sur le débit explicite a été présentée. Nous avons ainsi évalué l'équité et la convergence de certains algorithmes par approche analytique. Des illustrations par des exemples de simulations ont été également données. Les simulations que nous avons effectuées tout au long de cette partie confirment les hypothèses qu'on a considérées, particulièrement le fait de négliger le délai entre deux cellules RM consécutives (modèles fluides).

# Troisième partie

# Annexe

# Annexe A

# TCP Vegas *vs* Reno

## A.1 Preuve du Théorème 4.3.1

Pour Reno, après chaque mini-slot ($\delta_i$), la $i$ème fenêtre est incrémentée d'un paquet dans la phase de *congestion*, ou doublée dans la phase du *slow start.* Ainsi, on a

$$W_n = \begin{cases} 2W_{n-1} & \text{si } W_n \leq W_{th} \\ W_{n-1} + 1 & \text{ailleurs} \end{cases} \tag{A.1}$$

Dans Vegas, durant la phase de *congestion*, la fenêtre peut également rester constante, ainsi on va considérer en plus le cas $W_n = W_{n-1}$. Dans la phase de *congestion* (quand la fenêtre augmente), aussi bien la phase du *slow start*, les paquets de la fenêtre $W_n$ sont envoyés pendant que les acquittements de ceux de la fenêtre $W_{n-1}$ sont reçus. Seul, le dernier paquets de $W_n$ est envoyé $1/C$ après la réception du dernier acquittement de $W_{n-1}$ (si $W_n > W_{n-1}$ et $C > 2\ Thp_{out}(W_n)$, $\forall n$). Par conséquent, le temps de transmission du “burst” (train) $W_n$ (i.e. le temps nécessaire pour transmettre les paquets de la fenêtre $W_n$), est le temps de réception des acquittements du burst (train) $W_{n-1}$ plus $1/C$ (i.e. le temps nécessaire de recevoir (servir) les acquittements des paquets de la fenêtre $W_{n-1}$). On peut alors écrire

$$W_n\ Thp_{in}(W_n)^{-1} = W_{n-1} Thp_{out}(W_{n-1})^{-1} + \frac{1}{C} \tag{A.2}$$

Puisque $C >> 1$, de l'équation (A.2), on a

$$Thp_{in}(W_n) \cong \frac{W_n}{W_{n-1}} Thp_{out}(W_{n-1}). \tag{A.3}$$

On utilisera l'approximation fluide suivante : On remplacera le débit d'entrée (arrivées) instantané durant un burst, par le débit moyen pris sur le burst, et on fera de même pour le débit de sortie (services). Durant la transmission d'un burst, (fenêtre) on a

$$Thp_{out}(W_n) = \mu \frac{Thp_{in}(W_n)}{Thp_{in}(W_n) + \lambda} \tag{A.4}$$

qui reflète le fait que, durant la transmission d'un burst, la file n'est pas vide (propriété **Q0**), et le débit de sortie de chaque source est proportionnel à son débit d'entrée. Ceci, est aussi vérifié, quand il n'y a pas de périodes de silence (la file ne se vide pas).

*(i)* Considérons un slot $n$ donné, durant lequel la fenêtre reste constante ($W_n = W_{n-1}$, dans Vegas). Alors, on a $Thp_{in}(W_n) = Thp_{out}(W_{n-1})$, en combinant ceci avec l'équation (A.4), on obtient $Thp_{in}(W_n) = Thp_{out}(W_n) = \mu - \lambda$. Le premier paquet de $W_n$ trouve une file vide si et seulement si, le temps qu'il met pour atteindre la file (routeur), de l'instant de sa transmission, excède le temps de service des paquets de $W_{n-1}$ restant dans la file (routeur). Ceci est équivalent à

$$\begin{aligned}&\tau_1 \geq \left(W_{n-1} - \left(\frac{\tau}{2} + \tau_2\right) Thp_{out}(W_{n-1}) - \frac{Thp_{out}(W_{n-1})}{\mu}\right) Thp_{out}(W_{n-1})^{-1}\\ &\Longleftrightarrow W_n \leq T(\mu - \lambda)\end{aligned}$$

Le nombre de paquets contrôlés dans la file, quand elle est tout le temps non vide ($W_n > T(\mu - \lambda)$), est constant dans le temps tant que la fenêtre reste constante. Ceci est due au fait que le débit total d'entrée est égale au débit de sortie $\mu$, et le flux de paquets est continue dans le temps (il n'y a pas de périodes de silence). Ainsi, tous les paquets de $W_n$ trouvent une file d'attente de taille $Q(W_n)$, telle que

$$Q(W_n)\frac{Thp_{in}(W_{n-1})}{Thp_{in}(W_{n-1}) + \lambda} = \left(W_{n-1} - \tau Thp_{out}(W_{n-1}) - \frac{Thp_{out}(W_{n-1})}{\mu}\right) \tag{A.5}$$

Cela veut dire que le nombre de paquets en transit, juste avant de transmettre le premier paquet de $W_n$, égale à $W_{n-1}$. Le premier paquet de $W_n$ est transmis, une fois que l'acquittement du premier paquet de $W_{n-1}$ est reçu. A cet instant, le nombre de paquets de $W_{n-1}$ qui sont servis égale à $\left(\tau_2 + \frac{\tau}{2}\right) Thp_{out}(W_{n-1}) - Thp_{out}(W_{n-1})/\mu$. Le premier paquet de $W_n$ met $\tau_1$ pour atteindre la file (routeur), et durant ce temps, $\tau_1$ $Thp_{out}(W_{n-1})$ paquets de $W_{n-1}$ seront servis. Alors, le nombre de paquets contrôlés restant dans la file à l'arrivée du premier paquet de $W_n$, qui est proportionnel au débit d'entrée $Thp_{in}(W_{n-1})$, égale à $W_{n-1}$ moins le nombre de paquets de $W_{n-1}$ qui sont, déjà, servis.

Puisque, $Thp_{out}(W_n) = Thp_{out}(W_{n-1}) = \mu - \lambda$, de l'équation (A.5) on obtient

$$Q(W_n) = \left(\frac{W_n}{\mu - \lambda} - T\right)\mu.$$

Finalement, on a

$$\delta_n = \overline{RTT}_n = T + \frac{Q(W_n)}{\mu} = \frac{W_n}{\mu - \lambda} = T + \epsilon_{W_n}. \tag{A.6}$$

Pour $W_n \leq T(\mu - \lambda)$, $\delta_n = \overline{RTT}_n = T$.

La fenêtre de Vegas n'excède pas $W^v_{max}$, telle que $Q(W^v_{max}) = B$, d'où découle (4.3). Il est facile, en effet, de vérifier que pour $W^v_{max}$, $\epsilon_{W^v_{max}} = B/\mu$.

Dans la phase de *congestion*, pour Reno et Vegas, quand la fenêtre augmente, $Thp_{in}(W_n)$ et $Thp_{out}(W_n)$ sont obtenus de (A.3) et (A.4), en remplaçant $W_{n-1}$ par $W_n - 1$. Ceci donne, approximativement,

$$Thp_{in}(W_n) \cong \mu - \lambda + \frac{\mu}{W_n - 1} \quad \text{et} \quad Thp_{out}(W_n) \cong \mu - \lambda + \frac{\lambda}{W_n}$$

Puisqu'on transmet un paquet pour chaque acquittement reçu, et seulement pour le dernier acquittement que deux paquets sont transmis, on peut supposer comme mentionné ci-dessus, que le débit est constant sur une génération ("burst" ou "train"). Le premier paquet de $W_n$ trouve une file vide, si et seulement si

$$
\begin{aligned}
&\tau_1 \geq \left( W_{n-1} - \left( \frac{\tau}{2} + \tau_2 \right) Thp_{out}(W_{n-1}) - \frac{Thp_{out}(W_{n-1})}{\mu} \right) Thp_{out}(W_{n-1})^{-1} \\
&\Longleftrightarrow W_n - 1 \leq T(\mu - \lambda) + \frac{T\lambda}{W_{n-1}} \cong T(\mu - \lambda)
\end{aligned}
$$

Pour $W_n - 1 \leq T(\mu - \lambda)$ (le premier paquet de $W_n$ trouve une file vide, ou la file se vide après le service de chaque génération), le *temps d'attente* du $i$ème ($i = 0 \cdots W_n - 1$) paquet est donné par

$$
i \left( Thp_{out}(W_n)^{-1} - Thp_{in}(W_n)^{-1} \right) = \frac{i}{(W_n - 1)(\mu - \lambda) + \mu}
$$

Le $(W_n - 1)$ème paquet est transmis $1/C$ après le $(W_n - 2)$ème paquet, alors

$$
\begin{aligned}
\delta_n &= T + \frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \mu} + \frac{1}{C} \cong T + \frac{1}{\mu - \lambda} \\
\text{et} \quad \overline{RTT}_n &= T + \frac{1}{W_n} \left( \sum_{i=0}^{W_n - 1} \frac{i}{(W_n - 1)(\mu - \lambda) + \mu} \right) \cong T
\end{aligned}
$$

$W_{max}$ est la fenêtre pour laquelle le "temps de séjour" du dernier paquet ($(W_n - 1)$ème) égale à $T + B/\mu$, d'où on obtient la première équation de (4.4).

Quand la file est tout le temps non vide ($W_n > T(\mu - \lambda)$), on a

$$
\delta_n = W_n Thp_{out}(W_n)^{-1} = \frac{W_n}{\mu - \lambda} \left( \frac{1}{1 + \frac{\lambda}{(\mu - \lambda) W_n}} \right) \cong \frac{W_n}{\mu - \lambda}.
$$

Pour le "temps de séjour" moyen sur une fenêtre, dans la phase de *congestion* (quand la fenêtre augmente), on remarque que les paquets de la $n$ème fenêtre $W_n$ sont transmis, pendant que les acquittements de la fenêtre $W_n - 1$ sont reçus. Soit $t_0 = 0$ l'instant de réception de l'acquittement du premier paquet de $W_{n-1}$, alors l'instant de transmission $t_s$ et de réception $t_r$ du $i$ème paquet de $W_n$ ($i = 0 \cdots W_n - 2$) sont, respectivement, donnés par

$$
\begin{aligned}
t_s &= i \frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \lambda} \\
t_r &= (W_n - 2) \frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \lambda} + (i + 1) \frac{W_n}{W_n(\mu - \lambda) + \lambda}
\end{aligned}
$$

Le "temps de séjour" du paquet $i$, est alors

$$
t_r - t_s = (W_n - 1) \frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \lambda} + \frac{(i + 1)\lambda}{((W_n - 1)(\mu - \lambda) + \lambda)(W_n(\mu - \lambda) + \lambda)}.
$$

Le dernier paquet $(W_n - 1)$ème est transmis $1/C$ après le $(W_n - 2)$ème paquet, et son acquittement revient à la source $Thp_{out}(W_n)^{-1}$ après celui du $(W_n - 2)$ème, son "temps de séjour" est alors donné par

$$
t_r - t_s = W_n \frac{W_n}{W_n(\mu - \lambda) + \lambda} - \frac{1}{C}.
$$

Finalement, on obtient

$$\overline{RTT}_n = \frac{1}{W_n} \sum_{i=0}^{i=W_n-1} RTT_i \cong \frac{W_n(W_n-1)}{W_n(\mu-\lambda)+\lambda} \tag{A.7}$$

$W_{max}$ est la taille de la fenêtre, pour laquelle le "temps de séjour" du dernier paquet $((W_{max}-1)$ème) est égale à $T + B/\mu$, d'où découle (Eq 2 de) (4.4).

