Contribution au contrôle de congestion dans les protocoles de transport

Fabien Chatté [1]

HEUDIASYC UMR CNRS 6599
Université de Technologie de Compiègne,
Centre de Recherche de Royallieu, BP 20529,
60205, Compiègne, France.

Membres du Jury :

Pr. V. Vèque Univertsité Paris-sud XI (rapporteur)
Pr. G. Leduc Université de Liège (rapporteur)
Pr P. Rolin France Télécom
Pr. A. Bouabdallah Université de Technologie de Compiègne
Dr B. Ducourthial Université de Technologie de Compiègne (directeur de thèse)
Dr S. Niculescu Université de Technologie de Compiègne (directeur de thèse)

1. email: fabien.chatte@hds.utc.fr

À ma femme, avec une pensée particulière pour son soutien tout au long de mon doctorat et sa patience durant la rédaction de ce manuscrit

À ma famille

Remerciements

Je tiens à remercier :

Véronique Vèque d’avoir accepté d’être rapporteur et de faire partie de mon jury de thèse. Je tiens également à la remercier pour ses conseils qui m’ont permis d’améliorer certains points dans mon manuscrit.

Guy Leduc pour avoir accepté d’être rapporteur et de faire partie de mon jury de thèse. Je tiens également à le remercier pour les remarques pertinentes qu’il a pu me faire à travers son rapport.

Pierre Rolin pour avoir accepté d’être membre de mon jury de thèse.

Abdelmadjid Bouabdallha pour avoir accepté de faire partie de mon jury de thèse, ainsi que pour les conseils avisés qu’il a pu me donner durant mes études à l’UTC.

Bertrand Ducourthial et Silviu-Iulian Niculescu pour la qualité de leur encadrement, les nombreux conseils qu’ils m’ont donnés, ainsi que leur disponibilité.

Imed Romdhani pour la qualité de nos échanges et l’ambiance chaleureuse qu’il a apportée au sein de notre bureau.

Le personnel administratif et plus particulièrement Nathalie Hamel pour sa grande disponibilité.

Résumé

Dans ce manuscrit de thèse, nous commençons par présenter un panorama des différentes techniques de contrôle de congestion implémentées dans des protocoles de transport *unicast*. Ensuite, nous présentons une étude visant à définir les limites de validité de la modélisation continue des réseaux à commutation de paquets. Puis, nous décrivons et justifions la mise au point (dans le domaine du continu) d'un correcteur issu de techniques de contrôle d'automatique destiné à réguler le débit d'émission des sources d'un réseau, afin d'éviter la formation de congestions. Une fois le correcteur développé, il a été discrétisé de manière à l'implémenter dans un protocole de transport. Afin de comparer objectivement notre mécanisme de contrôle de congestion à ceux existants, nous définissons une méthodologie de comparaison. Cette méthodologie permet d'évaluer les performances des mécanismes de contrôle de congestion. Pour finir, grâce à la méthodologie développée, nous comparons les performances de notre protocole à celles de plusieurs protocoles de transport existants. Cette comparaison nous permet d'analyser, dans différentes situations, le fonctionnement des protocoles testés.

Abstract

In this PhD thesis manuscript, we begin by presenting a panorama of different congestion control techniques implemented in unicast transport protocols. Next, we present a study in which we try to define the validity limits of the fluid approximation of a packets switched network. After, we describe and justify the development (in the continuous time domain) of a controller, which is used to compensate the sending rate of network sources in order to avoid congestions. Once the controller developed, we discretize it in order to implement it in a transport protocol. To objectively compare our congestion control mechanism to the existing ones, we define a comparison methodology. This methodology allows to evaluate performance of congestion control mechanisms. At the end, we compare the performance of our protocol with those of several existing transport protocols. This comparison allows us to analyse in several cases, the behaviour of the tested protocols.

Sommaire

Chapitre 1

Introduction

1.1 Contexte de l'étude

1.1.1 Origines d'Internet

Dans les années soixante, alors que l'URSS et les États-Unis étaient en pleine guerre froide, l'armée américaine a demandé à un petit groupe de chercheurs de créer un réseau de communication capable de résister à une attaque nucléaire. Le concept d'un tel réseau reposait sur un système décentralisé, dans lequel la destruction d'une ou de plusieurs machines n'aurait pas bloqué les communications. C'est Paul Baran, qui en 1962, eut l'idée de créer un réseau ayant la forme d'une grande toile. Ce réseau avait une architecture en étoile et était maillé de telle sorte que les données pouvaient se déplacer en cherchant le chemin le plus court et attendre en cas d'embouteillage. Malheureusement, à l'issue de cette étude, le projet fut rejeté par le Pentagone.

Quelques années plus tard (en 1968), l'ARPA[1] du ministère de la Défense[2] des États-Unis a coordonné des travaux de recherche portant sur la définition d'un protocole de communication indépendant des vendeurs. En effet, à l'époque les protocoles de communication étaient « propriétaires », ce qui rendait impossible la communication entre deux machines provenant de constructeurs différents. Les travaux lancés par l'ARPA avaient pour but de relier ses centres de recherches pour partager leurs équipements informatiques. C'est ainsi qu'est né ARPAnet, un réseau suffisamment décentralisé pour qu'en cas d'attaque nucléaire il puisse continuer à fonctionner même si une partie du réseau était endommagée. En 1972, Ray Tomlinson mit au point un système de messagerie électronique, qui permettait l'échange d'informations au sein d'un réseau. Grâce au courrier électronique, il était désormais possible de contacter un grand nombre de personnes en envoyant un seul *mail*. À la même époque, des travaux ont permis la connexion d'autres réseaux au réseau ARPAnet : SATNET[3], PRNET[4], puis en 1973 le premier Ethernet développé par Xerox. C'est de cette interconnexion de plusieurs réseaux que vient le terme d'Internet. Ces avancées technologiques ont nécessité une gestion du réseau à un niveau plus élevé qu'il ne l'était dans ARPAnet (gestion au niveau matériel). Les protocoles de

1. *Advanced Research Projects Agency*.
2. *Department of Defense*.
3. *SATellite atlantic packet NET*.
4. *Packet Radio NETwork*.

communication devinrent alors un empilement de couches logicielles, s'ajoutant à la couche matérielle. Les paquets Internet seront dorénavant encapsulés dans d'autres paquets pour être véhiculés sur les divers réseaux connectés entre eux. Les réseaux de l'époque étaient basés sur la commutation de circuits. L'utilisation de ce mode impose une connexion physique de bout en bout pour pouvoir établir une communication. Afin de rendre les communications tolérantes aux pannes et aux défaillances du réseau physique (*cf.* attaque nucléaire), ARPAnet fut basé sur la commutation de paquets. Ce mode de communication permet aux données transmises d'être découpées en paquets et ainsi d'atteindre leur destination sans forcément toutes suivre la même route. Lorsqu'en 1975 le réseau ARPAnet fut quasiment au point, le gouvernement américain décida d'en confier la gestion à la DISA[5], une agence chargée des systèmes d'information liés à la Défense.

Dans le cadre du projet ARPAnet, les recherches sur TCP/IP[6] débutèrent en 1973. La version 4 d'IP (*cf.* annexe A.1) fut disponible en 1978, et ARPAnet migra vers IP en 1982. Au début des années quatre-vingt, le caractère militaire d'ARPAnet commença à s'estomper et, en 1982, la NSF[7] proposa de relier entre eux les différents réseaux existants. L'université de Berkeley en profita pour développer sa pile logicielle TCP/IP afin de l'intégrer dans BSD[8]. En 1984, seulement 1000 machines sont connectées à ARPAnet et en 1988 la France s'y connecte *via* l'INRIA[9]. En 1990, le réseau ARPAnet cesse d'exister et Internet devient le réseau mondial dédié à la recherche civile. En 1991, les travaux menés par Tim Berners-Lee au CERN[10] donneront naissance au *World Wide Web*, qui entraînera l'explosion d'Internet.

De nos jours, Internet désigne l'interconnexion de réseaux provenant des cinq continents, ce qui représente plusieurs dizaines de millions d'usagers. La pile protocolaire TCP/IP s'est imposée et représente la quasi totalité des communications sur Internet.

1.1.2 Le contrôle de congestion dans la pile protocolaire Internet

OSI. Le modèle de référence OSI[11] [52] a été mis en place par l'ISO[12] afin de standardiser les règles de communication entre les ordinateurs d'un réseau. À l'origine des réseaux, chaque constructeur avait ses propres règles de communication, ce qui avait pour conséquence de rendre incompatibles les réseaux entre eux. En cherchant à interconnecter les réseaux, il est devenu indispensable de normaliser les protocoles de communication. C'est de là qu'est né le modèle OSI.

Le rôle du modèle OSI est de standardiser la communication entre les machines afin que différents constructeurs puissent mettre au point des produits (logiciels ou matériels) compatibles. Ce modèle comporte sept couches distinctes (*cf.* figure 1.1). Chaque couche fournit des services à celle qui la succède. Les couches du modèle OSI [112, 105] sont les suivantes :

- couche physique (niveau 1) : assure la conversion des données en signaux électriques ;
- couche liaison de données (niveau 2) : assure l'interfaçage avec la carte réseau ;

5. *Defense Information Systems Agency*.
6. *Transmission Control Protocol / Internet Protocol*.
7. *National Science Foundation*.
8. Système UNIX gratuit développé par l'Université de Californie à Berkeley.
9. Institut National de la Recherche Informatique et Automatique.
10. Centre Européen pour la Recherche Nucléaire.
11. *Open Systems Interconnection.*
12. *International Standardisation Organisation*.

- couche réseaux (niveau 3) : assure la gestion des adresses, le routage des paquets et le contrôle de congestion ;
- couche transport (niveau 4) : assure le transport des segments et la gestion des erreurs ;
- couche session (niveau 5) : assure l'ouverture et la fermeture des sessions des ordinateurs du réseau ;
- couche présentation (niveau 6) : assure le formatage des données (compression, cryptage, *etc.*) ;
- couche application (niveau 7) : assure l'interfaçage entre l'application et le réseau.

Couches OSI		Modèle DoD	
N°	Nom	Nom	Exemples de protocoles
7	Application	Application	FTP, Telnet, SMTP, NVT, NFS, navigateurs web, *etc.*
6	Présentation		
5	Session		
4	Transport	hôte-à-hôte	TCP, UDP, *etc.*
3	Réseau	Internet	IP, ARP, RARP, ICMP *etc.*
2	Liaison	Accès réseau	Ethernet, *Token Ring*, FDDI, *etc.*
1	Physique		

FIG. 1.1 – *Pile protocolaire Internet.*

Il est intéressant de noter que les protocoles TCP et IP ne correspondent pas strictement à la classification du modèle OSI. En effet, de manière générale, TCP est classé comme étant un protocole de la couche 4 et IP comme étant un protocole de la couche 3. Or, si l'on se réfère au modèle OSI, c'est la couche 3, donc IP, qui devrait assurer le contrôle de congestion et non la couche 4 à laquelle TCP est associé.

TP-OSI. Les protocoles de la couche transport du modèle OSI gèrent le contrôle de bout-en-bout et les erreurs de transmission afin d'assurer le transfert de données. Ils effectuent le multiplexage et la séparation des connexions au niveau transport, et apportent des fonctionnalités telles que le séquencement, le contrôle de flot, la détection et le recouvrement d'erreurs.

Dans la pile OSI, il existe cinq protocoles pour la couche transport, classés du protocole de transport de classe 0 à celui de classe 4 [53] (TP[13]0, TP1, TP2, TP3 et TP4). Les protocoles de classes 0 à 3 fonctionnent uniquement sur des réseaux orientés connexion, alors que TP4 fonctionne quel que soit le type de réseau.