*(ii)* Dans la phase du *slow start*, on a $Thp_{in}(W_n) = 2Thp_{out}(W_{n-1}) = 2Thp_{out}(W_n/2)$.
Soit

$$Thp_{in}(W_n) = 2Thp_{out}(W_n/2) = 2\mu - \lambda + f(W_n), \tag{A.8}$$

la solution de l'équation (A.3) et (A.4). De l'équation (A.4) et (A.8), on obtient

$$f(W_n) = \frac{\lambda f(W_{n-1})}{2\mu + f(W_{n-1})} \tag{A.9}$$

La solution de cette équation dépend des conditions initiales (qui dépendent de $C$). Puisque, après réception du premier acquittement, on envoie deux paquets séparés de $1/C$, on a ainsi $Thp_{in}(W_1) = Thp_{in}(2) = C$. En combinant ceci avec l'équation (A.8), on obtient $f(W_1) = f(2) = C - 2\mu + \lambda$. Par conséquent, de (A.9), il découle, par récurrence, que pour $C \geq 2\mu - \lambda$, $f(W_n) \geq 0 \; \forall n$ et

$$f(W_n) = \frac{\lambda f(W_{n-1})}{2\mu} \frac{1}{1 + \frac{f(W_{n-1})}{2\mu}} \leq \frac{\lambda f(W_{n-1})}{2\mu} = \left(\frac{\lambda}{2\mu}\right)^2 f(W_{n-2}) = \cdots = \left(\frac{\lambda}{2\mu}\right)^{n-1} f(W_1)$$

Puisque $\lambda < 2\mu$, on a $f(W_n) \longrightarrow 0$ quand $W_n \longrightarrow +\infty$. Par conséquent, pour $C \geq 2\mu - \lambda$, la solution converge vers $Thp_{in}(W_n) = 2Thp_{out}(W_n/2) = 2\mu - \lambda$.

Le premier paquet de la $n$ème fenêtre $(W_n)$ trouve une file vide, si et seulement si

$$\begin{aligned} \tau_1 &\geq \left(\frac{W_n}{2} - \left(\frac{\tau}{2} + \tau_2\right) Thp_{out}(W_n/2) - \frac{Thp_{out}(W_n/2}{\mu}\right) Thp_{out}(W_n/2)^{-1} \\ \Longleftrightarrow & \frac{W_n}{2} \leq T(\mu - \lambda/2). \end{aligned}$$

– Quand le premier paquet de la fenêtre $n$ trouve une file vide $\left(\frac{W_n}{2} \leq T(\mu - \lambda/2)\right)$

Le $RTT$ ("temps de séjour") du paquet $2i-2$, $i = 1 \cdots W_n/2$ (les paquets sont numérotés de 0 à $W_n - 1$), excède $T$ de

$$\frac{2i-2}{\mu - \lambda/2} - \frac{i-1}{\mu-\lambda/2} = \frac{i-1}{\mu-\lambda/2},$$

qui est le *temps d'attente*; l'instant de service moins l'instant d'arrivée à la file. Le *temps d'attente* du paquet $2i-1$ est égale à

$$\frac{2i-1}{\mu-\lambda/2} - \frac{i-1}{\mu-\lambda/2} - \frac{1}{C} = \frac{i}{\mu-\lambda/2} - \frac{1}{C}. \tag{A.10}$$

Par conséquent, le "temps de séjour" moyen sur la fenêtre $W_n$, dans la phase du *slow start*, pour Reno est donné par

$$\overline{RTT}_n = T + \frac{1}{W_n} \sum_{i=1}^{W_n/2} \left(\frac{2i-1}{\mu-\lambda/2} - \frac{1}{C}\right)$$

d'où découle (4.6). $\delta_n$ est égale au "temps de séjour" du dernier paquet de la fenêtre $W_n$, plus $1/C$ (car il est transmis $1/C$ après réception de l'acquittement du dernier paquet de $W_{n-1}$). Le "temps de séjour" du dernier paquet est le délai $T$ plus le "temps d'attente" calculé dans (A.10), alors

$$\delta_n = \left(T + \frac{W_n}{2\mu - \lambda} - \frac{1}{C}\right) + \frac{1}{C} = T + \frac{W_n}{2\mu - \lambda}$$

– Quand la file n'est pas vide ($T(\mu - \lambda/2) < W_n/2$)

On a $\delta_n = W_n Thp_{out}(W_n)^{-1} = \dfrac{W_n}{\mu - \lambda/2}$.

Les paquets de la fenêtre $W_n$ sont transmis, une fois que la source commence à recevoir les acquittements des paquets de $W_{n-1} = W_n/2$. Soit $t_0 = 0$, l'instant d'arrivée de l'acquittement du premier paquet transmis dans la précédente fenêtre ($W_n/2$), et $t_s$ *resp.* $t_r$, l'instant de transmission d'un paquet *resp.* d'arrivée de son acquittement. Le paquet $2i - 2$ ($i = 1 \cdots W_n/2$, les paquets de $W_n$ sont numéroté de 0 à $W_n - 1$), est transmis quand on reçoit l'acquittement du paquet $i$ de $W_n/2$, alors l'instant de transmission est $t_s = \dfrac{i-1}{\mu - \lambda/2}$, et puisque $Thp_{out}(W_n)^{-1} = \mu - \lambda/2$, son acquittement sera donc reçu $(2i-2)Thp_{out}(W_n)^-1$ après la réception de l'acquittement du premier paquet de $W_n$, dont l'instant d'arrivée est $(W_n/2)Thp_{out}(W_n/2)^{-1}$, on a alors

$$t_r = \frac{W_n/2 + 2i - 2}{\mu - \lambda/2} ,$$

et son RTT est donc égale à

$$t_r - t_s = \frac{W_n/2 + i - 1}{\mu - \lambda/2}.$$

Le paquet $2i-1$ est transmis $1/C$ après le paquet $2i-2$ (on transmet deux paquets pour chaque acquittement), on a ainsi $t_s = \dfrac{i-1}{\mu - \lambda/2} + \dfrac{1}{C}$, et son acquittement sera reçu $Thp_{out}(W_n)-1$ après celui de $2i - 2$, alors

$$t_r = \frac{W_n/2 + 2i - 1}{\mu - \lambda/2} ,$$

et son RTT est

$$t_r - t_s = \frac{W_n/2 + i}{\mu - \lambda/2} - \frac{1}{C}$$

Finalement, on obtient

$$\overline{RTT}_n = \frac{1}{W_n} \sum_{i=1}^{W_n/2} \left( \frac{2W_n - 2}{2\mu - \lambda} + \frac{2i}{\mu - \lambda/2} - \frac{1}{C} \right) = \frac{3W_n}{4\mu - 2\lambda} - \frac{1}{2C}$$

*(iii)* Pour Vegas, dans la phase du *slow start*, on distingue deux cas :
1) Quand la taille de la fenêtre est doublée (un $\delta$ sur deux), dans ce cas on transmet deux paquets pour chaque acquittement reçu.
2) Quand la fenêtre reste constante, on transmet un paquet pour chaque acquittement reçu.

Les deux cas sont exécutés de façon alternative, dans le temps. Soit $Thp_{in}(W_n)$ *resp.* $Thp_{out}(W_n)$ le débit de transmission *resp.* de service (de réception des acquittement) des paquets de $n$ème fenêtre $W_n$, dans le premier cas, et $\widetilde{Thp}_{in}(W_n)$ *resp.* $\widetilde{Thp}_{out}(W_n)$ leurs correspondants dans le deuxième cas. Dans le premier cas, deux paquets sont transmis pour chaque acquittement d'un paquet transmis dans le deuxième cas, de (A.3) on obtient alors

$$Thp_{in}(W_n) = 2\widetilde{Thp}_{out}(W_n/2) \tag{A.11}$$

Dans le deuxième cas, un paquet est transmis à chaque réception d'un acquittement d'un paquet transmis dans le premier cas, on a donc

$$\widetilde{Thp}_{in}(W_n) = Thp_{out}(W_n) \tag{A.12}$$

On a également de (A.4)

$$Thp_{out}(W_n) = \mu \frac{Thp_{in}(W_n)}{Thp_{in}(W_n) + \lambda} \qquad \text{et} \qquad \widetilde{Thp}_{out}(W_n) = \mu \frac{\widetilde{Thp}_{in}(W_n)}{\widetilde{Thp}_{in}(W_n) + \lambda} \tag{A.13}$$

En combinant (A.11), (A.12) et (A.13) on obtient

$$Thp_{in}(W_n) = 2\mu \frac{Thp_{out}(W_n/2)}{Thp_{out}(W_n/2) + \lambda} = 2\mu \frac{\mu Thp_{in}(W_n/2)}{(\mu+\lambda)Thp_{in}(W_n/2) + \lambda^2} \tag{A.14}$$

$$\text{Soit} \quad Thp_{in}(W_n) = \frac{2\mu^2 - \lambda^2}{\mu + \lambda} + f_1(W_n),$$

la solution de l'équation (A.14). Puisque $f_1(W_1) = f_1(2) = C$, alors pour $C \geq \dfrac{2\mu^2 - \lambda^2}{\mu + \lambda}$, on a (la fenêtre dans le mini-slot précédent reste constante, par conséquent $W_n = 2W_{n-2}$ et $W_{n+1} = W_n$)

$$f_1(W_n) = \left(\frac{\lambda}{2\mu}\right)^2 f_1(W_n/2) \frac{1}{1 + \frac{\mu+\lambda}{2\mu^2} f_1(W_n/2)} \leq \left(\frac{\lambda}{2\mu}\right)^2 f_1(W_{n-2}) = \cdots = \left(\frac{\lambda}{2\mu}\right)^{n-2} f_1(W_2)$$

Puisque $\lambda < 2\mu$, on a $f_1(W_n) \longrightarrow 0$ quand $W_n \longrightarrow +\infty$. Ainsi, pour $C \geq \dfrac{2\mu^2 - \lambda^2}{\mu + \lambda}$, la solution converge vers $Thp_{in}(W_n) = \frac{2\mu^2 - \lambda^2}{\mu + \lambda}$. De (A.11), (A.12) et (A.13) on obtient

$$Thp_{in}(W_n) = \frac{2\mu^2 - \lambda^2}{\mu + \lambda} \Longrightarrow Thp_{out}(W_n) = \frac{2\mu^2 - \lambda^2}{2\mu + \lambda} \tag{A.15}$$

$$\Longrightarrow \widetilde{Thp}_{in}(W_n) = \frac{2\mu^2 - \lambda^2}{2\mu + \lambda} \Longrightarrow \widetilde{Thp}_{out}(W_n) = \frac{\mu^2 - \lambda^2/2}{\mu + \lambda} \tag{A.16}$$

Quand la taille de la fenêtre est doublée, le premier paquet de $W_n$ trouve une file vide ($W_{n-1} = W_n/2$), si et seulement si

$$\tau_1 \geq \left( \frac{W_n}{2} - \left(\frac{\tau}{2} + \tau_2\right) \widetilde{Thp}_{out}(W_n/2) - \frac{\widetilde{Thp}_{out}(W_n/2)}{\mu} \right) \widetilde{Thp}_{out}(W_n/2)^{-1}$$

$$\Longleftrightarrow \frac{W_n}{2} \leq T \times \widetilde{Thp}_{out}(W_n/2) = \frac{\mu^2 - \lambda^2/2}{\mu + \lambda} T,$$

et quand la fenêtre reste constante, le premier paquet de $W_n$ trouve une file vide ($W_n = W_{n-1}$), si et seulement si

$$\tau_1 \geq \left(W_n - \left(\frac{\tau}{2} + \tau_2\right) Thp_{out}(W_n) - \frac{Thp_{out}(W_n)}{\mu}\right) Thp_{out}(W_n)^{-1}$$
$$\Longleftrightarrow W_n \leq T \times Thp_{out}(W_n) = \frac{2\mu^2 - \lambda^2}{2\mu + \lambda} T$$

Une fois que les différents débits de transmission et de réception sont calculés, dans chaque cas il est facile de calculer les autres mesures de performance $\overline{RTT}_n$, $\delta_n$ ...etc, en suivant le même raisonnement et les mêmes étapes suivies précédemment, dans le cas de Reno. ∎

## A.2 Preuve du Corollaire 4.3.1

*(i)* La fenêtre de Vegas se stabilise quand elle atteint la taille $W_{vmax}$, telle que

$$\frac{W_{vmax}}{T} - \frac{W_{vmax}}{T + \epsilon_{W_{vmax}}} > \frac{\alpha}{T} \quad \text{équivalent à} \quad \epsilon_{W_{vmax}} > \frac{\alpha T}{W_{vmax} - \alpha} \tag{A.17}$$
$$\text{et} \quad \frac{W_{vmax}}{T} - \frac{W_{vmax}}{T + \epsilon_{W_{vmax}}} \leq \frac{\beta}{T} \quad \text{équivalent à} \quad \epsilon_{W_{vmax}} \leq \frac{\beta T}{W_{vmax} - \beta} \tag{A.18}$$

où $T + \epsilon_{W_{vmax}}$ est le *temps de séjour* moyen quand la $n$ème fenêtre $W_n$ est égale à $W_{vmax}$. $W_{vmax}$ est alors la solution des inégalités suivantes (après substitution de (A.6) dans (A.17) et (A.18), on obtient respectivement)

$$\left(\frac{W_{vmax}}{\mu - \lambda} - T\right) \geq \frac{\alpha T}{W_{vmax} - \alpha} \tag{A.19}$$
$$\text{et} \quad \left(\frac{W_{vmax}}{\mu - \lambda} - T\right) \leq \frac{\beta T}{W_{vmax} - \beta} \tag{A.20}$$

Comme $\beta - \alpha = 2$, (4.16) est obtenue en résolvant ces deux inégalités.

*(ii)* Dans la phase du *slow start*, la fenêtre de Vegas est doublée un mini-slot ($\delta$) sur deux, si $(Expected - Actual)T \leq \gamma$. $\gamma$ est, généralement, pris égale à 1, on a alors $\epsilon_{W_{th}} > \dfrac{T\gamma}{W_{th} - \gamma}$.
On obtient, après substitution de (4.8), Si $T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_{th}/2$,

$$W_{th}^2 - \left(\gamma + \frac{4\mu^2 - 2\lambda^2}{2C\mu} - \frac{\lambda}{\mu}\right) W_{th} - \gamma \frac{(2TC - 1)(4\mu^2 - 2\lambda^2) + 2C\lambda}{2C\mu} > 0$$
$$\text{Si} \quad T\frac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_{th}/2, \quad \text{et}$$
$$W_{th}^2 - \left(\frac{(3\gamma\mu + (2\gamma + 1)\lambda)C + (2CT + 1)(2\mu^2 - \lambda^2)}{C(3\mu + 2\lambda)}\right) W_{th} + \gamma \left(\frac{C\lambda + 2\mu^2 - \lambda^2}{C(3\mu + 2\lambda)}\right) > 0$$
sinon.

En résolvant ces deux inégalités (comme deux égalité au lieu de deux inégalités) on obtient (4.18). Notons que $W$ est doublée un $\delta$ sur deux, telle que $W_{th}$ soit égale au plus petit entier, puissance de 2 et satisfaisant $W > Min_{th}$. D'où, on obtient (4.17). ∎

## A.3 Preuve du Corollaire 4.3.2

Dans le cas de Vegas, comme on a considéré que la fenêtre se stabilise en effet, et puisque la durée de la phase du *slow start* est négligeable, alors on a $\overline{Thp} = \mu - \lambda$. Par conséquent, le “temps de séjour” moyen par paquet est $\overline{rtt} = \dfrac{W_{vmax}}{\mu - \lambda}$.

Il est toujours facile de calculer le nombre de paquets transmis et le “temps de séjour” moyen, une fois que le comportement transitoire est exactement défini. Pour simplifier les calculs, on a supposé l'hypothèse (ii) pour Reno, où les pertes successives ne sont pas prises en compte [73]. On a supposé également qu'il y a au plus une perte dans une fenêtre (génération), de telle sorte que la phase du *slow start* soit exécutée uniquement quelques fois au début d'une session. En d'autres termes, l'étude du comportement transitoire de chaque algorithme est possible, dès que son comportement est à priori fixé. Dans ce qui suit, on se basera sur l'étude de l'état stationnaire. En effet, puisque le *slow start* n'est exécuté que quelques fois (au plus trois fois) au début d'une session, et que sa durée est relativement courte, alors cette phase peut être négligée, et seul le comportement périodique est “intéressant” (la phase de *congestion*).

Dans la phase de *congestion*, on a à calculer le nombre de paquets transmis ($N_1$) (avec succès), le temps nécessaire pour que ces données soient acquittées ($T_1$) et le “temps de séjour” moyen (par paquet) ($R_1$), du début d'un cycle donné ($W = (W_{max} + 1)/2$) jusqu'à ce qu'il ait une perte ($W = W_{max} + 1$). On distingue deux cas :

– $(W_{max} + 1)/2 > T(\mu - \lambda)$ (la file est tout le temps non vide), on a :

$$T_1 = \sum_{i=(W_{max}+1)/2}^{W_{max}+1} \delta_i = \sum_{i=(W_{max}+1)/2}^{W_{max}+1} \frac{i}{\mu - \lambda} \quad \text{et} \quad N_1 = \sum_{i=(W_{max}+1)/2}^{W_{max}+1} i$$

$$R_1 = \frac{1}{N_1} \sum_{i=(W_{max}+1)/2}^{W_{max}+1} i \times \overline{RTT}_i = \frac{1}{N_1} \sum_{i=(W_{max}+1)/2}^{W_{max}+1} i \frac{i(i-1)}{i(\mu - \lambda) + \lambda} \cong \frac{1}{N_1} \sum_{i=(W_{max}+1)/2}^{W_{max}+1} \frac{i^2 - i)}{\mu - \lambda}$$

– $(W_{max} + 1)/2 < T(\mu - \lambda)$ (La file se vide après chaque cycle), on a

$$N_1 = N_{21} + N_{22} \qquad T_1 = T_{21} + T_{22} \qquad R_1 = \frac{N_{21}R_{21} + N_{22}R_{22}}{N_1}.$$

Où les indices 21 correspondent à la période où la file est vide, et 22 se réfèrent à la période où la file n'est pas vide. Durant la période où la file est vide, on a

$$N_{21} = \sum_{i=(W_{max}+1)/2}^{T(\mu-\lambda)} i \qquad \text{et} \qquad T_{21} = \sum_{i=(W_{max}+1)/2}^{T(\mu-\lambda)} \left(T + \frac{1}{\mu - \lambda}\right)$$

le “temps de séjour” moyen, puisque la file est tout le temps vide, est égale à $R_{21} = T$. Durant la deuxième période (la file n'est pas vide), on a

$$N_{22} = \sum_{i=T(\mu-\lambda)+1}^{W_{max}+1} i \qquad \text{et} \qquad T_{22} = \sum_{i=T(\mu-\lambda)+1}^{W_{max}+1} \frac{i}{\mu - \lambda}$$

$$R_{22} = \frac{1}{N_{22}} \sum_{i=T(\mu-\lambda)+1}^{W_{max}+1} i \times \overline{RTT}_i = \frac{1}{N_{22}} \sum_{i=T(\mu-\lambda)+1}^{W_{max}+1} i \frac{i(i-1)}{i(\mu - \lambda) + \lambda} \cong \frac{1}{N_{22}} \sum_{i=T(\mu-\lambda)+1}^{W_{max}+1} \frac{i^2 - i}{\mu - \lambda} \quad \blacksquare$$

# Annexe B

# Analyse de l'indication de congestion

## B.1 Analyse et preuve des résultats

### B.1.1 évolution de ACR

Dans notre cas ($ICR = \mu_N$, la file $N$ ne se vide jamais et les cellules RM n'ont aucune priorité sur les cellules de données), l'évolution de $ACR$ suit deux phases, selon que la congestion est détectée ou pas.

**Phase 1 :** Pas de congestion et la file $N$ non vide

Cette phase concerne (i) le premier cycle ($n = 0$) qui commence après que la première cellule RM retourne à la source ; on considère alors $ACR(T_N^0 - \sum_{\ell=1}^{N} \tau_\ell) = ACR(\tau) = \mu_N$, et (ii) tous les autres cycles en l'absence de congestion. L'évolution de l'$ACR$ dans cette phase est donnée par

$$\frac{dACR(t)}{dt} = AIR * \mu_N.$$

On obtient ainsi

$$ACR(t) - ACR(t_0) = AIR * \mu_N(t - t_0) \tag{B.1}$$

$t_0$ est l'instant initial de cette phase qui se termine avec la réception d'une cellule RM avec un bit $CI$ mis à 1.

**Phase 2 :** La congestion est détectée
Quand une congestion est signalée, $ACR(t)$ est réduit multiplicativement au dessus de MCR. Son évolution est donnée par

$$\frac{dACR(t)}{dt} = -\frac{\mu_N}{RDF} ACR(t)$$

Par conséquent,

$$ACR(t) = ACR(t_1)e^{-\frac{\mu_N}{RDF}(t-t_1)} \tag{B.2}$$

$t_1$ est l'instant initial de cette phase, qui correspond à l'instant de la fin de la phase 1.

### B.1.2 Analyse transitoire

On calcule tout d'abord $T_i^n$, l'instant où la file $i$ commence à se remplir (se construire). Par définition, on a

$$ACR\left(T_i^n - \sum_{\ell=1}^{i} \tau_\ell\right) = \mu_i$$

Puisque la file $i$ est supposée toujours non vide, l'évolution de $ACR(t)$ suit alors la phase une. Par conséquent

$$ACR\left(T_i^n - \sum_{\ell=1}^{i} \tau_\ell\right) = \mu_N + AIR * \mu_N \left(T_i^n - T_N^n + \sum_{\ell=i+1}^{N} \tau_\ell\right) = \mu_i,$$

D'où
$$T_i^n - T_N^n = \frac{\mu_i - \mu_N}{AIR * \mu_N} - \sum_{\ell=i+1}^{N} \tau_\ell \tag{B.3}$$

La file $i$ commence à se remplir la première, si et seulement si $T_i^n - T_j^n < 0, \forall j \neq i$. De l'équation (B.3), on obtient alors le Théorème 6.3.2 ($i$).

Des cellules RM avec un bit $CI$ à zéro, sont transmises par les commutateurs, jusqu'à l'instant ($T_{Q_i}^n$) où l'une des des tailles des files ($i$) excède son seuil de congestion ($Q_i$). On calculera cet instant, comme si aucune congestion n'est détectée (seule l'équation (B.1) est utilisée). Pour $T_{Q_i}^n$, on distingue deux cas :

**1)** $T_{Q_i}^n - \tau_i \leq T_{i-1}^n$

$$Q_i - Q_i^n = \int_0^{T_{Q_i}^n - T_i^n} AIR * \mu_N * x \; dx = \frac{1}{2} AIR * \mu_N (T_{Q_i}^n - T_i^n)^2$$

d'où on obtient

$$T_{Q_i}^n = \sqrt{\frac{2(Q_i - Q_i^n)}{AIR * \mu_N}} + T_i^n \tag{B.4}$$

Le cas (1) est équivalent à $\frac{\mu_{i-1} - \mu_i}{AIR * \mu_N} \geq \sqrt{\frac{2(Q_i - Q_i^n)}{AIR * \mu_N}}$, comme on peut facilement le vérifier en combinant l'équation (B.3) et (B.4).

**2)** Dans les autres cas, on a

$$Q_i - Q_i^n = \int_0^{T_{i-1}^n - T_i^n + \tau_i} AIR * \mu_N * x \; dx + (\mu_{i-1} - \mu_i)(T_{Q_i}^n + \tau_i - T_{i-1}^n)$$

après substitution de (B.3), on obtient

$$T_{Q_i}^n = \frac{Q_i - Q_i^n}{\mu_{i-1} - \mu_i} + \frac{\mu_{i-1} - \mu_i}{2 * AIR * \mu_N} + T_i^n \tag{B.5}$$

*La congestion est tout le temps due* au $i$ème commutateur, si et seulement si $T_{Q_i}^n + \sum_{\ell=1}^{i} \tau_\ell \leq T_{Q_j}^n + \sum_{\ell=1}^{j} \tau_\ell$, $\forall j \neq i$, d'où découle l'équivalence entre (ii.a) et (ii.b) dans la partie ($ii$) du Théorème 6.3.2. Ceci est obtenu, après substitution de l'équation (B.3) dans la précédente, et avoir remarquer que $T_{Q_i}^n = A_{i,n} - \frac{\mu_i - \mu_N}{AIR * \mu_N} + T_i^n$.

Le cas particulier est obtenu en remplaçant $i$ par $N$. Ainsi, il nous reste à prouver l'équivalence entre (ii.b.1) et (ii.c.1).

(ii.b.1) implique (ii.c.1) est triviale, puisque la condition doit être satisfaite dans tous les cycles ($\forall n$).
Maintenant on va prouver que $2\sum_{\ell=j+1}^{N}\tau_\ell \leq A_{j,0} - A_{N,0}\ \forall j \leq N \Longrightarrow 2\sum_{\ell=j+1}^{N}\tau_\ell \leq A_{j,n} - A_{N,n}\ \forall n, j$.