- **Protocole de transport de classe 0 (TP0).** Il effectue la fragmentation en segments et le ré-assemblage des données. Ce protocole segmente les données en fonction du plus petit PDU[14] rencontré dans les réseaux sous-jacents. Les segments sont ré-assemblés par le récepteur.
- **Protocole de transport de classe 1 (TP1).** Il reprend les fonctionnalités de TP0 et y ajoute le recouvrement d'erreurs. Ce protocole ré-émettera les segments non acquittés. Il ré-initialisera la connexion si un grand nombre de segments ne sont pas acquittés.

13. *Transport Protocols.*
14. *Protocol Data Unit.*

- **Protocole de transport de classe 2 (TP2).** Il reprend les fonctionnalités de TP0 et y ajoute des fonctions de multiplexage et démultiplexage de flux de données sur un unique circuit virtuel.
- **Protocole de transport de classe 3 (TP3).** Il reprend les fonctionnalités de TP2 et y ajoute le recouvrement d'erreurs effectué par TP1.
- **Protocole de transport de classe 4 (TP4).** Il reprend les fonctionnalités de TP3 et fournit un transport fiable sur des réseaux orientés ou non connexion (i.e., il permet d'´emettre des données même si aucune session n'est établie). Ce protocole est le plus étudié de tous les protocoles de transport OSI ; il est similaire au protocole TCP dans la pile TCP/IP.

Les protocoles TP4 et TCP sont tous deux conçus pour fournir un transport de bout en bout fiable, orienté connexion au dessus d'un réseau non fiable. Ces deux protocoles sont capables de gérer le ré-ordonnancement, la duplication et les pertes des paquets. Les différences les plus marquantes entre ces deux protocoles sont :

- lors d'une ouverture de connexion simultanée entre deux machines, TP4 ouvrira deux connexions bidirectionnelles alors que TCP n'en ouvrira qu'une ;
- les acquittements TCP peuvent contenir des données (*piggy-back*) alors que ceux de TP4 ne le peuvent pas ;
- la régulation du flot se fait avec une fenêtre glissante pour TCP, alors qu'elle se fait avec un sytème de credit pour TP4.

Au début des recherches sur le contrôle de congestion (fin des années 80 début 90), les études portaient autant sur TCP que sur TP-4 (avec par exemple le protocole DEC-bit). Puis, les recherches se sont focalisées sur TCP, qui est aujourd'hui sans conteste le protocole de transport sur lequel se basent la plupart des études.

Modèle DoD. La pile protocolaire OSI est fortement inspiré du modèle DoD[15] [20] qui n'utilise que quatre couches (*cf.* figure 1.1). Les couches du modèle DoD sont les suivantes :

- Couche accès réseau (Ethernet , *Token ring* , FDDI[16], *etc.*) : assure la définition de la forme sous laquelle des paquets doivent être acheminés en fonction du type de réseau. Cette couche regroupe les couches 1 et 2 du modèle OSI.
- Couche Internet (IP, ARP[17], RARP[18], ICMP[19], *etc.*) : assure le découpage des données en paquets, leur adressage ainsi que leur routage. Cette couche correspond à la couche 3 du modèle OSI.
- Couche hôte-à-hôte (TCP, UDP, *etc.*) : assure l'acheminement des données ainsi que le contrôle de l'état dans lequel elles sont reçues. Cette couche correspond à la couche 4 du modèle OSI.

15. *Department of Defense.*
16. *Fiber Distributed Data Interface.*
17. *Address Resolution Protocol.*
18. *Reverse Address Resolution Protocol.*
19. *Internet Control Messages Protocol.*

– Couche application (Telnet, FTP[20], navigateur web, NVT[21], SMTP[22], NFS[23], *etc.*) : comprend tout ce qui concerne les applications réseau. Elle regroupe les couches 5, 6, 7 du modèle OSI.

1.1.3 Réseaux à commutation de paquets

Il existe différents types de commutation : la commutation de circuits (celle utilisée dans le réseau RTC[24]), la commutation de trames (*frame relay*), la commutation de cellules (ATM[25]) et la commutation de paquets (IP) qui est à la base même du réseau Internet. Dans les réseaux à commutation de paquets implémentant un service datagramme, les données sont découpées par IP en paquets de petite taille (ou datagrammes). Ces paquets sont constitués d'une en-tête et des données à transmettre. L'en-tête contient notamment les adresses d'origine et de destination des paquets, ainsi que la manière de les ré-assembler (l'annexe A.1 décrit plus en détails le fonctionnement du protocole IP).

Le chemin suivi par les paquets pour aller de la source à la destination est généralement le plus court (en terme de nombre de routeurs traversés). À chaque intersection sur le réseau, un routeur se charge de déterminer en fonction de l'en-tête du paquet la route qu'il doit emprunter. Notons que deux paquets ayant la même source et la même destination peuvent suivre des routes différentes et donc arriver dans le désordre. Dans ce cas, c'est le récepteur qui aura la charge de les ré-ordonner.

L'aiguillage des paquets est effectué par les routeurs grâce à une table de routage qui met en correspondance l'adresse de destination du datagramme avec l'une des interfaces de sortie du routeur. Certains protocoles permettent aux routeurs de construire et de maintenir leur table de routage. Les protocoles RIP[26] [70] et OSPF[27] [77] sont les plus utilisés. Avant d'être aiguillés vers leur prochaine destination, les datagrammes sont temporairement stockés par les routeurs dans des files d'attente. Notons qu'il existe différentes politiques de gestion de file d'attente (*Drop Tail*, PBS, RIO, *etc.*). Les politiques *Drop Tail* et RED[28] [15] sont les plus fréquemment utilisées. Les différentes politiques de gestion de file d'attente ainsi que les protocoles de routage ne sont pas détaillés dans ce manuscrit dans lequel nous nous intéressons plus particulièrement au contrôle de congestion effectué par les protocoles de transport.

La différence d'architecture entre les réseaux à commutation de circuits et les réseaux à commutation de paquets est fondamentale. En effet, dans le cas d'un réseau à commutation de circuits, les données sont transmises au destinataire de manière canalisée (*i.e.*, elles empruntent toutes la même route) dans les circuits établis par le réseau.

Dans le cas d'un réseau à commutation de paquets, les paquets de différentes connexions partagent un même lien, et ceux d'une même connexion peuvent suivre des routes différentes, en fonction des routeurs traversés. Ce type de réseaux assure uniquement la transmission des

20. *File Tranfer Protocol*.
21. *Network Virtual Terminal*.
22. *Simple Mail Transfert Protocol*.
23. *Network File System*.
24. Réseau Téléphonique Commuté.
25. *Asynchronous Transfert Mode.*
26. *Routing Interface Protocol*.
27. *Open Shortest Path First*.
28. *Random Early Detection*.

données sans garantir qu'elles arriveront à destination. Dans un tel réseau, ce sont les équipements à chaque extrémité de la connexion qui assurent l'intégrité de la communication.

1.1.4 Congestions

Une congestion se produit lorsqu'une machine du réseau à commutation de paquets reçoit plus de paquets qu'elle ne peut temporairement en stocker, avant de les ré-émettre. En effet, Les paquets sont d'abord stockés dans des *buffers* avant retransmission en sortie, lorsqu'ils arrivent plus vite qu'ils ne sont retransmis, ils s'aglutinent dans les *buffers* provoquant une congestion de ces *buffers*. Lorsqu'une congestion survient, les paquets en excès sont supprimés lors de leur arrivée à la machine saturée. On peut alors distinguer deux cas :

- les applications fonctionnant en mode dégradé (*e.g.*, sur UDP [29] qui n'implémente aucun mécanisme fiabilisant les transmissions, *cf.* annexe A.2)
- les applications nécessitant la retransmission des paquets détruits (*e.g.*, sur TCP qui assure la fiabilité des transmissions). Dans ce cas, les applications devront supporter le retard engendré par la retransmission des données perdues. De plus, les retransmissions contribuent à la mauvaise utilisation des ressources du réseau (plusieurs émissions de la même donnée), déjà fortement sollicitées puisque des congestions y sont apparues.

Les problèmes liés aux congestions concernent donc tous les acteurs d'Internet : les utilisateurs (*i.e.*, les applications) et les opérateurs (*i.e.*, les équipements).

En 1988 [56], sous le poids du nombre croissant de connexions, et à cause de l'importance du trafic à écouler, Internet connut une première série de graves congestions. Après ces premiers signes d'écroulement d'Internet, les chercheurs ont commencé à travailler sur l'évitement des congestions. Certaines améliorations furent apportées à TCP ; elles assurèrent la survie et contribuèrent grandement au développement d'Internet. De nos jours, ce problème reste de première importance : le trafic double dans Internet tous les 100 jours (selon le constructeur CISCO). Le sur-dimensionnement des équipements n'étant pas une solution en soi, il est impératif d'optimiser l'utilisation des ressources d'Internet, ce qui se traduit par une meilleure gestion du trafic et donc par une amélioration des mécanismes de contrôle de congestion.

1.2 Problématique de l'évitement de congestion

1.2.1 Analyse

Comme nous venons de le voir, les congestions se forment lorsqu'un équipement du réseau est incapable d'absorber le flot de données entrant. Pour éviter ou contrôler les congestions, plusieurs mécanismes peuvent être mis en œuvre à différents niveaux :

- Au niveau de la source : le contrôle à la source interdirait que de nouvelles connexions s'établissent sur un chemin déjà saturé [104].
- Au niveau du réseau : l'équilibrage de la charge du réseau [78] permettrait aux routeurs de détourner certains paquets afin qu'ils évitent une portion du réseau saturée.
- Au niveau des routeurs : les routeurs pourront privilégier certains flots, et détruire préventivement des paquets provenant de flots ayant tendance à congestionner le réseau [15].

29. *User Datagram Protocol*.

L'ensemble de ces mécanismes vise à réduire la probabilité d'apparition des congestions ainsi que leur importance. Mais quelle que soit l'efficacité de tels mécanismes, une technique d'évitement des congestions au niveau des protocoles de transport restera nécessaire [36]. En effet, il est pertinent de maîtriser le débit d'émission des sources, afin de pouvoir le réguler en fonction de l'état de congestion du réseau. Ainsi, si toutes les sources d'un réseau sont équipées d'un tel mécanisme, elles réduiront leur débit d'émission lors de l'apparition d'une congestion, ce qui permettra de résorber la congestion.

Nos travaux ont porté sur les techniques d'évitement de congestions au niveau des protocoles de transport.

1.2.2 Différents travaux menés

Le protocole TCP implémente un mécanisme de contrôle de congestion qui force la source à réduire son débit lorsqu'une congestion est détectée. Ce mécanisme est apparu dans TCP en 1988 [56]. Grâce à la réception d'acquittement, TCP peut déterminer si les données qu'il émet sont reçues ou non par le destinataire. Lorsque TCP détecte que des données ont été perdues, il suppose que ces pertes sont dues à une congestion et réduit son débit d'émission. La régulation du débit d'émission de TCP se fait par l'intermédiaire d'une fenêtre d'émission qui représente la quantité de données que peut envoyer l'émetteur sans avoir reçu les acquittements correspondant.

Le fonctionnement de ce premier mécanisme de contrôle de congestion a, depuis, été amélioré de différentes manières.

- **Amélioration de la gestion des acquittements.** Une gestion plus fine des acquittements [35] ainsi que des informations qu'ils transportent permet une meilleure estimation de l'état de congestion du réseau. Cette estimation plus précise de l'état du réseau permet d'éviter les réductions de débit inutile.
- **Amélioration de la gestion de la fenêtre d'émission.** Une adaptation plus précise de la taille de la fenêtre d'émission [19, 42, 29] (*i.e.*, du débit) en fonction de l'état du réseau donne la possibilité de mieux exploiter les capacités du réseau.
- **Utilisation d'informations provenant du réseau.** Le mécanisme de contrôle de congestion de TCP peut également utiliser des informations explicites provenant des routeurs. L'utilisation de ces informations autorise une détection plus rapide et plus précise des congestions [98, 97].

Notons également que de nombreuses études ont été menées sur le contrôle de congestion dans les réseaux à commutation de cellules (ATM) [60, 51].

Avec la croissance des lignes à haut débit, les applications à flux de données temps réel telles que les vidéos ou les flux audio, se sont rapidement répandues sur Internet. Ces applications n'utilisent pas TCP car les fortes variations de débit et la fiabilité qu'il impose ne sont pas compatibles avec les flots temps réel. C'est la raison pour laquelle elles utilisent UDP qui, malheureusement, n'effectue aucun contrôle de congestion, et qui ne partage pas équitablement la bande passante disponible avec les autres flots.

L'équité entre flots est devenue un paramètre crucial, qui assure que tous les flots de données peuvent utiliser un même réseau sans subir de discrimination. Afin de respecter les critères d'équité et de transporter des flux multimédia, de nouveaux protocoles ont vu le jour (TFRC

[44], RAP [101], PASTRA [107], *etc.*). Ils proposent de cohabiter équitablement avec TCP tout en apportant un mode de transport adapté aux flux temps réel. Ces protocoles sont dits *TCP Friendly* car ils partagent équitablement la bande passante avec les flots TCP. Notons que la plupart d'entre eux n'utilisent pas une fenêtre pour réguler leur débit d'émission.

Le contrôle de congestion peut être fait de manière corrective ou préventive. Dans le cas d'un contrôle correctif, comme celui de TCP Reno, le protocole fait croître le débit d'émission de la source jusqu'à ce qu'il observe une ou plusieurs pertes (*i.e.*, jusqu'à saturer les files des routeurs). Dès que cela se produit, le débit est réduit. À l'inverse, le contrôle préventif, comme celui de TCP Vegas, a pour but de ne jamais saturer les files d'attente. Pour cela, lorsqu'une situation pouvant conduire à une congestion est détectée, le débit de la source est immédiatement (sans attendre de perdre des paquets) réduit. Il a été montré dans [2, 75] qu'en termes d'équité, de taux de bonnes transmissions, *etc.*, les protocoles préventifs ont de bien meilleurs résultats que les protocoles correctifs. Malheureusement, la quasi totalité du trafic Internet est gérée par diverses versions correctives de TCP (Tahoe, Reno, New Reno, Sack). Ces versions de TCP imposent aux nouveaux protocoles d'avoir un comportement correctif, sous peine de s'approprier toute la bande passante. Suite à l'étude bibliographique présentée au chapitre 2, et malgré les problèmes de déploiement que cela pose, nous avons décidé de nous intéresser aux protocoles préventifs.

1.2.3 Utilisation de l'automatique

En parallèle des travaux visant à améliorer le contrôle de congestion, d'autres études tentant de modéliser les réseaux à commutation de paquets ont été réalisées. La modélisation analytique (*i.e.*, dans le domaine du continu) des réseaux à commutation de paquets présente de nombreux avantages :

- **Simplification.** La modélisation continue simplifie l'étude des réseaux. En effet, un grand nombre de problèmes liés aux réseaux (comme le contrôle de congestion) peuvent être réduits à l'étude de systèmes dynamiques équivalents, largement connus et traités en automatique. Cette simplification des problèmes donne la possibilité de mieux comprendre le fonctionnement des mécanismes étudiés [84, 50, 75] dans un environnement totalement maîtrisé.
- **Solutions exactes.** La réduction des problèmes de réseau à l'étude de systèmes dynamiques, permet de trouver des solutions exactes dans le domaine du continu, alors qu'elles sont généralement empiriques dans le domaine du discret [22, 55].
- **Extension des possibilités de simulations.** Actuellement, les simulateurs à événements discrets ne permettent pas d'effectuer des simulations à large échelle (de l'ordre de plusieurs milliers de connexions) en un temps raisonnable. L'utilisation de modèles continus, qui optimisent les temps de calcul [67] des simulateurs, devrait rendre possible ce genre de simulations.

Sous certaines conditions (que nous étudions au chapitre 3), les études menées avec des modélisations continues du réseau peuvent s'appliquer à des réseaux à commutation de paquets (intrinsèquement discrets) et conduisent alors au développement de nouveaux protocoles. C'est dans cette problématique que s'inscrivent nos travaux.

1.3 Contributions

L'objectif principal de nos travaux est le développement d'un mécanisme de contrôle de congestion améliorant la qualité des transmissions multimédia. Nous avons également souhaité montrer la possibilité d'utiliser certains principes d'automatique pour y parvenir.

1.3.1 Déroulement de l'étude

Les travaux menés durant ces trois années de thèse peuvent être décomposés en cinq étapes. Les deux premières ont été menées en parallèle et les suivantes de manière séquentielle.

État de l'art. Comme pour toute étude, nous avons commencé par nous documenter sur l'existant. Les travaux ont donc débuté par une recherche bibliographique portant sur les mécanismes de contrôle de congestion dans les protocoles de transport *unicast*.

En parallèle de cette étude bibliographique, nous nous sommes intéressés aux travaux sur les réseaux menés dans le domaine du continu. Ces travaux nous ont poussés à nous demander quelle était la validité des résultats obtenus en continu une fois discrétisés.

Validité de l'approximation continue. L'évaluation de la validité de l'approximation continue passe par la comparaison de protocoles discrets et de leur modélisation continue. Deux possibilités se sont alors présentées :

- comparaisons analytiques des modèles discret et continu ;
- comparaison de simulations effectuées en discret et en continu.

La validité de l'approximation continue ayant jusqu'à présent été principalement étudiée à travers des comparaisons de simulations [13, 99], nous avons poursuivi l'étude dans cette voie. De plus, notons qu'il est très difficile d'obtenir des comparaisons analytiques.

Une fois le mode de comparaison déterminé, nous avons dû choisir le protocole dont les modèles continu et discret seraient comparés. Deux choix se sont de nouveau présentés :

- définir ou utiliser une modélisation continue d'un protocole de transport existant, comme TCP ;
- développer un mécanisme de contrôle de congestion simple, admettant une modélisation continue exacte.

Le but de cette étude n'étant pas d'évaluer la validité d'une modélisation de TCP, mais la cohérence des comportements observés en continu et en discret, nous avons préféré utiliser un mécanisme de contrôle de congestion simple. En effet, l'utilisation d'un tel mécanisme permet de ne pas entacher les résultats d'erreurs dues à une mauvaise modélisation. Le mécanisme utilisé dans cette étude est assez proche du modèle proposé par Izmailov dans [55] et de celui utilisé dans [13].

Développement d'un mécanisme de contrôle de congestion. Sous l'hypothèse que l'approximation continue d'un réseau à commutation de paquets est valide, nous avons voulu développer un mécanisme de contrôle de congestion employant des résultats issus du domaine du continu. Dans un premier temps, nous avons souhaité mettre en œuvre les résultats que nous avions obtenus dans [22]. Ces résultats étaient basés sur le modèle d'Izmailov [55]. Malheureusement, ce modèle suppose que l'on puisse connaître certains paramètres du réseau (tel que la taille des files d'attente), qu'il est impossible d'obtenir dans un environnement réel.

Nous avons donc décidé de développer un nouveau mécanisme de contrôle de congestion simple, basé sur des principes d'automatique largement connus et traités [26, 41]. La première étape de ce développement fut d'adapter au contrôle de congestion un correcteur classiquement utilisé en automatique. Cette adaptation a été réalisée dans un environnement continu où nous maîtrisons tous les paramètres. Une fois cette étape terminée, nous avons discrétisé et intégré à un protocole de transport le correcteur obtenu.

Développement d'une méthode d'évaluation des performances. Afin de connaître l'apport de notre protocole pour le contrôle de congestion, nous avons souhaité évaluer ses performances. Nous avons donc cherché une méthodologie de référence permettant une évaluation multi-critères et autorisant une comparaison avec d'autres protocoles. Nous n'avons pas trouvé de telle méthodologie ; c'est pourquoi nous avons décidé d'en créer une.

Évaluation des performances. Pour finir, nous avons comparé les performances de notre protocole avec celles de protocoles existants. Cette comparaison avait également pour but d'analyser le comportement des protocoles testés dans différentes situations.

1.3.2 Résultats

L'étude bibliographique que nous avons réalisée nous a permis de rédiger un état de l'art (*cf.* chapitre 2) dégageant les principales pistes de la recherche en contrôle de congestion.

Les travaux menés sur la validité de l'approximation continue (*cf.* chapitre 3) ont permis d'en montrer les limites. En effet, les résultats obtenus en continu et en discret sont cohérents dans un grand nombre de situations. Cependant, nous avons mis en évidence certains contextes d'exécution dans lesquels les comportements observés en continu et en discret divergent.

La méthode d'évaluation que nous avons développée (*cf.* chapitre 5) autorise une comparaison objective des performances dans des environnements variés. Cette méthode est basée sur des simulations. L'influence des différents paramètres du réseau y est évaluée selon 7 critères de performance du contrôle de congestion. La diversité des critères d'évaluation et des situations testées permet de ne pas favoriser un type de comportement et ainsi de juger les performances du contrôle de congestion de manière globale.

Le protocole de transport que nous avons développé implémente un mécanisme de contrôle de congestion basé sur un contrôleur de type « Proportionnel Intégral » modifié (*cf.* chapitre 4). La simplicité et l'efficacité de ce mécanisme rendent ce protocole de transport préventif très performant en terme de contrôle de congestion (*cf.* chapitre 6).

1.3.3 Publications

Les travaux présentés dans ce manuscrit de thèse ont donné lieu à des publications.

Chapitre de livre

Advances in communication control networks
chapitre : Delay effects on the asymptotic stability of various fluid models in high performances networks
S.-I. Niculecu, W. Michiels, D. Melchor-Aguillar, T. Luzyanina, F. Mazenc, K. Gu, etF.

Chatté
Springer-Verlag : Heidelberg collection : LNCIS 2004
À paraître ; accepté en novembre 2003.

Articles dans des revues

État de l'art des protocoles de transport.
F. Chatté et B. Ducourthial.
Revue TSI (Techniques et Sciences Informatiques), Hermès sciences,
À paraître ; accepté en décembre 2003.

Fluid modelling of packets switched networks: perspectives for congestion control.
F. Chatté, B. Ducourthial, D. Nace et S.-I. Niculescu.
Special issue of IJSS (International Journal of System Sciences)
Décembre 2002.

Conférences internationales

Stability analysis in High-performances networks: a time-domain approach
S.-I. Niculecu, W. Michiels, D. Melchor-Aguillar, F. Mazenc et F. Chatté
À paraître dans IASTED APPLIED SIMULATION AND MODELLING
Grèce, Juin 2004

Robustness issues of fluid approximation for congestion detection in best effort networks.
F. Chatté, B. Ducourthial, et S.-I. Niculescu
Proceedings of 7th IEEE Symposium on Computers and Communication (ISCC 2002),
Italie, Juillet 2002.

Results on fluid modeling of packet switched networks.
F. Chatté, B. Ducourthial, D. Nace et S.-I. Niculescu.
Proceedings of 3rd IFAC Workshop on Time Delay Systems (IFAC TDS 2001),
USA, Décembre 2001.