Puisque $Q_N^n \geq 0 = Q_N^0$ pour tout entier $n$, on a $A_{N,0} \geq A_{N,n}\ \forall n$. $2\sum_{\ell=j+1}^{N}\tau_\ell \leq A_{j,0} - A_{N,0}\quad \forall j < N$, implique que la congestion est due au $N$ème commutateur dans le premier cycle, i.e. la taille de la file $N$ atteint son seuil $Q_N$, pas plus tard que $\sum_{\ell=j+1}^{N}\tau_\ell$ avant que la taille de la file $j$ atteigne le sien ($Q_j$), pour tout $j < N$. Par conséquent, $Q_j^1 = 0$, $\forall j < N$ (en vertu du Lemme 6.3.1 **P2**). Il en découle, que $A_{j,1} = A_{j,0}$, $\forall j < N$, et donc $2\sum_{\ell=j+1}^{N}\tau_\ell \leq A_{j,1} - A_{N,0}\ \forall j < N$. En combinant ceci avec le fait que $A_{N,0} \geq A_{N,n}\ \forall n$, on obtient $2\sum_{\ell=j+1}^{N}\tau_\ell \leq A_{j,1} - A_{N,1}$, ce qui signifie que la congestion est due au $N$ème commutateur dans le deuxième cycle. En répétant cet argument, on obtient (ii.b.1).

Dans ce qui suit, on suppose que

- (**H1**) *La congestion est tout le temps due* au $N$ème commutateur ($T_{Q_N}^n + \sum_{\ell=1}^{N}\tau_\ell \leq T_{Q_j}^n + \sum_{\ell=1}^{j}\tau_\ell\ \forall j < N$)

La file $N$ est un *goulot au sens strict*, si et seulement si

$$T_{Q_N}^n + \sum_{\ell=1}^{N}\tau_\ell \leq T_i^n - \sum_{\ell=1}^{i}\tau_\ell,\ \forall i < N \tag{B.6}$$

Après substitution de (B.3), on obtient

$$2\sum_{\ell=1}^{N}\tau_\ell \leq \frac{\mu_i - \mu_N}{AIR * \mu_N} - A_{N,n} \quad \forall i < N.$$

Puisque $A_{N,n} < A_{N,0}$, $\forall n$, et que la condition doit être satisfaite dans le premier cycle en particulier, alors la relation ci-dessus est équivalente à

$$2\sum_{\ell=1}^{N}\tau_\ell \leq \frac{\mu_i - \mu_N}{AIR * \mu_N} - A_{N,0} \quad \forall i < N.$$

L'équivalence entre *(i.b)* et *(i.c)*, dans la partie *(i)* du Théorème 6.3.3, est obtenue en remarquant que $\mu_0 < \mu_1 < \cdots < \mu_{N-1} < \mu_N$, et qu'une fois de plus, la relation ci-dessus doit être satisfaite pour la première file $i = 1$ en particulier.

La partie *(i.1)* est obtenue en utilisant les mêmes arguments, en inversant les inégalités dans (B.6).

Pour la taille maximale de la file dans chaque commutateur, on considère les deux classes de commutateurs : les commutateurs basés EFCI et ceux basés ER. Soit $T_P^{n,i} \triangleq min(T_{i-1}^n - \sum_{\ell=1}^{i-1}\tau_\ell,\ T_{Q_N}^n + \sum_{\ell=1}^{N}\tau_\ell)$, et $ACR_{max}^n \triangleq min\left(PCR,\ ACR(T_{Q_N}^n + \sum_{\ell=1}^{N}\tau_\ell)\right)$

On suppose également que toutes les files se remplissent (se construisent) (*(i.1)* dans le Théorème 6.3.3 est vraie).

#### B.1.2.1 Commutateurs basés EFCI

La première cellule RM avec un bit $CI$ à 1, arrive à la source à l'instant $T_{Q_N}^n + \sum_{\ell=1}^{N} \tau_\ell$ (Les cellules RM "en amont" sont mise à jours à la détection d'une congestion). A partir de cet instant, l'évolution de $ACR(t)$ correspond à la seconde phase. Par conséquent, l'instant où $ACR(t) = \mu_i$, dans le $n$ème cycle en décroissant est donné par

$$T_{\mu_i}^n = \frac{RDF}{\mu_N} \log\left(\frac{ACR_{max}^n}{\mu_i}\right) + T_{Q_N}^n + \sum_{\ell=1}^{i} \tau_\ell$$

Soit $T_M^{n,i} \triangleq max(T_{\mu_i}^n,\ T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell)$

Puisque *la congestion est tout le temps due* au $N$ème commutateur (**H1**), alors

$$\begin{aligned}
Q_{max}^{i,n} &= Q_i^n + \int_{T_i^n - \sum_{\ell=1}^{i} \tau_\ell}^{T_P^{n,i}} (ACR(x) - \mu_i)\ dx + \int_{T_P^{n,i}}^{T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell} (\mu_{i-1} - \mu_i)\ dx \\
&\quad + \int_{T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell}^{T_M^{n,i}} (ACR(x) - \mu_i)\ dx \\
&= Q_i^n + \int_0^{T_P^{n,i} - T_i^n + \sum_{\ell=1}^{i} \tau_\ell} AIR * \mu_N * x\ dx + (\mu_{i-1} - \mu_i) \left(T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell - T_P^{n,i}\right) \\
&\quad + \int_0^{T_M^{n,i} - T_0^{n,i-1} + \sum_{\ell=1}^{i-1} \tau_\ell} \left(ACR(T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell) e^{-\frac{\mu_N}{RDF} x} - \mu_i\right)\ dx \\
&= Q_i^n + \frac{AIR * \mu_N}{2} \left(T_P^{n,i} - T_i^n + \sum_{\ell=1}^{i} \tau_\ell\right)^2 + (\mu_{i-1} - \mu_i) \left(T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell - T_P^{n,i}\right) \\
&\quad + \int_{T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell}^{T_M^{n,i}} (ACR(x) - \mu_i)\ dx \qquad \text{(B.7)}
\end{aligned}$$

A présent, on a besoin de calculer $T_0^{n,i}$. On a

$$\begin{aligned}
Q_{max}^{i,n} - Q_i^{n+1} &= \int_{T_M^{n,i}}^{T_0^{n,i} - \sum_{\ell=1}^{i} \tau_\ell} (\mu_i - ACR(x))\ dx \\
&= \int_0^{T_0^{n,i} - \sum_{\ell=1}^{i} - T_M^{n,i}} \left(\mu_i - ACR(T_M^{n,i})\ e^{-\frac{\mu_N}{RDF} x}\right)\ dx
\end{aligned}$$

Posons $y_i \triangleq T_0^{n,i} - \sum_{\ell=1}^{i} - T_M^{n,i}$, alors on a

$$Q_{max}^{i,n} - Q_i^{n+1} = \mu_i y_i + \frac{RDF}{\mu_N} ACR(T_M^{n,i}) \left(e^{-\frac{\mu_N}{RDF} y_i} - 1\right) \qquad \text{(B.8)}$$

En combinant l'équation (B.8) avec l'équation (B.7), et après quelques simplifications, on obtient ($Q_i^N = 0 \; \forall i < N$, et $C_{i,n} \triangleq T_P^{n,i} - T_i^n + \sum_{\ell=1}^{i} \tau_\ell$)

$$\begin{aligned} Q_{max}^{i,n} &= Q_i^n + \frac{AIR * \mu_N}{2} {C_{i,n}}^2 + \int_{T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell}^{T_M^{n,i}} (ACR(x) - \mu_i) \; dx \\ &+ (\mu_{i-1} - \mu_i) \left[ (T_0^{n,0} - T_P^{n.i}) + \sum_{j=1}^{i-1} \left( \frac{Q_{max}^{j,n}}{\mu_j} + (T_M^{n,j} - T_0^{n,j-1} + \sum_{\ell=1}^{j-1} \tau_\ell) \right. \right. \\ &\left. \left. + \frac{RDF * ACR(T_M^{n,j})}{\mu_N * \mu_j} \left( 1 - e^{-\frac{\mu_N}{RDF} \left( T_0^{n,j} - \sum_{\ell=1}^{j} \tau_\ell - T_M^{n,j} \right)} \right) \right) \right] \end{aligned} \tag{B.9}$$

Notons que $T_0^{n,0} = T_{Q_N}^n + \sum_{\ell=1}^{N} \tau_\ell$. On distingue alors le cas $i = 1$ et $i > 1$. Puisque $Q_i^n = 0 \; \forall n, \; i < N$, alors on a, après avoir remarquer que $B_{i,n} \triangleq T_{Q_N}^n + \sum_{\ell=1}^{N} \tau_\ell - T_P^{n,i}$

$$\begin{aligned} Q_{max}^{1,n} &= Q_1^n + \frac{AIR * \mu_N}{2} {C_{1,n}}^2 \\ &+ \int_{T_{Q_N}^n + \sum_{\ell=1}^{N} \tau_\ell}^{T_{\mu_1}^n} (ACR(x) - \mu_i) \; dx + (\mu_0 - \mu_1) (T_{Q_N}^n + \sum_{\ell=1}^{N} \tau_\ell - T_P^{n.1}) \\ &= Q_1^n + \frac{AIR * \mu_N}{2} {C_{1,n}}^2 + (\mu_0 - \mu_1) B_{1,n} + RDF * AIR * C_{1,n} \\ &- \frac{RDF * \mu_1}{\mu_N} \log \left( 1 + \frac{AIR * \mu_N}{\mu_1} C_{1,n} \right) \end{aligned} \tag{B.10}$$

d'où vient l'équation (6.2) dans le Théorème 6.3.3. Pour $i > 1$, l'intégrale dans la première ligne de l'équation (B.9) peut être négligée, car $T_M^{n,i} \approxeq T_0^{n,i-1} - \sum_{\ell=1}^{i-1} \tau_\ell \quad \forall i > 1$. Par conséquent, de l'équation (6.2), on a

$$\begin{aligned} Q_{max}^{i,n} &\approxeq Q_i^n + \frac{AIR * \mu_N}{2} {C_{i,n}}^2 + (\mu_{i-1} - \mu_i) \left[ (T_M^{n,1} - T_P^{n.i}) \right. \\ &\left. + \sum_{j=1}^{i-1} \left( \frac{Q_{max}^{j,n}}{\mu_j} + \frac{RDF * ACR(T_M^{n,j})}{\mu_N * \mu_j} \left( 1 - e^{-\frac{\mu_N}{RDF} \left( T_0^{n,j} - \sum_{\ell=1}^{j} \tau_\ell - T_M^{n,j} \right)} \right) \right) \right] \\ &\approxeq Q_i^n + \frac{AIR * \mu_N}{2} {C_{i,n}}^2 + (\mu_{i-1} - \mu_i) \left[ (T_{\mu_1}^n - T_P^{n.i}) \right. \\ &\left. + \sum_{j=1}^{i-1} \left( \frac{Q_{max}^{j,n}}{\mu_j} + \frac{RDF}{\mu_N} \left( e^{-\frac{\mu_N}{RDF} \left( T_M^{n,j} - T_{\mu_j}^n \right)} - e^{-\frac{\mu_N}{RDF} \left( T_0^{n,j} - \sum_{\ell=1}^{j} \tau_\ell - T_{\mu_j}^n \right)} \right) \right) \right] \\ &\approxeq Q_i^n + (\mu_{i-1} - \mu_i) \left[ \sum_{j=1}^{i-1} \left( \frac{Q_{max}^{j,n}}{\mu_j} + \frac{RDF}{\mu_N} \left( 1 - e^{-\frac{\mu_N}{RDF} \left( T_{\mu_{j+1}}^n - T_{\mu_j}^n \right)} \right) \right) \right. \\ &\left. + \left( \frac{RDF}{\mu_N} \log \left( \frac{ACR_{max}^n}{\mu_1} \right) + T_{Q_N}^n + \sum_{\ell=1}^{N} \tau_\ell - T_P^{n.i} \right) \right] + \frac{AIR * \mu_N}{2} {C_{i,n}}^2 \end{aligned} \tag{B.11}$$

De laquelle découle l'équation (6.3) dans le Théorème 6.3.3 . Le nombre de cellules restantes dans la file $N$, à la fin du cycle, est donné par

$$\begin{aligned}
Q_N^{n+1} &= Q_N + \int_{T^n_{Q'_N}-\sum_{\ell=1}^N \tau_\ell}^{T^n_{Q'_N}-\sum_{\ell=1}^N \tau_\ell+\tau} (ACR(x)-\mu_N)\ dx + \int_{T^n_{Q'_N}-\sum_{\ell=1}^N \tau_\ell+\tau}^{T_N^{n+1}-\sum_{\ell=1}^N \tau_\ell} (ACR(x)-\mu_N)\ dx \\
&= Q_N + ACR(T^n_{Q'_N} - \sum_{\ell=1}^N \tau_\ell)\frac{RDF}{\mu_N}\left(1-e^{-\frac{\mu_N}{RDF}\tau}\right) \\
&\quad -\mu_N * \tau - \frac{\left(\mu_N - ACR(T^n_{Q'_N}-\sum_{\ell=1}^N \tau_\ell)\ e^{-\frac{\mu_N}{RDF}\tau}\right)^2}{2*AIR*\mu_N}
\end{aligned} \tag{B.12}$$

Les inégalités (6.4), dans le Théorème 6.3.3, sont dérivées de l'équation (B.12), en utilisant le fait que $0 \le ACR(T^n_{Q'_N} - \sum_{\ell=1}^N \tau_\ell) \le \mu_N$.

#### B.1.2.2 Commutateurs basés ER

Pour les commutateurs basés ER, une fois que la congestion est détectée, l'équation (6.1) prend place (pour l'évolution de l'$ACR$). $ACR(t)$ est alors réduit à $ER_N < \mu_N$. La taille maximale de la file d'attente $i$ est atteinte, soit $\sum_{\ell=1}^i \tau_\ell$ après qu'une cellule RM avec un bit $CI$ à un est arrivée à la source, soit $\tau_i$ après que la file $i-1$ soit vide. Puisque, on a supposé que toutes les files se construisent (aucune file ne reste tout le temps vide), alors c'est le deuxième cas qui se produit. On a, alors

$$\begin{aligned}
Q_{max}^{i,n} &= Q_i^n + \int_{T_i^n-\sum_{\ell=1}^i \tau_\ell}^{T_P^{n,i}} (ACR(x)-\mu_i)\ dx + \int_{T_P^{n,i}}^{T_0^{n,i-1}-\sum_{\ell=1}^{i-1}\tau_\ell} (\mu_{i-1}-\mu_i)\ dx \\
&= Q_i^n + \int_0^{T_P^{n,i}-T_i^n+\sum_{\ell=1}^i \tau_\ell} AIR*\mu_N*x\ dx + (\mu_{i-1}-\mu_i)\left(T_0^{n,i-1} - \sum_{\ell=1}^{i-1}\tau_\ell - T_P^{n,i}\right) \\
&= Q_i^n + \frac{AIR*\mu_N}{2}\left(T_P^{n,i} - T_i^n + \sum_{\ell=1}^i \tau_\ell\right)^2 \\
&\quad + (\mu_{i-1}-\mu_i)\left(T_0^{n,i-1} - \sum_{\ell=1}^{i-1}\tau_\ell - T_P^{n,i}\right)
\end{aligned} \tag{B.13}$$

$T_0^{n,i}$ satisfait l'équation suivante :

$$\begin{aligned}
Q_{max}^{i,n} - Q_i^{n+1} &= \int_{T_0^{n,i-1}-\sum_{\ell=1}^{i-1}\tau_\ell}^{T_0^{n,i}-\sum_{\ell=1}^{i}\tau_\ell} (\mu_i - ER_N)dx \\
&= (\mu_i - ER_N)(T_0^{n,i} - T_0^{n,i-1} - \tau_i)
\end{aligned} \tag{B.14}$$

$$T_0^{n,i} = \frac{Q_{max}^{i,n} - Q_i^{n+1}}{\mu_i - ER_N} + T_0^{n,i-1} + \tau_i \tag{B.15}$$

Du moment que $T_0^{n,0} = T^n_{Q_N} + \sum_{\ell=1}^N$, de l'équation (B.15), on obtient alors

$$T_0^{n,i} = \sum_{\ell=1}^i \left(\frac{Q_{max}^{\ell,n} - Q_\ell^{n+1}}{\mu_\ell - ER_N} + \tau_\ell\right) + T^n_{Q_N} + \sum_{\ell=1}^N \tau_\ell \tag{B.16}$$

Puisque $Q_i^n = 0,\ \forall i < N$ et $Q_i^{tr}$ est obtenue dans le premier cycle, alors après substitution de $C_{i,n}$ et $B_{i,n}$, déjà définis, on obtient l'équation (6.5) dans le Théorème 6.3.3.

La file $N$ est le *goulot essentiel*, si et seulement si $Q_{max}^{i,n} < Q_i\ \forall i < N$, qui est équivalent à $Q_{max}^{i,0} < Q_i\ \forall i < N$. De l'équation (B.13), en remarquant que $T_P^{i,n} \leq T_0^{n,i-1} - \sum_{\ell=1}^{i-1}$, on obtient

$$Q_{max}^{i,n} \leq Q_i^n + \frac{AIR * \mu_N}{2} \left( \sum_{\ell=1}^{i-1} \frac{Q_{max}^{\ell,n}}{\mu_\ell - ER_N} + B_{i+1,n} \right)^2$$

la condition suffisante pour que $Q_{max}^{i,n} < Q_i\ \forall n,\ i < N$ est

$$Q_{max}^{i,n} \leq \frac{AIR * \mu_N}{2} \left( \sum_{\ell=1}^{i-1} \frac{Q_{max}^{\ell,n}}{\mu_\ell - ER_N} + B_{i+1,n} \right)^2 < Q_i\ \forall n,\ i < N$$

D'où découle (*iii*), dans le Théorème 6.3.3 (en remarquant que$Q_i^0 = 0,\ \forall i$).

Dans le cas des commutateurs ER, $Q_N^{n+1} = Q_N^n\ \forall n \geq 1$ ($Q_i^n = 0,\ \forall n, i < N$) ; comme on peut le constater ci-dessous, ceci est due au fait que $ACR(t) = ER_N$ est constant, dès que la congestion est détectée. $Q_N^n$ alors ne dépend pas de la taille maximale de la file durant le cycle, mais uniquement du seuil. Quand la taille de la file $N$ est au dessous de $Q_N$, la source est informée $\sum_{\ell=1}^{N} \tau_\ell + 2\tau_{N+1}$ après (les commutateurs ne peuvent pas mettre à jours le bit CI des cellules RM dans le cas de non congestion, si le bit était mis à un par la destination), en recevant une cellule RM avec un bit $CI$ mis à zéro, et commence ainsi à croître son débit ($ACR(t)$) selon la première phase (phase 1). Par conséquent, le temps séparant l'instant où la taille de la file est au dessous de $Q_N$, jusqu'à l'instant où la file $N$ commence de nouveau à se remplir, est donné par $T_N^{n+1} - T_{Q'_N}^n = \tau + \frac{\mu_N - ER_N}{AIR * \mu_N}$ (voir équation (B.1). On a donc

$$Q_N^{n+1} = Q_N + (ER_N - \mu_N)\ \tau + \int_0^{\frac{\mu_N - ER_N}{AIR * \mu_N}} ((ER_N - \mu_N) + AIR * \mu_N * x)\ dx, \qquad \text{(B.17)}$$

### B.1.3 Analyse stationnaire

Les conclusions qu'on a obtenues pour l'analyse transitoire sont valables pour $Q_i^n = 0$, cependant les formules qu'on a développées sont valables pour n'importe quelles tailles initiales des files, en particulier celles obtenues dans l'état stationnaire, dans le cas d'un comportement périodique (un régime périodique est, en effet, obtenu pour les commutateurs basés ER, puisque le reste des celles à la fin d'un cycle est constant. Pour les commutateurs basés EFCI, on a observé un comportement périodique dans toutes les simulations).

Dans ce qui suit, on ne va pas supposer l'existence d'un comportement périodique. Pour les commutateurs basés EFCI, on utilisera les formules développées pour l'état transitoire, dans le but de calculer des bornes sur les tailles maximales des files. On obtient ci-après une borne asymptotique supérieure $Q_i^{st}$ pour la taille maximale de la file $i$, dans l'état stationnaire, en utilisant les équations (B.10) et (B.11), dans lesquelles nous substituons $Q_N^n$ par une *borne asymptotique inférieure* de $Q_N^n$ :

$$Q_N^\infty \triangleq \underline{\lim}_{n \to \infty} (Q_N^n),$$

et en prenant $Q_i^n = 0\ \forall i < N$. Le fait, que la borne inférieure pour $Q_N^n$ engendre une borne supérieure pour la taille maximale de la file, découle du Théorème 6.3.1 et de la remarque

6.3.2. De façon similaire, on peut obtenir des bornes asymptotiques inférieures, sur les tailles maximales des files, en substituant dans (B.10) et (B.11) une borne asymptotique supérieure pour $Q_N^n$.

#### B.1.3.1 Commutateurs basés EFCI

De l'équation (B.12), on a

$$\begin{aligned} Q_N^\infty &= \lim_{n\to\infty}\left(Q_N + ACR\left(T^n_{Q'_N} - \sum_{\ell=1}^{N}\tau_\ell\right)\frac{RDF}{\mu_N}\left(1-e^{-\frac{\mu_N}{RDF}\tau}\right)\right. \\ &\quad \left. -\mu_N*\tau - \frac{\left(\mu_N - ACR\left(T^n_{Q'_N} - \sum_{\ell=1}^{N}\tau_\ell\right)\ e^{-\frac{\mu_N}{RDF}\tau}\right)^2}{2*AIR*\mu_N}\right) \end{aligned}$$

Puisque

$$\lim_{n\to\infty} ACR\left(T^n_{Q'_N} - \sum_{\ell=1}^{N}\tau_\ell\right) = \lim_{n\to\infty} \mu_N\ e^{-\frac{\mu_N}{RDF}\left(T^n_{\mu_N} - T^n_{Q'_N} + \sum_{\ell=1}^{N}\tau_\ell\right)} = 0,$$

et $Q_N^\infty \geq 0$, on déduit alors, l'équation (6.6) de l'équation (B.10) et l'équation (6.7) de l'équation (B.11). La preuve du Théorème 6.3.4 *(i)* est ainsi établie.

#### B.1.3.2 Commutateurs basés ER

Pour les commutateurs basés ER, on a $Q_N^n = Q_N^{n+1}\ \forall n \geq 1$. Le régime stationnaire est alors atteint juste après le premier cycle. Par conséquent, de l'équation (B.17), on obtient

$$Q_N^\infty = \lim_{n\to\infty}(Q_N^n) = \left(Q_N - (\mu_N - ER_N)\left(\tau + \frac{\mu_N - ER_N}{2*AIR*\mu_N}\right)\right)^+.$$

La preuve du Théorème 6.3.4 *(ii)* découle après substitution de $Q_N^\infty$ et (B.16) dans (B.13). ∎

# Annexe C

# Stabilité des algorithmes ABR

## C.1 Taille de la file d'attente dans ERAQLES et insuffisances

Dans le chapitre 7, on a supposé qu'ERAQLES a la connaissance exacte de la bande passante disponible et la répartit ainsi sur les différentes connexions, sans qu'il l'estime en fonction de la taille de la file d'attente. Cette supposition est motivée par quelques exemples de simulation donnés dans [83], où la bande passante disponible est considérée connue. En fait, le fait qu'ERAQLES estime la bande passante lui même en fonction de la taille de la file d'attente, lui permet dans certains cas d'éviter que celle-ci ne déborde, et ce par la correction de la bande passante estimée à chaque fois que la taille de la file dépasse le seuil $Q$ (toutes les $AN_{rm} = 512$ cellules RM traitées).

En effet, dans [83] il a été montré qu'en dehors des intervalles d'estimation de la bande passante, la taille de la file d'attente converge vers le seuil de congestion. Aussi, notre étude précédente n'est valable qu'à l'intérieur de ces intervalles d'estimation. Si ces intervalles sont de durées assez grandes , il est tout à fait vrai que la file d'attente peut déborder si la bande passante disponible est assez grande (voir le Lemme 7.3.1).

On peut constater que le temps de $AN_{rm}$ cellules RM peut être un intervalle assez grand. Également, $s_0$ doit avoir une limite inférieure pour assurer une équité assez rapide. La valeur de $s_0$ recommandée dans [83] est $1/N$, où $N$ est le nombre de connexions, ce qui fait que la bande passante allouée, à l'intérieur des intervalle d'estimation, est supérieure à la bande passante réelle.