Conférence nationale

Modélisation en continu des réseaux à commutation de paquets : perspectives pour la qualité de service.
F. Chatté et B. Ducourthial.
Actes d'AlgoTel 2002,
Mèze, France, Mai 2002 (INRIA).

Rapports internes

État de l'art des protocoles de transport (version longue actualisée).
F. Chatté et B. Ducourthial.
Avril 2004.

Construction d'un correcteur PI pour le contrôle de congestion. F. Chatté, B. Ducourthial et S.-I. Niculescu.
Février 2004.

Fluid modelling of packets switched networks: perspectives for congestion control (extended version)
F. Chatté, B. Ducourthial, D. Nace et S.-I. Niculescu.
Décembre 2002.

Robustness issues of fluid approximation for congestion detection in best effort networks (extended version)
F. Chatté, B. Ducourthial et S.-I. Niculescu.
Juillet 2002.

1.3.4 Plan du document

Dans ce manuscrit, nous commençons au chapitre 2 par étudier différents protocoles de transport *unicast* implémentant un mécanisme de contrôle de congestion. Le protocole de transport le plus utilisé sur Internet est sans conteste TCP. L'une des versions de TCP les plus répandues sur Internet est TCP Reno [90]. Son fonctionnement est présenté dans la section 2.1.

Pour améliorer le contrôle de congestion de TCP, les protocoles TCP Sack, TCP New Reno et TCP Fack proposent de mettre en place une meilleure gestion des acquittements. Leur fonctionnement est décrit dans la section 2.2. Les protocoles TCP Vegas, TCP-L et TCP Westwood proposent d'optimiser la gestion de la fenêtre de congestion. Leur fonctionnement est présenté dans la section 2.3.

De nouveaux protocoles n'utilisant pas de fenêtre d'émission pour réguler leur débit sont décrits dans la section 2.4. Enfin, des techniques d'évitement de congestion basées sur des informations provenant directement des routeurs sont présentées dans la section 2.5.

Dans le chapitre 3, nous commençons par présenter dans la section 3.1 deux études (représentatives du domaine) portant sur la modélisation continue des réseaux. Ensuite, nous nous intéressons à la validité de l'approximation continue d'un réseau à commutation de paquets (intrinsèquement discret). Dans la section 3.2, nous présentons les modèles continu et discret utilisés dans cette étude. La méthode de comparaison des deux modèles est présentée dans la section 3.3. Enfin, les résultats obtenus sont décrits et analysés dans la section 3.4.

Dans le chapitre 4, un contrôleur de type PI (« Proportionnel - Intégral ») permettant de réguler le débit d'émission des sources est développé. Le choix du type de correcteur ainsi que des variables de contrôle et de commande est justifié dans la section 4.1. Le correcteur est ensuite construit pas à pas, les différentes étapes sont décrites dans les sections 4.2 à 4.5. Une fois le correcteur construit et validé dans un environnement continu, sa discrétisation est présentée dans la section 4.6.

Afin d'évaluer les performances du nouveau protocole obtenu (appelé Primo), une méthode d'évaluation a été définie dans le chapitre 5. Pour commencer, nous discutons dans la section 5.1 le mode d'évaluation à utiliser. Le mode d'évaluation choisi étant la simulation, le simulateur *ns*[30] est présenté dans la section 5.2. Puis les topologies ainsi que les paramètres du réseau utilisés lors des simulations sont choisis dans la section 5.3. Enfin, les critères d'évaluation ainsi que les indicateurs s'y rapportant sont définis dans la section 5.4.

Une étude comparative de différents protocoles de transport – dont Primo – implémentant un mécanisme de contrôle de congestion est proposée dans la chapitre 6. Cette étude permet

30. *network simulator.*

de mesurer et d'analyser, en fonction d'une situation donnée, l'efficacité d'un mécanisme de contrôle de congestion.

Pour finir, un bilan général des études présentées dans ce manuscrit est dressé dans le chapitre 7. Ce chapitre présente également les perspectives qu'ouvrent ces travaux.

Les dernières pages de ce manuscrit sont réservées à des annexes comprenant :

- des rappels sur les protocoles IP, UDP et ICMP, respectivement proposés dans les annexes A.1, A.2 et A.3 ;
- les codes sources du protocole utilisé pour étudier la cohérence des modèles continu et discret. Le code utilisé sous *matlab* est fourni dans l'annexe B.1 et celui utilisé sous *ns* est fourni dans l'annexe B.2 ;
- les codes sources continu et discret de Primo. Le code décrivant le fonctionnement du contrôleur sous *matlab*, est fourni dans l'annexe C.1. Le code de l'implémentation discrète de Primo sous *ns* est fourni dans l'annexe C.2 ;
- les tableaux récapitulant les résultats obtenus par les protocoles testés lors des évaluations présentées au chapitre 6. Ces tableaux sont présentés dans l'annexe D.

Chapitre 2

Un état de l'art sur les protocoles de type TCP incluant un contrôle de congestion

Dans ce chapitre, nous présentons un état de l'art des différentes techniques de contrôle de congestion développées dans les protocoles de transport *unicast*. Le contrôle de congestion est apparu dans TCP Tahoe. Puis rapidement des améliorations lui ont été apportées dans TCP Reno ; cette version de TCP est présentée dans la section 2.1.

Le protocole TCP assure un transport fiable d'un flot de données grâce aux acquittements. L'idée de base est d'émettre un nouveau segment lorsque le précédent a été reçu. En réalité, pour améliorer le débit d'émission, TCP autorise l'émission de plusieurs segments sans avoir à attendre leur acquittement. La quantité de données envoyées mais non encore acquittées est gérée par une fenêtre d'émission. Plus cette fenêtre est grande, plus le débit d'émission sera important ; la gestion de sa taille est donc primordiale. Ces caractéristiques laissent apparaître les types d'améliorations susceptibles d'être apportées au contrôle de congestion : amélioration de la gestion des acquittements ou de celle de la taille de la fenêtre de congestion. Les protocoles TCP Sack, TCP New Reno, et TCP Fack proposent d'améliorer la gestion des acquittements. Leur fonctionnement est présenté dans la section 2.2. Les protocoles TCP Vegas, TCP Westwood et TCP-L proposent d'améliorer la gestion de la fenêtre de congestion. Leur fonctionnement est décrit dans la section 2.3.

Le développement des applications multimédia a donné naissance à de nouveaux protocoles n'utilisant pas de fenêtre d'émission pour réguler leur débit. Ces nouveaux protocoles proposent de cohabiter équitablement avec TCP et d'offrir un débit lissé (nécessaire aux flots multimédia), tout en respectant les contraintes liées au contrôle de congestion. Des travaux parmi les plus représentatifs du domaine sont présentés dans la section 2.4.

Les techniques de prévention des congestions basées sur un retour d'information de la part des routeurs (ECN[1], ICMP) sont présentées dans la section 2.5.

1. *Explicit Congestion Notification.*

2.1 Transmission Control Protocol (TCP)

2.1.1 Caractéristiques et fonctionnement général

Le protocole TCP a été conçu dans les années soixante dix. Aujourd'hui, c'est le protocole de transport le plus utilisé sur Internet (95% des données transitant entre l'Europe et les États Unis sont transportées par TCP [32]). Depuis sa création, ce protocole a subi de nombreuses modifications. Il a été standardisé pour la première fois en 1981 dans la RFC 793 [95], devenue STD 7. Cette version a ensuite été amendée en 1989 par la RFC 1122 [16], puis clarifiée en 1997 par la RFC 2001 [109], pour finir par être mise à jour en 1999 par la RFC 2581 [6]. Dans ces conditions, il est facilement imaginable que de nombreuses versions de TCP cohabitent actuellement sur Internet.

La première version de TCP à avoir implémenté un mécanisme de contrôle de congestion basé sur l'article de V. Jacobson [56] est fréquemment nommée TCP Tahoe (ce nom fait référence au système BSD Tahoe dans lequel cette pile TCP a été implémentée). De nos jours, la version la plus répandue sur Internet est appelée TCP Reno [38]. Cette version apporte à TCP Tahoe une meilleure gestion du débit d'émission (réduction adaptée au type de détection de congestion) et est conforme aux RFC 793, 1122 et 2581. Les techniques de contrôle de congestion utilisées dans cette version de TCP seront décrites dans la suite du chapitre.

Contrairement à UDP, TCP [24, 110] est un protocole orienté connexion, fiable et permettant de transmettre un flux d'octets. Le terme orienté connexion signifie que deux machines souhaitant communiquer *via* TCP doivent commencer par établir une connexion avant de pouvoir dialoguer. L'établissement de la connexion donne lieu à un échange d'informations entre les deux machines avant qu'elles ne commencent réellement à transmettre des données. Une fois la connexion établie, les deux extrémités de la connexion peuvent communiquer de manière bidirectionnelle (*full-duplex*). Cette connexion peut s'apparenter à une communication téléphonique. Afin de simplifier les explications nous distinguerons cependant les deux cotés de la connexion, l'un s'appellera « émetteur » et l'autre « récepteur ».

Avec TCP, les données sont transmises sous forme de flux d'octets, ce qui signifie que les données fournies par l'application ne sont pas segmentées. Les données à transmettre sont coupées à des endroits quelconques par TCP pour être envoyées sur le réseau sous la forme de segments. Les segments seront ensuite ré-assemblés par le récepteur afin de reformer un flux d'octets qu'il fournira à l'application destinataire. Notons que la taille des segments peut varier mais elle reste toujours inférieure à MSS[2] octets. Cette constante est fréquemment initialisée à la plus grande taille admise par les paquets IP avant fragmentation. Si les données fournies par l'application sont jugées trop « petites » pour être émises directement, l'émetteur peut choisir d'attendre pour regrouper des données avant de les émettre dans un même segment [79]. Certaines applications n'acceptent pas que TCP retarde l'émission des données afin de les regrouper. Dans ce cas, elles utilisent la directive « *push* » pour forcer l'émetteur à envoyer immédiatement les données.

La fiabilité fournie par TCP consiste à s'assurer que chaque octet émis par l'émetteur sera bien transmis à l'application destinataire grâce à un système d'acquittements. De plus, le récepteur vérifie l'intégrité de chaque segment reçu grâce à une somme de contrôle (*checksum*). Si la somme de contrôle est correcte, le récepteur enverra un acquittement à l'émetteur

2. *Maximum Segment Size*.

pour lui spécifier que les données seront transmises à l'application destinataire. En revanche, si la somme de contrôle est fausse, le segment reçu sera ignoré et donc non acquitté. Lorsque l'émetteur s'apercevra que le segment émis n'a pas été acquitté il le ré-émettera dès que possible (*cf.* paragraphe 2.1.4).

Afin de ne pas engorger le destinataire, l'émetteur gère son débit d'émission grâce à un mécanisme de fenêtre de réception. Dans les acquittements, un champ « *Win* » indique à l'émetteur l'espace dont dispose le récepteur pour stocker temporairement les données avant de les transmettre à l'application destinataire. De même, pour ne pas saturer le réseau, l'émetteur utilise une fenêtre de congestion (*cf.* paragraphe 2.1.5) servant à limiter son débit d'émission. La taille de cette fenêtre détermine la quantité de données que l'émetteur peut émettre sans recevoir d'acquittements.

2.1.2 Segment TCP

Structure. La figure 2.1 donne la structure d'un segment TCP. L'en-tête d'un segment se compose des champs suivants :

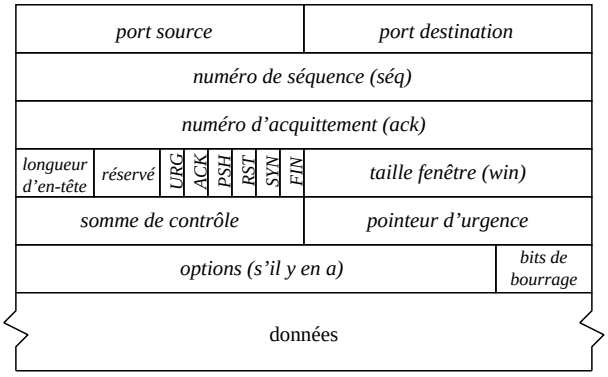

FIG. 2.1 – *Structure d'un segment TCP*

- Les champs *port source* et *port destination* identifient les applications émettrice et réceptrice.
- Le champs *numéro de séquence*[3] donne la position du premier octet du segment dans le flux de données envoyé par l'émetteur. Ce numéro permet au récepteur de ré-ordonner les données avant de les transmettre à l'application destinataire.
- Le champ *numéro d'acquittement* donne le prochain numéro de séquence attendu par le récepteur (*i.e.*, numéro du dernier octet correctement reçu + 1).
- Le champ *longueur d'en-tête* contient la taille de l'en-tête, qui peut varier de 20 à 60 octets suivant les options utilisées.
- Le champs *réservé* comporte six bits réservés à un usage ultérieur.
- les six drapeaux qui suivent permettent de spécifier le rôle et le contenu du segment. Il permettent ainsi d'interpréter correctement certains champs de l'en-tête. La signification de ces drapeaux, lorsqu'ils sont positionnés à « vrai », est la suivante :

3. Il s'agit d'un entier non signé codé sur 32 bits qui retourne à 0 après avoir atteint sa valeur maximale.

  - *URG* : le pointeur de données urgentes est valide.
  - *ACK* : le numéro d'acquittement est valide.
  - *PSH* : les données doivent être émises dès que possible même si la quantité de données à émettre est faible.
  - *RST* : la connexion va être ré-initialisée.
  - *SYN* : les numéros de séquence de la connexion sont en cours d'initialisation.
  - *FIN* : la fermeture de la connexion est en cours de négociation.
- Le champ *taille de la fenêtre* permet au récepteur d'indiquer à l'émetteur l'espace dont il dispose pour stocker temporairement les segments reçus avant de les transmettre à l'application.
- Le champ *somme de contrôle* est utilisé pour vérifier l'intégrité de l'en-tête et des données transmises. Le calcul de cette somme de contrôle porte sur les données et une pseudo en-tête constituée des adresses IP source et destination, du numéro d'identification du protocole (6 pour TCP) et de la taille du segment.
- Le champ *pointeur d'urgence* est un entier positif qui, additionné au numéro de séquence, pointe sur le premier octet de données urgentes[4]. Ce champ sera pris en compte par le récepteur seulement si le drapeau *URG* est positionné à « vrai ».
- Le champ *option* et les données peuvent être ou non utilisés. En effet, certains segments (dits de gestion) ne contiennent pas de données et n'utilisent aucune option. Les données et le champ *option* peuvent éventuellement être complétés par des bits de bourrage (qui n'ont aucune signification) afin que la taille du segment soit un multiple de 32 bits.

Taille des segments. La taille des segments TCP peut varier durant une connexion, sans jamais dépasser une taille maximale (MSS). Il est généralement intéressant que la taille maximale des segments corresponde à la taille maximale des datagrammes IP avant qu'ils ne soient fragmentés. C'est pourquoi, en général, la taille maximale d'un segment TCP est égale au minimum entre la taille du tampon d'émission, du tampon de réception, des MTU[5] et des MRU[6] du chemin emprunté.

Les tampons d'émission et de réception ont une taille de l'ordre de 8 à 16 ko. L'émetteur connaît la taille du tampon du récepteur grâce au champ *taille fenêtre* des en-têtes TCP. Pour découvrir le plus petit MTU des réseaux empruntés entre émetteur et récepteur, on peut utiliser la technique *Path MTU Discovering*, qui consiste à envoyer un datagramme IP le plus grand possible avec le drapeau « *Don't Fragment* » à « vrai ». Si le paquet est trop gros pour l'une des machines traversées, le protocole ICMP retournera une erreur et l'émetteur pourra effectuer une nouvelle tentative avec un datagramme plus petit[7].

2.1.3 La connexion

Comme nous venons de le voir, une connexion est identifiée par une paire de *sockets* (association des numéros de port aux adresses IP [110]). Notons que plusieurs connexions peuvent

4. Il existe des différences notables sur la gestion de ce pointeur dans les diverses implémentations de TCP, ce qui nuit à la portabilité des applications l'utilisant.

5. *Maximum Transmission Unit*.

6. *Maximum Receive Unit*.

7. Cependant, certains routeurs ignorent ces messages d'erreur pour des raisons de sécurité.

partager une même *socket* mais pas les deux.

Établissement d'une connexion. L'établissement d'une connexion se déroule en trois phases (*cf.* figure 2.2) :

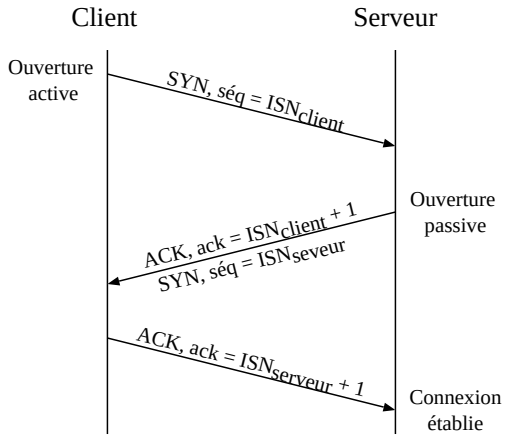

FIG. 2.2 – *Établissement d'une connexion TCP.*

- Une application, généralement appelée « client », effectue une ouverture active en demandant à son TCP d'établir une connexion avec la *socket* distante. Le client fournit donc à son TCP le numéro de port de l'application[8] qu'il cherche à joindre ainsi que l'adresse IP de la machine où se trouve cette application.
  La demande du « client » est ensuite envoyée à la machine où se trouve l'application « serveur ». Pour cela, la machine du client envoie un segment TCP ne contenant aucune donnée, et ayant son drapeau *SYN* à « vrai » pour indiquer qu'il faut synchroniser les numéros de séquence. Le numéro de séquence ISN_{client}[9] contenu dans ce segment est tiré au sort. Le drapeau *ACK* du premier segment est positionné à « faux » afin d'indiquer que la valeur contenue dans le champs *numéro d'acquittement* n'est pas à prendre en compte.

- Le « serveur », de son coté, a effectué une ouverture passive en créant une *socket* locale et en acceptant les connexions sur ce point d'entrée. À la réception du premier segment, le « serveur » reconnaît par le drapeau *SYN* qu'il s'agit de l'établissement d'une nouvelle connexion. Si cette connexion est réalisable, le « serveur » envoie un acquittement au client. Le segment émis ne contient aucune donnée, ses drapeaux *SYN* et *ACK* sont positionnés à « vrai », pour indiquer qu'il effectue la synchronisation des numéros de séquence et que le champ *numéro d'acquittement* est à prendre en compte. Le champ *numéro de séquence* de cet acquittement contient la valeur $ISN_{serveur}$ (tirée au sort) et le champ *numéro d'acquittement* contient la valeur $ISN_{client} + 1$ (ce numéro correspond au premier octet de données qui sera transmis par le client [11, 16]).
  Dans le cas où la connexion n'est pas réalisable (*e.g.*, mauvais numéro de port), le « serveur » avertit le « client » de l'impossibilité d'établir la connexion.

8. Un certain nombre de numéros de ports a été prédéfini pour les applications courantes par l'IANA (*Internet Assigned Number Authority*) : port 80 pour HTTP, 20 pour FTP, 23 pour TELNET, *etc.*

9. *Initial Sequence Number*.

- À la réception du premier segment émis par le « serveur », la machine cliente accuse réception de ce segment en envoyant un segment, ayant drapeau *SYN* positionné à « faux », et dont le champ *numéro d'acquittement* contient le numéro de séquence $ISN_{serveur}$ +1 (numéro de séquence du prochain segment qu'émettera le « serveur »).

Notons qu'il est possible que les deux extrémités de la connexion effectuent une ouverture active simultanée.

Fermeture de la connexion. La fermeture de la connexion peut être demandée indifféremment par l'une des deux extrémités de la connexion. La fermeture d'une connexion se compose de deux demi-fermetures puisque deux flots de données peuvent être transmis simultanément (un dans chaque sens). La fermeture d'une connexion se déroule de la manière suivante (dans la figure 2.3, c'est le client qui demande la fermeture de la connexion) :

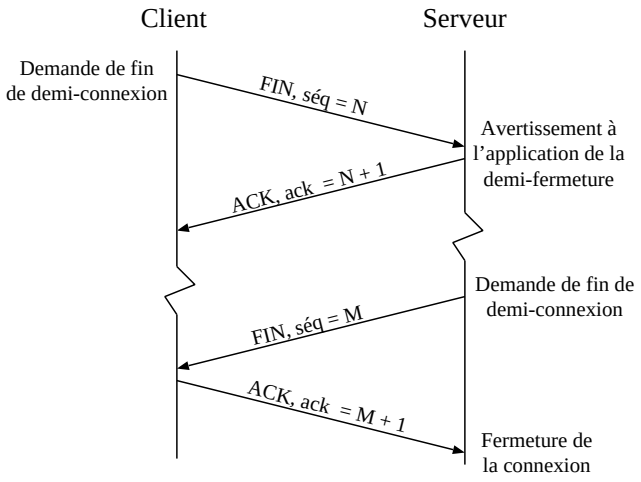

FIG. 2.3 – *Fermeture d'une connexion TCP.*

- En envoyant un segment vide ayant le drapeau *FIN* positionné à « vrai », avec le champ *numéro de séquence* incrémenté, le client informe le serveur qu'il souhaite mettre fin à la connexion.
- À la réception de ce segment, le serveur l'acquitte et informe l'application de la demi-fermeture de la connexion. À partir de là, les données ne peuvent plus transiter que dans un sens (du « serveur » vers le « client »). Lorsque le serveur a terminé d'envoyer des données, il peut à son tour fermer sa demi-connexion en envoyant un segment ayant le drapeau *FIN* positionné à « vrai ».
- À la réception de ce segment, le client envoie un acquittement qui sera le dernier segment émis sur cette connexion. Le champ *numéro d'acquittement* de ce segment contient encore le prochain numéro de séquence attendu, mais le drapeau *FIN* est positionné à « faux ». Suite à l'émission de ce dernier segment, la partie « client » de la connexion sera fermée après une attente d'une durée de deux MSL [10]. Cette attente permet de s'assurer que l'autre extrémité a bien reçu l'acquittement final (*i.e.*, on attend de voir si l'autre extrémité ne renvoie pas son segment *FIN* après l'expiration de son délai d'attente).

10. *Maximum Segment Lifetime*.

- La réception de ce dernier segment par le « serveur » entraîne la fermeture de la deuxième moitié de la connexion, qui disparaît alors totalement.

Notons qu'une connexion peut aussi être fermée de manière inopinée à cause d'un problème quelconque. Dans ce cas, l'extrémité souhaitant mettre fin immédiatement à la communication envoie un segment ayant le drapeau *RST* positionné à « vrai ». Ce type de fermeture ne permet pas de garantir que les segments en transit auront tous été reçus avant la fermeture de la connexion.