Pour illustrer cela, prenons un exemple assez simple. Supposons le même modèle décrit dans le chapitre 7, avec $N = 2$, $s_0 = 0.5$. A chaque allocation de la bande passante, on a

$$x_i(t) = \left( (1-0.5) \frac{x_i(t-\tau_i)}{\sum_{\ell=1}^{N} x_\ell(t-\gamma_\ell-\beta_i)} + 0.5 \right) \left( C^* + \frac{h}{T}(Q - y(t-\beta_i)) \right)$$

Puisque les deux sources sont équivalentes, alors

$$x_1(t) + x_2(t) = 1.5 \left( C^* + \frac{h}{T}(Q - y(t-\beta_i)) \right)$$

Supposons qu'à cet instant les deux sources envoient à un débit $\mu/2$, où $\mu$ est le débit réel disponible. Supposons également, que la taille de la file d'attente a atteint $Q$ à cet instant. Il est tout à fait juste de supposer qu'au même instant, la valeur de la bande passante estimée est la valeur réelle $C^* = \mu$, du moment que $C^*$ augmente tant que la taille de la file n'atteint pas $Q$. Ainsi, le débit total alloué aux deux sources, pendant l'intervalle $\frac{AN_{rm}^2}{\mu}$ (au minimum), est $1.5\mu$. Ce qui implique que dans le prochain RTT, la file va se construire au delà de $Q$ à un débit de $\mu/2$ pendant $\frac{AN_{rm}^2}{\mu}$, et résultera d'une file d'attente égale à $Q+(AN_{rm}^2/2) = Q+8192$ (au minimum), ce qui justifie les conclusions du chapitre 7 sur ERAQLES.

Une autre insuffisance d'ERAQLES est l'utilisation du délai d'aller retour maximal. En cas d'existence d'une seule source, parmi les $n$ sources, avec un délai assez grand, celle-ci alors influe négativement sur tout le système dans la mesure où $h/T$ est assez petit, et si la valeur initiale de $C^*$ n'est pas bien estimée, elle le restera pour une longue période.

ERAQLES également est moins performant dans la situation suivante : considérons un réseau simple avec deux commutateurs en tandem. Supposons que les bandes passantes disponibles sont $\mu$ et $b\mu$ ($b \geq 2$), respectivement dans le premier et le deuxième commutateur. Supposons également qu'il y a une source qui traverse les deux commutateurs. La source, dans le cas optimal, transmet alors avec un débit de $\mu$. Ceci implique que la file d'attente au second commutateur est toujours vide. Par conséquent, celui-ci estime une bande passante, égale à sa capacité maximale qu'on note $C$, puisque à chaque réception de $AN_{rm}$ cellules RM, l'affectation suivante est exécutée : $C* = min(C*+aQ, C)$. Supposons maintenant que $C = d\mu$, et que $b, d$ sont très grands, et que s'il y a une connexion voulant traverser uniquement le deuxième commutateur, elle peut facilement avoir $ICR = \mu$ comme débit initial.

A l'établissement d'une connexion, avec les caractéristiques citées ci-dessus, utilisant uniquement le deuxième commutateur, le débit qui lui sera allouer sera de $0.75(C+\beta Q/h)$ pendant un RTT au minimum. On aura ainsi dans le prochain RTT un débit total ABR entrant au deuxième commutateur, égal à $\mu+0.75(C+\beta Q/h) = \mu+0.75(d\mu+\beta Q/h) > (1+0.75d)\mu$. La file d'attente va se construire à ce moment là à un débit d'au moins $(1+0.75d-b)\mu$. Ainsi, si la capacité maximale de la bande passante est beaucoup plus large que la bande passante disponible pour le service ABR, alors la file va se construire à un débit d'au moins $\mu$ ce qui peut engendrer une taille de la file d'attente très large, et par conséquent des pertes.

## C.2 Stabilité avec estimation de la bande passante

Dans ce qui suit, on étudiera la stabilité de l'algorithme en tenant compte du fait que la bande passante est estimée par ERAQLES, et en utilisant le même modèle décrit dans le chapitre 7.

ERAQLES réestime la bande passante chaque fois que $AN_{rm}$ cellules RM sont servies. Si le nombre de connections partageant le même lien est immense, on peut dire alors qu'ERAQLES estime, pratiquement, la bande passante à chaque passage d'une cellule RM d'une connexion donnée (si le nombre de connexions est $AN_{rm}$ par exemple). D'autre part, si la file d'attente ne se vide jamais, alors ERAQLES estime la bande passante à des intervalle réguliers de durée $\delta t = \frac{AN_{rm}^2}{\mu}$ secondes. Par conséquent, en supposant que la taille de la file ne varie pas énormément pendant l'intervalle d'estimation ($y(t)$ est proche de la moyenne pendant cet intervalle), on a

$$C^*(t) = C^*(t-\delta t) + a(Q-y(t)) \Longrightarrow \dot{C}^*(t) \triangleq \frac{dc^*(t)}{dt} = \frac{a(Q-y(t))\mu}{AN_{rm}^2} \tag{C.1}$$

D'autre part on a

$$x_i(t+\beta_i) = \left((1-s_0)\frac{1}{N}+s_0\right)(C^*(t)+a(Q-y(t))) \tag{C.2}$$

Puisque la file ne se vide jamais, on a

$$y(t) = \left(y(t-\delta t) + \left(\sum_{i=1}^{N} x_i(t-\gamma_i)-\mu\right)\delta t\right)^+ \Longrightarrow \dot{y}\ (t) = \sum_{i=1}^{N} \dot{x}_i\ (t-\gamma_i)$$

En substituant l'équation (C.2), on peut déduire

$$\ddot{y}\ (t) = \sum_{i=1}^{N}\left(\left((1-s_0)\frac{1}{n}+s_0\right)\left(\dot{C}^*\ (t-\tau_i) - a\ \dot{y}\ (t-\tau_i)\right)\right) \tag{C.3}$$

En substituant, également, l'équation (C.1), et puisque les sources sont équivalentes ($\tau_i = \tau\ \forall i$), on déduit

$$\ddot{y}\ (t) + (1+(N-1)s_0)a\left(\frac{\mu}{AN_{rm}^2}(y(t-\tau)-Q)+\dot{y}\ (t-\tau)\right) = 0 \tag{C.4}$$

En effectuant le changement de variable suivant : $H(t) \triangleq y(t) - Q$, on obtient, finalement

$$\ddot{H}\ (t) + (1+(N-1)s_0)a\left(\dot{H}\ (t-\tau)+\frac{\mu}{AN_{rm}^2}H(t-\tau)\right) = 0 \tag{C.5}$$

L'équation caractéristique de cette équation différentielle est la suivante :

$$\lambda^2 + (1+(N-1)s_0)a\left(\lambda e^{-\tau\lambda}+\frac{\mu}{AN_{rm}^2}e^{-\tau\lambda}\right) = 0 \tag{C.6}$$

**Théorème C.2.1** *Soit l'équation différentielle suivante*

$$\frac{d^2x(t)}{dt^2} + \alpha\frac{d^2x(t-\tau)}{dt^2} + a\frac{dx(t)}{dt} + b\frac{dx(t-\tau)}{dt} + cx(t) + dx(t-\tau) = 0 \tag{C.7}$$

*dont l'équation caractéristique*

$$\lambda^2 + \alpha\lambda^2 e^{-\tau\lambda} + a\lambda + b\lambda e^{-\tau\lambda} + c + de^{-\tau\lambda} = 0 \tag{C.8}$$

*soient $\omega_\pm$, $0 \le \theta_1 \le 2\pi$ et $\tau_{0,1}$ tels que*

$$\begin{aligned}
{\omega_\pm}^2 &= \frac{1}{2}\left(1-\alpha^2\right)^{-1}\Big\{\left(b^2+2c-a^2-2d\alpha\right) \\
&\quad \pm\left[\left(b^2+2c-a^2-2d\alpha\right)^2 - 4\left(1-\alpha^2\right)\left(c^2-d^2\right)\right]^{1/2}\Big\}; \\
\cos\theta_1 &= -\frac{ab{\omega_+}^2+(c-{\omega_+}^2)(d-\alpha{\omega_+}^2)}{b^2{\omega_+}^2+(d-\alpha{\omega_+}^2)^2}, \\
\sin\theta_1 &= \frac{(d-\alpha{\omega_+}^2)a\omega_+ - b\omega_+(c-{\omega_+}^2)}{b^2{\omega_+}^2+(d-\alpha{\omega_+}^2)^2}, \\
\tau_{0,1} &= \frac{\theta_1}{\omega_+}.
\end{aligned} \tag{C.9}$$

*Supposons que $|\alpha| < 1$, $c+d \neq 0$, et $a^2+b^2+(d-\alpha c)^2 \neq 0$. Le nombre des solutions imaginaires différentes avec une partie imaginaire positive (négative), peut être zéro, un ou deux seulement.*

1. *S'il n'y a pas de telles solutions, alors la stabilité de la solution zéro ne change pas pour tout* $\tau \geq 0$.
2. *S'il y a une seule solution imaginaire avec la partie imaginaire positive, alors une solution zéro instable ne deviendra jamais stable pour tout* $\tau \geq 0$. *Si par contre, la solution zéro est stable pour* $\tau = 0$ *alors elle est uniformément asymptotiquement stable pour* $\tau < \tau_{0,1}$ *et devient instable pour* $\tau \geq \tau_{0,1}$ *(équation (C.9)).*
3. *S'il y a deux solutions imaginaires avec une partie imaginaire positive,* $i\omega_+$ *et* $i\omega_-$*, tels que* $\omega_+ > \omega_- > 0$*, alors la stabilité de la solution zéro peut changer un nombre fini de fois, au plus quand* $\tau$ *est augmenté, et devient instable.*

La preuve du théorème et l'énoncé général peut être trouvé dans [71].

Revenant à présent à la stabilité de la solution-zéro $H(t) = 0$ (solution triviale) de notre équation différentielle (équation (C.5)). D'après [71] (pp. 74), l'équation caractéristique (C.6) n'a qu'une seule solution $i\omega$ à partie imaginaire positive ($\omega_+ > 0$), puisque $(1 + (N - 1)s_0)a\frac{\mu}{AN_{rm}^2} > 0$. Du moment que notre équation différentielle satisfait toutes les conditions du Théorème précédent, alors par application directe de celui-ci, on déduit que :

1. Si pour $\tau = 0$ la solution zéro de l'équation (C.5) n'est pas stable, alors elle demeurera instable pour tout $\tau \geq 0$.
2. Si par contre elle est stable alors, on déterminera le domaine de stabilité

Le problème se ramène ainsi à l'étude de l'équation différentielle suivante :

$$\ddot{H}\ (t) + (1 + (N-1)s_0)a\left(\dot{H}\ (t) + \frac{\mu}{AN_{rm}^2}H(t)\right) = 0 \tag{C.10}$$

Dont l'équation caractéristique est

$$\lambda^2 + (1 + (N-1)s_0)a\lambda + \frac{(1 + (N-1)s_0)a\mu}{AN_{rm}^2} = 0 \tag{C.11}$$

Deux cas sont à prendre en compte :

– $(1 + (N-1)s_0)^2a^2 - 4\frac{(1 + (N-1)s_0)a\mu}{AN_{rm}^2} < 0 \Longrightarrow 2T > \frac{AN_{rm}^2}{(1 + (N-1)s_0)h\mu}$,

et l'équation (C.11) admet deux solutions complexes $\lambda_\pm = k_1 \pm ik_2$ avec $k_1 < 0$. Par conséquent, toutes les solutions de (C.10) sont des combinaisons linéaires des fonctions $H_\pm(t) = Ke^{\lambda_\pm t}$. Il est facile de vérifier que $\lim_{t\to\infty} H_\pm(t) = 0$, ce qui nous permet de déduire que la solution zéro $H(t) = 0$ est stable pour $\tau = 0$.

– $T \leq \frac{AN_{rm}^2}{(1 + (N-1)s_0)h\mu}$,

et l'équation (C.11) admet alors, deux solutions réelles négatives $\lambda_1$ et $\lambda_2$. Par conséquent toutes les solutions de (C.10) sont des combinaisons linéaires des fonctions $H_1(t) = e^{\lambda_1 t}$ et $H_2(t) = e^{\lambda_2 t}$, avec $\lim_{t\to\infty} H_1(t) = lim_{t\to\infty}H_2(t) = 0$, ainsi la solution zéro $(H(t) = 0)$ est stable pour $\tau = 0$.