Lorsque l'une des extrémités de la connexion n'effectue pas d'émission régulière, et que l'autre extrémité soupçonne une rupture de la connexion, cette dernière peut émettre un segment *keep alive*. La RFC 1122 indique que ce mécanisme reste optionnel et doit être utilisé uniquement s'il est indispensable au fonctionnement de l'application.

2.1.4 Gestion des acquittements

Les acquittements sont utilisés pour fiabiliser la communication. En effet, chaque octet émis dans un flux TCP sera acquitté par le récepteur. Tout segment non acquitté sera considéré comme perdu et sera ré-émis. Sachant que la perte d'un segment est généralement due à une congestion sur le réseau, les acquittements servent également à réguler le débit d'émission des sources TCP.

Dans certaines conditions, il se peut que les acquittements ne parviennent plus à l'émetteur. Dans ce cas, après un délai d'attente fixé, tous les segments émis et non acquittés sont ré-émis. Ce délai d'attente (RTO[11]) est géré par le chien de garde de retransmission et est calibré en fonction du délai d'aller-retour entre l'émetteur et le récepteur (noté RTT[12]). Le RTT est lui-même mesuré grâce aux dates de réception des acquittements.

Mise en œuvre. Comme nous l'avons vu précédemment, chaque octet du flot transmis est numéroté avec un numéro de séquence, à partir du premier octet transmis, qui porte le numéro ISN+1 [11, 16]. L'en-tête d'un segment envoyé contient le numéro de séquence du premier octet de données ainsi que la quantité de données qu'il transporte. À la réception d'un segment, le récepteur enverra un acquittement (*cf.* figure 2.4) dont le champ *numéro d'acquittement* contiendra le dernier numéro d'octet reçu + 1 (*i.e.*, numéro de séquence du segment reçu + quantité de données contenue dans le segment). L'acquittement envoyé indiquera à la source que le récepteur est prêt à recevoir les données à partir de l'octet *numéro d'acquittement* (*cf.* en-tête des segments).

Pour éviter de charger inutilement le réseau, les acquittements ne sont pas forcément émis dès la réception d'un segment. En effet, au lieu d'envoyer un acquittement à chaque réception, le récepteur retarde l'émission des acquittements (*delayed acknowledgment*) en espérant avoir des données à transmettre vers l'émetteur (ou tout simplement recevoir d'autres segments afin de tous les acquitter en même temps). Si le récepteur a des données à transmettre en direction de l'émetteur, l'acquittement qu'il enverra contiendra également des données. On parle dans ce cas d'acquittement superposé (*piggyback*). Notons que l'émission d'un acquittement ne peut pas être retardée de plus de 500 ms . Notons également que si deux segments de taille maximale (MSS) sont reçus, leur acquittement ne pourra pas être retardé.

11. *Retransmit Time Out*.
12. *Round Trip Time*.

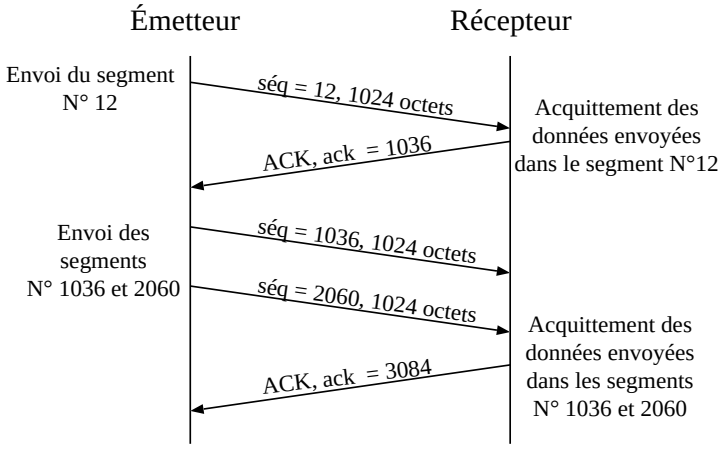

FIG. 2.4 – *Acquittement de plusieurs segments.*

En général, un émetteur TCP n'attend pas qu'un segment émis soit acquitté pour émettre le suivant. De ce fait, le récepteur peut recevoir plusieurs segments de suite et n'envoyer qu'un seul acquittement pour toutes les données reçues. On parle dans ce cas d'acquittements cumulés (*cf.* second acquittement dans la figure 2.4). Néanmoins, si plusieurs segments sont envoyés à la suite et que le premier est perdu, le récepteur ne pourra acquitter aucune des données reçues. Dans ce cas, le récepteur indiquera à l'émetteur qu'il a reçu des données (sans pour autant les acquitter) en envoyant des acquittements indentiques au dernier émis (on parle d'acquittements dupliqués). L'émission d'acquittements dupliqués prend fin à la réception du segment manquant.

Détection et retransmission des segments perdus. La détection de la perte d'un segment peut se faire de deux manières :

- Expiration du chien de garde de retransmission : si aucun segment n'a été acquitté depuis un certain temps (RTO ms), le chien de garde de retransmission vient à expirer. Dans ce cas, tous les segments émis depuis le premier segment non acquitté sont retransmis et le chien de garde est ré-armé avec un délai d'attente doublé [91]. Afin d'éviter que le chien de garde n'expire prématurément, il est ré-armé à chaque réception d'un nouvel acquittement. Si tous les segments émis ont été acquittés, le chien de garde de retransmission est désactivé.
- réception de trois acquittements dupliqués : comme nous venons de le voir, lorsqu'un segment est perdu, la réception des segments le suivant entraîne l'émission d'acquittements dupliqués. Lorsque l'émetteur reçoit trois acquittements dupliqués (*i.e.*, quatre acquittements pour le même segment), le segment suivant celui qui a été acquitté quatre fois sera considéré comme perdu [16]. En effet, il est exceptionnel que le désordre dans l'ordre d'arrivée soit supérieur à deux segments. Une détection de perte de segment par trois acquittements dupliqués entraîne simplement la ré-émission du segment perdu (*i.e.*, les segments suivants sont supposés reçus [6]). Notons que cette méthode de détection de pertes est généralement plus rapide que celle utilisant l'expiration du chien de garde.

  Pour que cette méthode de retransmission soit efficace, il faut que le récepteur conserve les segments reçus après celui perdu, de sorte qu'à la réception du segment perdu, tous

les octets du flot soient transmis à l'application.

Estimation du délai d'aller-retour (RTT) et retransmission (RTO). Le délai d'aller-retour (RTT) est mesuré grâce aux dates d'émission des segments et de réception de leur acquittement. Le RTT utilisé par TCP pour calculer le RTO est obtenu avec l'algorithme de V. Jacobson [56]. Cet algorithme fournit une valeur lissée du RTT en effectuant une moyenne pondérée de la valeur actuelle du RTT lissé et de la dernière mesure du RTT instantané. Il permet également de calculer un indice de variation du RTT (RTTVAR) :

$$\begin{aligned} \text{RTT} &= \alpha \times \text{RTT} + (1-\alpha) \times \text{RTT}_{\text{inst}} \\ \text{RTTVAR} &= (1-\beta) \cdot \text{RTTVAR} + \beta \cdot |\text{RTT} - \text{RTT}_{\text{inst}}| \end{aligned} \tag{2.1}$$

Dans ces équations, α et β sont des coefficients de lissage dont la valeur est comprise entre 0 et 1. Plus la valeur de ces coefficients est faible, plus la dernière mesure du RTT_{inst} aura de l'influence sur le calcul du RTT lissé et de sa variation RTTVAR. Il est recommandé dans [59] de prendre $\alpha = \frac{1}{8}$ et $\beta = \frac{1}{4}$.

L'utilisation d'acquittements cumulés (rendant imprécise les dates utilisées pour mesurer les RTTs instantanés) nuit à la précision de cet algorithme. Pour y remédier, l'option *timestamp* [58] (remplaçant les options *echo* et *echo reply* [57]) a été introduite. Cette option permet de placer dans l'en-tête des segments une estampille indiquant leur heure d'émission. À l'arrivée de ces segments au récepteur, l'estampille est recopiée dans l'en-tête des acquittements. À la réception des acquittements, l'emetteur pourra mesurer le délai d'aller-retour de manière exacte en soustrayant la valeur contenue dans l'estampille à l'heure d'arrivée de l'acquittement.

À l'établissement de la connexion, ne connaissant pas le délai d'aller-retour, le chien de garde de retransmission est arbitrairement armé à 3 s [16, 91]. Puis, sa valeur est affinée en fonction du RTT lissé et de sa variation (RTTVAR).

Chaque évaluation du RTT et de sa variation donne lieu à la mise à jour du RTO [91] avec l'équation suivante :

$$\text{RTO} = \text{RTT} + \max(\text{P}, \text{K} \times \text{RTTVAR}) \tag{2.2}$$

Dans cette équation, K représente une constante fixée à 4, et P la granularité des mesures du RTT (P = 500 ms dans le système BSD).

Lorsque des segments sont perdus et que l'option *timestamp* n'est pas utilisée, l'estimation du RTT (et donc du RTO) pose problème. En effet, lorsqu'un segment est considéré comme perdu, il est ré-émis, mais à la réception de son acquittement l'émetteur ne sait pas quelle date d'émission utiliser (celle du premier segment émis ou celle de la ré-émission) pour calculer le RTT. Pour résoudre ce dilemme, l'algorithme de Karn [61] propose d'ignorer les RTT_{inst} des segments retransmis dans le calcul du RTT lissé, ce qui évite d'influencer le RTO.

La RFC 1122 [16] indique que l'emploi des algorithmes de Van Jacobson et de Karn dans les implémentations de TCP améliore les performances.

2.1.5 Gestion du débit d'émission

Fenêtre d'émission glissante. La gestion du débit d'émission dans TCP est assurée par un système fenêtre d'émission glissante. La fenêtre d'émission détermine la quantité de données qui peut être émise sans être acquittée. Plus cette quantité est importante, plus le débit d'émission

de la source le sera. L'utilisation d'une fenêtre glissante consiste à déterminer quels sont les octets du flux de données que la source peut émettre (*cf.* figure 2.5). La fenêtre d'émission contient les octets à émettre (octets 8 et 9 dans l'exemple) ainsi que les octets émis mais non acquittés (octets 4 à 7). Lorsque l'octet le plus à gauche dans la fenêtre (octet 4) aura été acquitté, la fenêtre se déplacera d'un cran vers la droite (elle couvrira alors les octets 5 à 10).

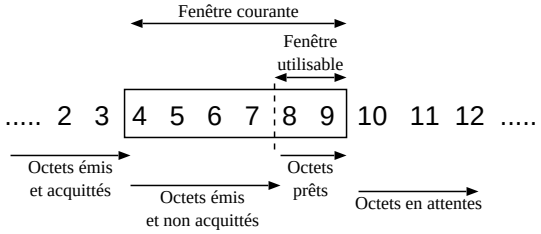

FIG. 2.5 – *Exemple de fonctionnement d'une fenêtre d'émission glissante.*

Lorsque la taille de la fenêtre d'émission augmente, l'émetteur déplace le bord droit de la fenêtre d'émission et émet immédiatement les octets qui viennent d'y entrer. En revanche, losque la taille de la fenêtre d'émission diminue, il est déconseillé (et souvent impossible) de déplacer le bord droit de la fenêtre vers la gauche. Le rétrécissement est alors opéré lors du glissement de la fenêtre à l'arrivée des accusés de réception. La taille de la fenêtre d'émission dépend de deux paramètres : la taille de la fenêtre de réception et la taille de la fenêtre de congestion. La taille de la fenêtre de réception est connue grâce au champ *taille de la fenêtre* contenu dans l'en-tête des acquittements. La taille de la fenêtre de congestion dépend, elle, de l'état de congestion du réseau. La taille de la fenêtre d'émission est égale à la plus petite des deux fenêtres (réception et congestion). Généralement, le réseau (*i.e.*, la fenêtre de congestion) est le facteur limitant l'augmentation du débit d'émission.

Principe de la fenêtre de congestion dans TCP Reno. Pour faire face au problème d'engorgement des réseaux, des mécanismes de contrôle de congestion ont été ajoutés aux spécifications de TCP dans la RFC 1122 [16]. Ces mécanismes permettent à TCP d'adapter son débit d'émission aux capacités du réseau. Ils sont décrits dans [56] et dans la RFC 2001 [109], qui a été mise à jour dans la RFC 2581 [6].

Comme nous l'avons vu précédemment, TCP peut envoyer plusieurs segments sans avoir reçu les acquittements correspondants. Cela peut s'apparenter à un effet « *pipeline* » dans la connexion émetteur-récepteur. Mais la quantité de données présente dans le « *pipeline* » ne doit pas dépasser les capacités du réseau sous peine de créer des congestions. Cette quantité de données en transit est régulée grâce à la taille de la fenêtre de congestion (*cwnd*[13]). Les ajustements successifs de la taille de la fenêtre de congestion en fonction des congestions détectées, ou, au contraire, des bonnes conditions de trafic rencontrées, doivent permettre à l'émetteur de déterminer le débit d'émission optimal. Les ajustements de la taille de la fenêtre de congestion sont gérés par quatre algorithmes : *démarrage lent* (*slow start*), *évitement de congestion* (*congestion avoidance*), *retransmission rapide* (*fast retransmit*), et *reprise rapide* (*fast recovery*). La taille de la taille de la fenêtre de congestion évolue ainsi (figure 2.6) :

13. *congestion window*.

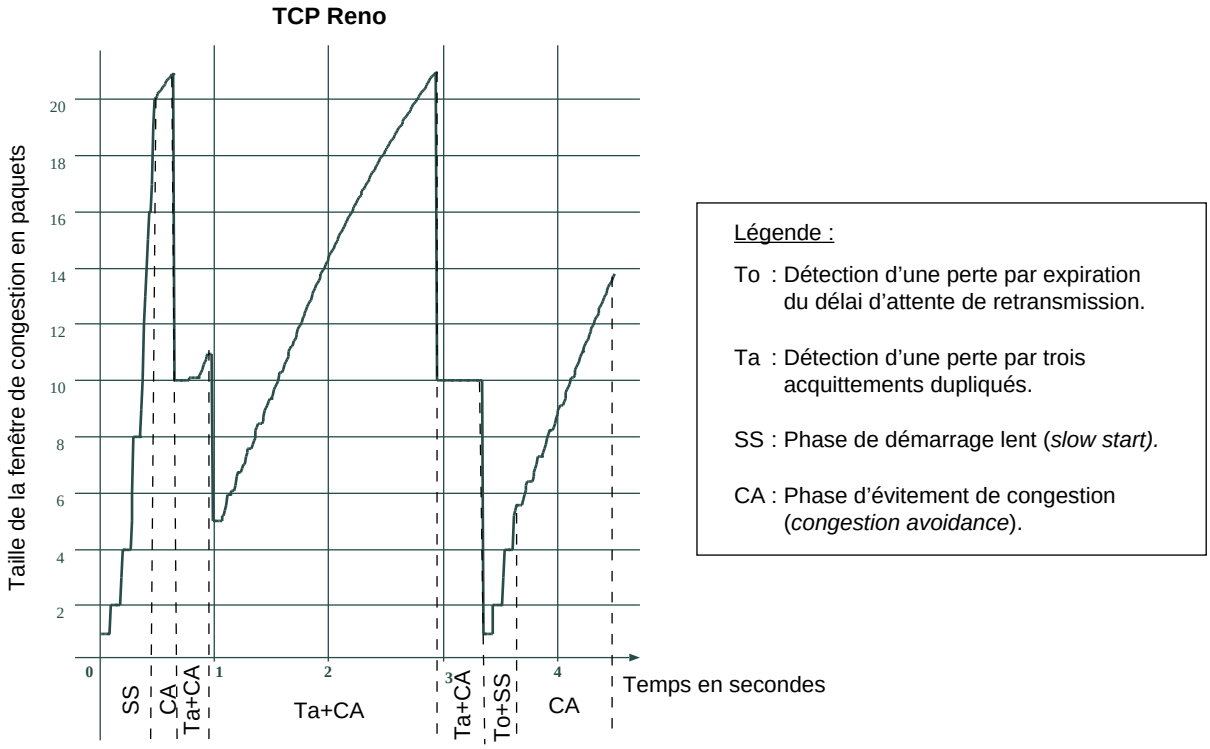

FIG. 2.6 – *Évolution de la taille de la fenêtre de congestion avec TCP Reno.*

– À l'initialisation de la connexion, la taille de la fenêtre de congestion (*cwnd*) a pour valeur IW[14]$= 2$ MSS octets. Au début de la communication, cette taille croît exponentiellement grâce à l'algorithme de *démarrage lent*, jusqu'à ce qu'elle atteigne le seuil *sstrhesh* [15]. En phase de *démarrage lent*, la taille de la fenêtre de congestion augmente de MSS octets pour chaque réception d'un acquittement.
– Lorsque la taille de la fenêtre de congestion dépasse le seuil de démarrage lent (*cwnd* $>$ *ssthresh*), la fenêtre s'arrête de croître exponentiellement. Le débit optimal étant proche, la taille de la fenêtre de congestion continue à croître mais de manière quasi-linéaire grâce à l'algorithme d'*évitement de congestion*. En phase d'*évitement de congestion*, la fenêtre augmente de MSS octets à chaque fois que la totalité de la fenêtre de congestion a été émise et acquittée.
– Lorsqu'une perte de segment est détectée (*i.e.*, une congestion s'est formée sur le réseau) grâce à l'expiration du chien de garde, ou *via* un message ICMP (*cf.* paragraphe 2.5.1), la fenêtre de congestion est réduite à sa taille initiale *IW* et l'algorithme de *démarrage lent* est relancé. Notons qu'avant d'entrer en phase de *démarrage lent*, le seuil *ssthresh* est recalculé :

$$ssthresh = \max\left(\frac{\text{taille de la fenêtre de congestion avant réduction}}{2}, IW\right) \qquad (2.3)$$

– Si la perte d'un segment est détectée grâce à trois acquittements dupliqués, le segment fautif est imméditement ré-émis grâce à l'algorithme de *retransmission rapide*. Après la

14. *Initial Window*.
15. *slow start threshold*.

ré-émission du segment perdu, l'algorithme de *reprise rapide* est lancé et le seuil *ssthresh* est réduit (*cf.* équation 2.3). L'algorithme de *reprise rapide* réduit de moité la taille de la fenêtre de congestion (sans qu'elle devienne inférieure à *IW*). Puis elle est artificiellement augmentée de trois MSS, car les trois acquittements dupliqués reçus signifient que trois segments ont quitté le réseau. À chaque arrivée d'un acquittement dupliqué, la fenêtre est augmentée d'un segment. Lorsqu'un acquittement non dupliqué est reçu, l'algorithme de *reprise rapide* prend fin. La fenêtre est alors réduite d'autant qu'elle a été artificiellement augmentée pour la réception d'acquittements dupliqués et l'algorithme d'*évitement de congestion* est lancé.

2.2 Améliorer la gestion des acquittements

L'émetteur utilisant les acquittements pour s'informer sur l'état de congestion du réseau, une meilleure gestion du mécanisme d'acquittements améliorerait le contrôle de congestion. Des travaux ont été menés dans ce sens ; nous présentons les principaux dans cette section.

2.2.1 TCP *Selective acknowlegment* (Sack)

Principe. L'option TCP Sack [39] a pour but d'enrichir les informations contenues dans les acquittements en ce qui concerne les segments perdus. Dans TCP Reno, lorsque plusieurs segments consécutifs sont perdus, et que la perte est détectée par trois acquittements dupliqués, l'émetteur n'a pas la possibilité de savoir combien de segments ont été perdus. Il ne ré-émet donc que le premier segment perdu, puis continue d'émettre de nouvelles données. Si plusieurs segments ont été perdus, le chien de garde de retransmission viendra à expirer car ces segments n'auront pas tous été retransmis. L'expiration du chien de garde entraînera une réduction inutile de la fenêtre de congestion à sa taille initiale *IW*. L'option Sack donne la possibilité au récepteur d'envoyer des acquittements spécifiques, indiquant les segments correctement reçus. Cette information supplémentaire permet à l'émetteur de mieux gérer la retransmission des données perdues. L'évolution la taille de la fenêtre de congestion de TCP Reno avec et sans l'option Sack est illustrée à la figure 2.7.

Mise en œuvre. L'option Sack est négociée entre les deux machines à l'initialisation de la connexion. Si l'une des deux extrémités de la connexion ne supporte pas l'option Sack, cette dernière ne pourra pas être utilisée.

Les acquittements Sack permettent de spécifier les segments correctement reçus : les blocs de données contiguës (reçus) sont indiqués *via* leur numéro de séquence de début et de fin plus un. Selon l'espace disponible dans le champ « options » des en-têtes des segments TCP, un acquittement Sack peut spécifier trois à quatre blocs par acquittement. Notons que la signification du champ *ack* situé dans l'en-tête des segments ne change pas.

L'option Sack impose au récepteur les règles suivantes [39] :

- Le premier bloc d'un acquittement Sack doit inclure le numéro de séquence du segment ayant déclenché l'émission de cet acquittement. Cette règle assure que les acquittements reflètent les informations les plus récentes sur l'état de la file du récepteur.
- Le récepteur doit inclure le plus possible de blocs distincts dans les acquittements Sack (un bloc ne peut pas être un sous-ensemble d'un autre bloc).

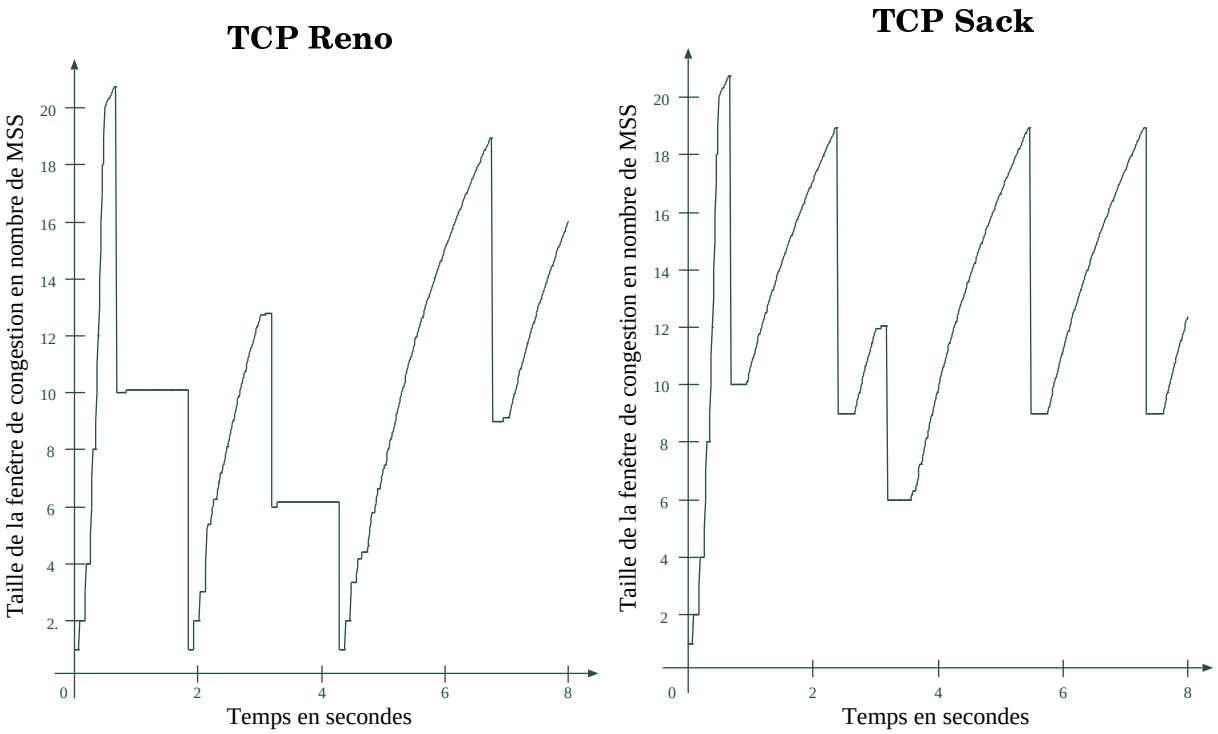

FIG. 2.7 – *Évolution de la taille de la fenêtre de congestion dans TCP Reno sans utiliser l'option Sack à gauche et en l'utilisant à droite.*

- Les blocs de données reçus seront spécifiés dans plusieurs acquittements Sack successifs. Cette redondance d'informations permet de pallier les pertes d'acquittements, et ainsi d'éviter au maximum les expirations inutiles du chien de garde de retransmission.