Puisque la solution zéro est stable dans tous les cas pour $\tau = 0$, alors elle est stable pour tout $\tau \leq \tau_0$ et instable au delà de cette valeur, avec $\tau_0 = \frac{\theta}{\omega}$, où

$$\begin{aligned}
\omega^2 &= \frac{((1+(N-1)s_0)\frac{h}{T})^2}{2}\left(1+\sqrt{1+4\frac{((1+(N-1)s_0)\frac{\mu h}{AN_{rm}^2 T})^2}{((1+(N-1)s_0)\frac{h}{T})^4}}\right); \\
\cos\theta &= \frac{((1+(N-1)s_0)\frac{\mu h}{AN_{rm}^2 T})\omega^2}{((1+(N-1)s_0)\frac{h}{T})^2\omega^2 + ((1+(N-1)s_0)\frac{\mu h}{AN_{rm}^2 T})^2}; \\
\sin\theta &= \frac{((1+(N-1)s_0)\frac{h}{T})\omega^3}{((1+(N-1)s_0)\frac{h}{T})^2\omega^2 + ((1+(N-1)s_0)\frac{\mu h}{AN_{rm}^2 T})^2}. \qquad \text{(C.12)}
\end{aligned}$$

Pour déterminer $\theta$, on a

$$\theta = \arctan \frac{(1+(N-1)s_0)\frac{h}{T}}{(1+(N-1)s_0)\frac{\mu h}{AN_{rm}^2 T}}\omega = \arctan \frac{AN_{rm}^2}{\mu}\omega$$

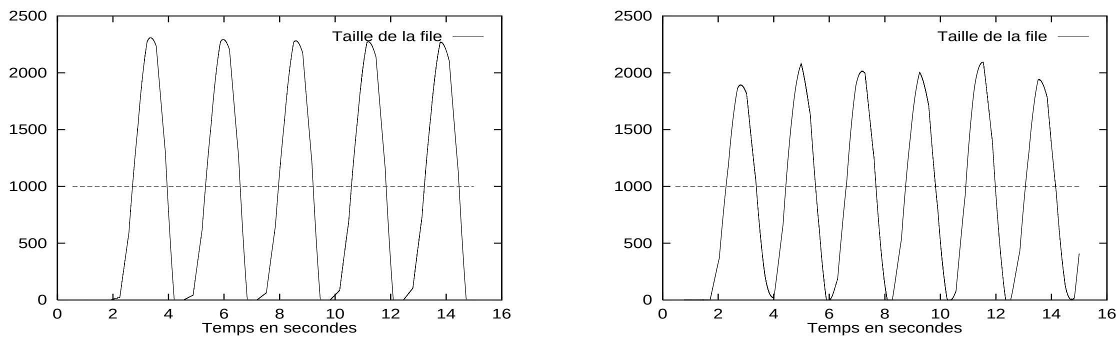
D'autre part, on a $\arctan(x) \leq x \; \forall x$. Par conséquent,

$$\theta \leq \frac{AN_{rm}^2}{\mu}\omega \Longrightarrow \tau_0 < \frac{AN_{rm}^2}{\mu}$$

Ce qui nous permet enfin d'écrire la proposition suivante :

**Proposition C.2.1** *Si* $2T \geq \frac{AN_{rm}^2}{\mu}$, *alors* $\forall\, h, Q, N > 1$, *ERAQLES, avec estimation de la bande passante n'est pas stable*[1].

### C.2.1 Simulations




(a) $\tau = 0.327$ (b) $\tau = 0.270$

FIG. C.1: Taille de la file d'attente sous ERAQLES, pour $\tau = 0.327$ *sec* et $\tau = 0.270$ *sec*

Considérons un exemple où deux sources ($N = 2$) partagent le goulot d'étranglement, avec $h = h_{opt} = 0.1839$, $a = a_{opt} = \frac{h}{T}$, $\mu = 50000$ *cellule/sec*, $C^*(0) = 30000$ *cellules/sec* (la

[1] On rejoint ici l'analyse du chapitre 7 (section 7.3.1), encore une fois vérifiée pour des bandes passantes très larges

bande passante estimée à l'instant initial), $s_0 = 1/n = 1/2$, $Q = 1000$ et enfin $AN_{rm} = 512$. D'après la proposition précédente, ERAQLES n'est pas stable pour $T = \frac{AN_{rm}^2}{2\mu} = (32 * 512)/100000 = 0.16384$ secondes. Considérons que le délai d'aller retour entre le commutateur et les sources est de 0.32768 secondes.

La Figure C.1 illustre la taille de la file d'attente dans le temps. Cet exemple (C.1 (a)) montre bien que la valeur de $\tau_0$ donnée dans la proposition précédente est très optimiste, dans la mesure où l'algorithme n'est pas stable même pour des valeurs inférieures à celle-là (C.1 (b)). Rappelons que le fait qu'ERAQLES estime la bande passante en utilisant une moyenne de la taille de la file d'attente, prise sur l'intervalle d'estimation, au lieu de la taille au passage de la $AN_{rm}$ème cellule RM, ne change rien à la stabilité de l'algorithme ; on obtient le même comportement de la taille de la file d'attente. La Figure C.1 (b) quant à elle illustre un autre exemple, où le délai d'aller retour considéré est de $0.270 < 0.327$ pour confirmer l'optimisme de la proposition précédente.

L'analyse donnée ici concerne bien évidemment le cas où toutes les sources sont équivalentes. On peut s'attendre à ce que les performances de l'algorithme se dégradent plus, dans un milieu hétérogène.

# Bibliographie


[1] The ATM forum Technical Committee, *Traffic Management Specification*, Version 4.0, Avril 1996.

[2] ITU-T Recommendation I.371, "Traffic Control and Congestion Control in B-ISDN", Août 1996.

[3] H. Afifi, D. Bonjour, O. Alloumi, "TCP over Native ATM", *Proceedings of the JENC7 Conference*, Budapest, Hungary. Mai 1996.

[4] J.S. Ahn, P. B. Danzig, Z. Liu, L. Yan, "Experience with TCP Vegas : Emulation and experiment", *In Proceedings of the SIGCOMM'95 Symposium*, Août 1995.

[5] O. Ait-Hellal, "Stability analysis of ABR congestion Control Algorithms", Manuscript.

[6] O. Ait-Hellal, E. Altman, "Problems in TCP Vegas and TCP Reno", *DNAC (De Nouvelles Architectures pour les Communications)*, UVSQ, Paris, 3-5 Décembre 1996.

[7] O. Ait-Hellal, E. Altman, "Analysis of TCP-Vegas and TCP-Reno", *IEEE International Conference on Communications (ICC'97)*, pp. 495-499. Montreal, 8-12 Juin 1997.

[8] O. Ait-Hellal, E. Altman, "Performance Evaluation of the Rate-Based Flow Control Mechanism for ABR Service : Generalization", *IEEE INFOCOM'99*, New-York. Février 1999.

[9] O. Ait-Hellal, E. Altman, T. Basar, "Rate based flow control with bandwidth information", *European Trans. on Telecom.*, pp. 55-66, 1997. Un e version courte a paru dans *Proceedings of the 35th IEEE Conference on Decision and Control*, pp. 2911-2916, Kobe, Japon. Décembre 1996.

[10] O. Ait-Hellal, E. Altman, T. Basar, "A robust identification algorithm for traffic models in telecommunications", soumis à la revue *Mathematical Models of Operations Research.*

[11] O. Ait-Hellal, E. Altman, D. Elwadghiri, M. Erramdani, "Performance Evaluation of the Rate-Based Flow Control Mechanism for ABR Service", *INRIA Research Report No. RR-3131. Second IFIP Workshop on Traffic Management and synthesis of ATM networks*, Montreal, 24-26 Septembre 1997. À paraître dans le journal de *Telecommunication Systems*, special issue.

[12] O. Ait-Hellal, E. Altman, A. Jean-Marie, "The Loss Process of Messages width Redundant Packets in an M/M/1/K", *Thirty-third annual Allerton Conference on Communication, Control, and Computing*, Allerton, 4-6 Octobre 1995.

[13] O. Ait-Hellal, E. Altman, A. Jean-Marie, I. Kurkova, "On Loss Probabilities in Presence of Redundant Packets and Several Traffic Sources", soumis pour présentation à *INFOCOMM'99.*

[14] E. Altman, D. Artiges, K. Traore, "On the Integration of Best-Effort and Guaranteed Performance Services", *INRIA Research report No. RR-3222*, 1997.

[15] E. Altman, F. Baccelli, J. C. Bolot, "Discrete-time analysis of adaptive rate control mechanisms", *High Speed Networks and their performance*, H. G. Perros and Y. Viniotis Eds., North Holland, pp. 121-140, 1994.

[16] E. Altman, F. Boccara, J. Bolot, P. Nain, P. Brown, D. Collange, C. Fenzy "Analysis of TCP/IP Flow Control Mechanism in Hight- Speed Wide-Area Networks", *IEEE Conference on Decision and Control*, New Orleans, Décembre 95.

[17] E. Altman, H. J. Kushner, "Admission Control for Combined Guaranteed Performance and Best Effort Communications Systems Under Heavy Traffic", *IEEE Conference on Decision and Control*, Tampa, Florida, USA, Décembre 1998.

[18] A. Arulambalam, X. Chen, N. Ansari, "Allocating Fair Rates For Available Bit Rate Service in ATM Networks", *IEEE Communication magazine*, Vol. 34, No 11, pp. 92-100, Novembre 1996.

[19] S. Basu, A. Mukherjee, S. Klivansky, "Time series models for internet traffic", in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Fransisco, California), pp. 611-619, Mars 1996.

[20] R. Bellman, K. L. Cooke, "Differential-Difference Equations", *Academic press*, 1963.

[21] L. Benmohamed, D. Su, "Analysis of the Rate-Based Traffic Management Proposal for ATM Networks", *Proceedings of the Third International Conference on Telecommunications Systems Modelling and Analysis*, 16-19 mars, 1995.

[22] J-C. Bolot, "End-to-end packet delay and loss behavior in the Internet", *Proc. ACM Sigcomm '93*, pp. 289-298, San Fransisco, CA, Septembre 1993. (A longer version appears in Journal of High Speed Networks).

[23] T. Bonald, "Comparison of TCP Reno and TCP Vegas via fluid approximations", manuscript, *INRIA-Sophia.*

[24] F. Bonomi, K. W. Fendick, "The Rate-Based Flow Control Framework for the Available Bit Rate", *IEEE Network*, Vol. 9, pp. 25-39, mars 1995.

[25] O. Bonaventure, "A simulation study of TCP with the proposed GFR service category", Presented at DAGSTUHL seminar 9725, *High-Performance Networks for Multimedia Applications*, Juin 1997, Schloβ Dagstuhl, Allemagne.

[26] L.S. Brakmo, L.L. Peterson, "TCP Vegas : End to End Congestion Avoidance on a Global Internet", *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465-1480, Octobre 1995.

[27] L.S. Brakmo, "TCP Vegas Release 0.8", Nov 94. Disponible à ftp ://ftp.cs.arizona.edu /xkernel/new-protocols/Vegas.Tar.Z.

[28] M. O. Brouillet, U. Madhow, "Rate control for adaptive bit rate sources on ATM networks using one bit congestion notification", *Research report UILU-ENG-96-2214, University of Illinois at Urbana-Champaign*, Mai 1996.

[29] D. Sisalem, H. Schulzrinne, "Congestion Control in TCP : Performance of Binary Congestion Notification Enhanced TCP Compared to Reno and Tahoe TCP", *International Conference on Network Protocols (ICNP)*, (Columbus, Ohio), pp. 268-275, Octobre 1996.

[30] Y. Chang, N. Golmie, D. Siu, "A Rate-Based Flow Control Switch design for ABR Service in an ATM Network", *Twelfth International Conference on Computer Communication ICCC'95*, Août 1995.

[31] Y. Chang, N. Golmie, D. Siu, "Study of Interoperability between EFCI and ER Switch Mechanisms for ABR Traffic in an ATM Network", *Proc. of Gigabit Networking Workshop*, Avril 1995.

[32] A. Charny, D.D. Clark, R.Jain, "Congestion Control With Explicit Rate Indication",*Proc. ICC'95*, Juin 1995.

[33] D. D. Clark, "window and acknowledgement strategy in TCP", *RFC-813*, Juillet, 1982.

[34] R. Cole, D. Shur, "IP over ATM : A Framework Document", *Request For Comments 1932*, Avril 1996.

[35] D. E. Comer, J. C. Lin, "TCP Buffering And Performance Over An ATM Network", *Perdue Technical Report CSD-TR 94-026*, mars 1994.

[36] D. E. Comer, D. L. Stevens, "Internetworking with TCP/IP", *Volume 1, 2, 3. Addison Wesley*, 1994.

[37] C. Courcoubetis, V.A. Siris, G.D. Stamoulis, "Integration of Pricing and Flow Control for Available Bit Rate Services in ATM Networks", *Proc IEEE Globecom' 96*, London, 1996.

[38] J. Crowcroft, Ph. Oechslin, "Differentiated End to End Internet Services using a Weighted Proportional Fair Sharing TCP", draft submitted for publication.

[39] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *Proceedings of ACM SIGCOMM'95*, Mai 1996. disponible sur le site ftp à ftp.ee.lbl.gov/papers/.

[40] C. Fang, H. Chen, J. Hutchins, "A Simulation Study of TCP Performance in ATM Networks", *GLOBCOMM'94*.

[41] S. Floyd, "TCP and successive fast retransmits". *Lawrence Berkeley Laboratory, Technical Report*, 24 Février 1995. Disponible sur le site ftp à ftp.ee.lbl.gov/papers/fastrans.ps.

[42] S. Floyd, "TCP and Explicit Congestion Notification", *ACM Computer Communication Review*, vol. 24, n. 5, pp. 10-23, Octobre 1994.

[43] S. Floyd, V. Jacobson, "Random Early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking*, Août 1993.

[44] C. Fulton, S. Q. Li, C.S. Lim, "UT : ABR Feedback Control with Tracking", *ATM-Forum contribution*, atm97-0239.

[45] M. Grossglauser, S. Keshav, D. Tse, "RCBR : A Simple and Efficient Service for Multiple Time-Scale Traffic", to appear in *IEEE/ACM Trans. on Networking*, 1998.

[46] R. Guerin, J. Heinanen, "UBR+ Service Category Definition", *ATM Forum contribution 96-1598*, Décembre 1996.

[47] R. Guerin, J. Heinanen, "UBR+ Enhancements", *ATM Forum contribution 97-0015*, Février 1996.

[48] J. C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *In proceedings of the ACM SIGCOMM'96*, pages 270-280, Septembre 1996.

[49] H. Hsiaw, L. Roberts, A. Jain, B. Makrucki, T. Tofigh, A. W. Barnhart, B. Holden, G. Fedorkow, J. N. Daigle, D. Kline, H. Suzuki, G. Ramamurthy, N. Giroux, R. Kositpaiboon, D. Hughes, M. G. Hluchyj, G. Garg, K. Y. Siu, H. Y Tzeng, N. Yin, "Closed-Loop Rate-Based Traffic Management", *ATM Forum Contribution 94-0438R2*, Septembre 1994.

[50] C. Ikeda, H. Suzuki, M. Murata, "Transient State Analysis and Maximum Buffer Requirements for Enhanced PRCA", *ATM Forum Contribution 94-0910*, Septembre 1994.

[51] R. Izmailov, "Analysis and Optimization of Feedback Control Algorithms for Data Transfers in High-Speed Networks", *SIAM Journal on Control and Optimization*, Volume 34, Number 5 pp. 1767-1780, 1996.

[52] V. Jacobson, "Congestion avoidance and control". *ACM SIGCOMM'88*, pp. 273-288, 1988.

[53] V. Jacobson, "Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno", *Proceedings of the Eighteenth Internet Engineering Task Force*, pp. 365, University of British Columbia, Vancouver, B.C, Septembre 1990

[54] V. Jacobson, "Modified TCP congestion avoidance algorithm", *mailing list, end2end-interest*, reçu le 30 Avril 1990.

[55] V. Jacobson, R Braden, "TCP extensions for Long-Delay Paths", *RFC 1072*, Octobre 1988.

[56] V. Jacobson, R Braden, D. Borman, "TCP extensions for High Performance", *RFC 1323*, Mai 1992.

[57] V. Jacobson, L. Peterson, L. Brakmo, S. Floyd, "Problems with Arizona's Vegas" *mailing list, end2end-tf* (discussion sur TCP Vegas), 1994.

[58] R. Jain, "Congestion Control and Traffic Management in ATM Networks : Recent Advances and A Survey", *invited submission to Computer Networks and ISDN Systems.*

[59] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, R. Viswanathan, "ERICA switch algorithm : A Complete Description", *ATM Forum/96-1172. http ://www.cis.ohio-state.edu/ jain/atmf/a96-1172.htm.*

[60] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, "Source Behavior for ATM ABR Traffic Management : An Explanation", *IEEE Communications Magazine*, Novembre 1996.

[61] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, F. Lu, S. Srinidhi, "TBE and TCP/IP traffic", *ATM Forum contribution 96-0177*, Los Angeles, Février 1996.

[62] R. Jain, S. Kalyanaraman, R. Viswanathan, "The OSU Scheme for Congestion Avoidance in ATM Networks using Explicit Rate Indication", *Proceedings WATM'95 First Workshop on ATM Traffic Management*, Paris, Décember 1995.

[63] Steven M. Kay, "Modern Spectral Estimation : Theory and Application", *Prentice Hall,* 1988.

[64] F.P. Kelly, "Effective Bandwidths at Multi-Class Queues", *Queueing Systems*, 9, pp. 5-16, 1991.

[65] F. P. Kelly, "Notes on effective bandwidths", *Stichastic networks : theory and applications*, (Ed. S. Zachary, I.B. Ziedins & F. P. Kelly), Oxford University Press, 1996.

[66] F. P. Kelly, "Charging and rate control for elastic traffic", *European Transactions on Telecommunications*, volume 8, pp. 33-37 1997.

[67] F. P. Kelly, A.K. Maulloo, D.K.H. Tan, "Rate control in communication networks : shadow prices, proportional fairness and stability",*Journal of the Operational Research Society*, 49, pp. 237-252, 1998.

[68] S. Keshav, "REAL : A network simulator", *Department of Computer Science, UC Berkeley, Technical Report* 88/472, 1988.

[69] B.K. Kim, B.G. Kim, I. Chong, "Fixed Point Algorithm for ABR Congestion Control", *AF-TM 96-0061*, Avril 1996.

[70] D. Kofman, M. Gagnaire, “Réseaux Haut Débit : Réseaux ATM, réseaux locaux et réseaux tout-optiques”, *IterEditions/Masson*, Paris, 1996.

[71] Y. Kuang, “Delay differential equations, with applications in population dynamics”, *Academic press*, 1993.

[72] T. V. Lakshman, U. Madhow, “Performance analysis of window-based flow control using TCP/IP : the effect of high bandwidth-delay products and random loss”, *IFIP Transactions C-26, High Performance Networking*, pp. 135-150, North-Holland, 1994.

[73] T. V. Lakshman, U. Madhow, B. Suter, “Window-based error recovery and flow control with a slow acknowledgment channel : a study of TCP/IP performance”, draft soumis pour publication.

[74] M. Laubach, “Classical IP and ARP over ATM”, *Request For Comments (RFC) 1577.*

[75] M. Lin, J. Hsieh, J. P. Thomas, J. A. MacDonald, “Distributed Network Computing Over Local Networks”, To appear in *IEEE Journal on Selected Areas in Communications, Special issue of ATM LANs : Implementations and Experiences with an Emerging Technology.*

[76] Z. Liu, P. Nain, D. Towsley, “ Exponential bounds with application to call admission”, *Journal of the ACM*, Vol. 44, No. 3, pp. 366-394, Mai 1997.

[77] S. L. Marple Jr, “Digital Spectral Analysis with Applications”, *Prentice Hall,* 1987.

[78] L. Massoulié, J. Roberts, “Fairness and Quality of Service for Elastic Traffic”, Submitted to *SIGCOMM'98.*

[79] M. Mathis, J. Mahdavi, “Forward Acknowledgment : Refining TCP Congestion Control”, *Proceedings of ACM SIGCOMM'96*, Août 1996.

[80] M. Mathis, J. Mahdavi, S. Floyd, A. Romanov, “TCP Selective Acknowledgment Options”, *RFC 2018*, Octobre 1996.

[81] G. Minden, V. Frost, J. Evans, B. Ewy, “TCP/ATM Experiences in the MAGIC Testbed”, disponible sur ftp ://ftp.tisl.ukans.edu/pub/papers/TCP-Perform.ps

[82] Y. Moret, “Conception et évaluation d'un algorithme ABR dans les réseaux ATM : ERAQLES”, *Thèse de Doctorat de l'Université Paris VI*, Décembre 1997.

[83] Y. Moret, S. Fdida, “ERAQLES an Efficient Explicit Rate Algorithm for ABR”, *GLOBECOM'97*, Phoenix, USA, Novembre 1997.

[84] P. Newman, G. Minshall, T. Lyon, “IP Switching : ATM Under IP”, *Soumis pour publication dans IEEE/ACM Transaction on Networking.*

[85] H. Ohsaki, G. Hasegawa, M. Murata, A.Miyahara, “Parameter Tuning of Rate-Based Congestion Control Algorithms and its Application to TCP over ABR”, *Proceedings WATM'95 First Workshop on ATM Traffic Management*, Paris, Décember 1995.

[86] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, H. Miyahara, “Rate-based Congestion control for ATM Networks”, *Computer Communication Review, acm-sigcomm*, special issue in ATM, ed. R. Jain and K.Y. Siu, Vol 25, No. 2, pp. 60-71, 1995.

[87] J. Padbye, V. Firoiu, D. Towsley, J. Kurose, “Modeling TCP Throughtput : A Simple Model ans its Empirical Validation”, *in ACM SIGCOMM'98*, Vancouver, British Columbia, Septembre 1998.

[88] V. Paxon, “Measurements and Analysis of End-to-End Internet Dynamics”, *Ph.D Thesis*, Computer Science Division, University of California, Berkley. Avril, 1997.

[89] V. Paxson, S. Floyd, "Wide-Area Traffic : The Failure of Poisson Modeling", *IEEE/ACM Transactions on Networking*, Vol. 3, No. 3, pp. 226-244, Juin 1995.

[90] K. Ramakrishnan, R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with Connectionless Network Layer", *Proc. SIGCOMM'88*, Août 1988.

[91] G. Ramamurthy, B. Sengupta, "A predictive hop-by-hop congestion control policy for high-speed networks" *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, vol. 3, (San Francisco), pp. 1033–1041, 1993.

[92] M. Ritter, "Network Buffer Requirements of the Rate-Based Control Mechanism for ABR services", *IEEE INFOCOM'96*, san francisco, USA, mars 1996.

[93] L. Roberts, "Enhanced PRCA (Proportional Control Algorithm)", *ATM Forum contribution 94-0735R1*, Août 1994.

[94] A. Romanow, "TCP over ATM : Some Performance Results", *ATM Forum/93-784*. ftp ://playground.sun.com/pub/tcp_atm/tcp_forum.7_93.ps.

[95] A. Romanow, S. Floyd, "The Dynamics of TCP Traffic over ATM Networks", *ACM SIGCOMM Computer Communications Review*, Octobre 1994. ftp ://ftp.ee.lbl.gov/papers/tcp_atm.ps.Z

[96] S. G. Sanjay, A. Kumar, "TCP Over End-to-End ABR : A Study of TCP Performance with End-to-End Rate Control and Stochastic Available Capacity", *IEEE Globecom'98*, Sydney, 1998.

[97] K.Y. Siu, H.Y. Tzeng, "Intelligent Congestion Control for ABR Service in ATM Networks", *Computer Communication Review, acm-sigcomm*, Vol 24, No. 5, pp. 81-106, 1995.

[98] G. Stépán, "Retarded Dynamical Systems : Stability and Characteristic Functions", Longman Scientific & Technical, 1989.

[99] W. R. Stevens, "TCP/IP illustrated", *volume 1. Addison Wesley*, 1994.

[100] A. Tanenbaum, "Réseaux", 3ème édition, *InterÉditions*, 1997.

[101] D. Tse, M. Grossglauser, "Measurement-Based Call Admission Control : Analysis and Simulation", *Proc. IEEE Infocom '97*, Kobe, Japan, Avril 1997.

[102] Z. Wang, J. Crowcroft, "A New Congestion Control Scheme : Slow Start and Search (Tri-S)", *ACM Computer Communication Review*, 21(1), pp.32-43, Janvier 1991.

[103] N. Yin, M. G. Hluchyj, "On closed-loop rate control for ATM cell relay networks", *Proc. IEEE Infocom '94*, pp. 99-108, Juin 1994.

[104] Z.L. Zhang, Z. Liu, J. Kurose, D. Towsley, "Call Admission Control Schemes under the Generalized Processor Sharing Scheduling Discipline", *INRIA* RR, No. 2711, 1995.

[105] L. Zhang, S. Shenker, D.D. Clark, "Observations on the dynamics of a Congestion Control Algorithm : The Effects of Two-Way Traffic", *In proceedings of the SIGCOMM'91 Symposium*, pp. 133-147, Septembre 1991.

# Index