L'utilisation de l'option Sack ne modifie pas le contrôle de congestion (*démarrage lent, évitement de congestion, reprise rapide*). En revanche, à la détection d'une perte par trois acquittements dupliqués, l'émetteur sera capable de retransmettre tous les segments manquants (et pas seulement le premier). Une fois que tous les segments manquants sont reçus, le récepteur les acquittera sans utiliser l'option Sack. En effet, lorsqu'il n'y a aucun trou dans la séquence de réception, les acquittements émis n'utilisent pas cette option.

Dans la définition de l'option Sack, il est indiqué que l'émetteur possède une file de retransmission qui contient les segments émis mais non acquittés. La RFC 2018 propose d'ajouter un drapeau *Sacked* aux segments qui sont dans la file de retransmission. À la réception d'acquittements comportant des blocs Sack, l'émetteur compare les segments présents dans la file de retransmission à ceux acquittés. Lorsqu'un segment présent dans la file de retransmission est contenu dans l'un des blocs de l'acquittement, son drapeau *Sacked* et mis à « vrai ». À l'émission suivante, les segments ayant leur drapeau *Sacked* à « vrai » seront extraits de la file de retransmission. À la réception de trois acquittements dupliqués, les segments présents dans la file de retransmission seront immédiatement retransmis si :

– ils ont leur drapeau *Sacked* à « faux » ;
– et que leur numéro de séquence est inférieur à celui du plus haut segment acquitté dans un bloc Sack.

Notons que l'option Sack ne modifie pas le comportement de TCP lors d'une expiration

du chien de garde de retransmission : tous les segments émis depuis le premier segment non acquitté sont ré-émis.

2.2.2 TCP New Reno

Principe. Comme nous venons de le voir, les performances de TCP Reno se détériorent lorsque plusieurs segments consécutifs sont perdus. L'option Sack permet de résoudre ce problème grâce à l'enrichissement des informations contenues dans les acquittements. Malheureusement, l'option Sack a besoin d'être implémentée aux deux extrémités de la connexion pour fonctionner. L'option New Reno [35] propose de résoudre le problème des pertes consécutives en modifiant l'algorithme de *reprise rapide*. Cette modification n'impliquant que la partie émetteur du code de TCP, l'option New Reno n'a pas besoin d'être implémentée aux deux extrémités de la connexion pour fonctionner.

Notons que cette option est moins performante que Sack. Elle ne doit donc être utilisée que si l'option Sack n'est pas supportée par l'une des deux extrémités de la connexion.

Mise en œuvre. Les modifications apportées à l'algorithme de *reprise rapide* résident principalement dans l'ajout de la variable *recover* et dans la prise en compte d'acquittements partiels. Un acquittement partiel acquitte le segment retransmis sans acquitter tous les segments émis avant la détection de la perte.

Comme dans TCP Reno, à la réception du troisième acquittement dupliqué, l'émetteur détecte la perte d'un segment et entre en phase de *retransmission rapide*. En entrant dans cette phase, l'émetteur recopie le plus grand numéro de séquence émis dans la variable *recover* et retramsmet le segment perdu. Ensuite, comme dans l'algorithme de *reprise rapide* de TCP Reno, le seuil *ssthresh* est actualisé, la fenêtre de congestion prend la valeur de *ssthresh* augmentée de trois MSS.

À la réception d'un acquittement partiel (*i.e.*, n'acquittant pas le numéro de séquence *recover*), le premier segment non acquitté est ré-émis. Puis, la fenêtre de congestion est diminuée du nombre de nouveaux segments acquittés moins un et l'émetteur reste en phase de *reprise rapide*. S'il s'agit du premier acquittement partiel, le chien de garde de retransmission est réarmé. Si l'acquittement reçu est total (*i.e.*, il acquitte un numéro de séquence supérieur ou égal à *recover*), comme dans TCP Reno, la fenêtre de congestion est réduite à *ssthresh* et l'émetteur passe en phase d'*évitement de congestion*. L'évolution de la fenêtre de congestion (*cwnd*) de TCP New Reno est illustrée dans la figure 2.7.

La seconde modification apportée par TCP New Reno se situe au niveau de l'algorithme de *démarrage lent*, dans lequel est introduit la variable *send_hight*. Lorsque le chien de garde de retransmission expire, l'émetteur TCP New Reno passe en *démarrage lent* (comme pour TCP Reno). Dans cette phase, l'émetteur ré-émet tous les segments émis depuis celui perdu. Or, s'il retransmet des segments qui ont été correctement reçus, le récepteur enverra des acquittements dupliqués, qui risquent de déclencher une phase de *retransmission rapide* inutile.

Pour éviter de déclencher inutilement cette phase, à l'expiration du chien de garde, le plus haut numéro de séquence émis est mémorisé dans la variable *send_hight*. À la réception d'un acquittement dupliqué, le numéro d'acquittement qu'il contient est comparé à cette variable, cela permet de déterminer s'il doit être ignoré ou pris en compte.

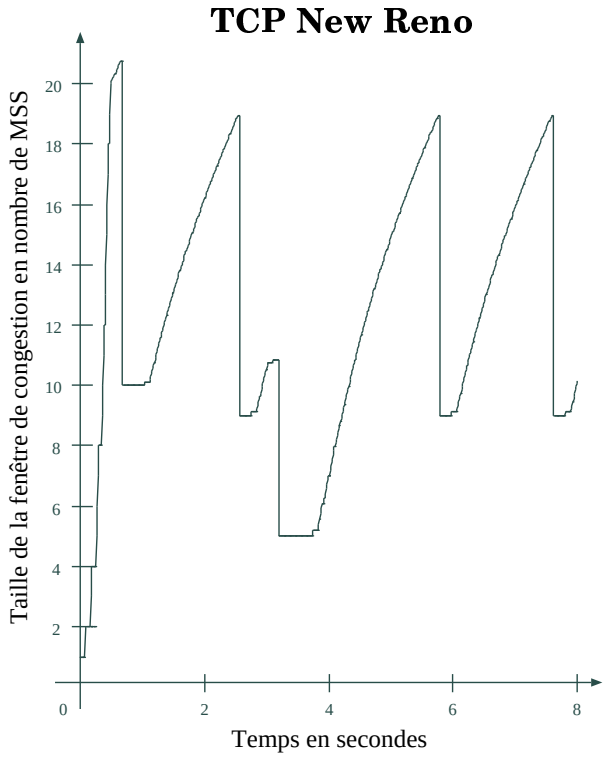

FIG. 2.8 – *Évolution de la taille de la fenêtre de congestion dans TCP New Reno.*

2.2.3 TCP *Forward acknowledgment* (Fack)

Principe. Le protocole TCP Fack [73] propose lui aussi d'améliorer le comportement de TCP Reno lors de pertes multiples durant un même RTT. Pour cela, il estime la quantité de données en transit grâce aux informations fournies par l'option Sack (*cf.* paragraphe 2.2.1). La quantité de données en transit permet à l'émetteur en phase de *retransmission rapide* de gérer plus précisément la taille de la fenêtre de congestion et donc l'émission de nouveaux segments. Lorsqu'une perte est détectée, TCP Fack réduit progressivement la taille de la fenêtre de congestion afin de lisser le débit d'émission.

Mise en œuvre. Cinq nouvelles variables sont introduites par TCP Fack. Lorsque l'émetteur ne se trouve pas en phase de *reprise rapide*, les variables *snd_una* et *snd_fack* sont mises à jour avec le numéro de séquence contenu dans le dernier acquittement reçu. En revanche, durant les phases de *reprise rapide*, la variable *snd_una* est toujours mise à jour grâce au numéro de séquence contenu dans les acquittements, mais la variable *snd_fack* est mise à jour avec le plus grand numéro de séquence acquitté dans les blocs Sack. Quelle que soit la phase dans laquelle se trouve l'émetteur, la variable *snd_nxt* représente toujours le numéro de séquence du prochain nouveau segment (*i.e.*, qui n'a pas déjà été émis) à émettre. Ces trois variables permettent de spécifier que :

- tous les octets de numéro de séquence inférieur à *snd_una* ont été correctement reçus ;
- certains octets de numéro de séquence compris entre *snd_una* et *snd_fack* ont été perdus et doivent être ré-émis (ces octets sont connus grâce aux acquittements Sack) ;

– les octets de numéro de séquence compris entre *snd_fack* et *snd_nxt* ont été émis mais ne sont pas encore acquittés.

La variable *retran_data*, qui représente la quantité de données en cours de retransmission, est également maintenue à jour par TCP Fack. Grâce à ces quatre variables, TCP Fack estime la quantité de données en transit et stocke cette valeur dans la variable *awnd*. Cette variable représente la quantité de données envoyées mais non encore acquittées :

$$awnd = snd\_next - snd\_fack + retran\_data \tag{2.4}$$

Si le récepteur indique que sa file de réception (dans laquelle sont stockés les segments arrivés avant les autres) contient plus de trois MSS octets, alors l'émetteur entre en phase de *reprise rapide*. Cette méthode permet de détecter plus rapidement les pertes lorsque la technique des acquittements retardés est utilisée (avec les acquittement retardés, trois acquittements peuvent représenter la réception de 6 segments, *cf.* section 2.1). L'émetteur entre également en phase de *reprise rapide* si $snd\_fack - snd\_una$ est supérieur à trois MSS. Cette méthode permet d'améliorer la détection de pertes multiples et consécutives. En effet, si quatre segments sont envoyés et que les trois premiers sont perdus, la perte ne pourra pas être détectée par la réception de trois acquittements dupliqués. En revanche, grâce à l'information (plus haut numéro de séquence reçu) contenue dans le seul acquittement envoyé, la perte sera immédiatement détectée. À l'entrée en phase de *reprise rapide*, la valeur de la variable *snd_next* est récupérée. Lorsque la variable *snd_una* aura rattrapé cette valeur (comme avec la variable *recover* de TCP New Reno), TCP Fack sortira de la phase de *reprise rapide* et passera en phase d'*évitement de congestion*. À la détection d'une perte, après avoir été divisée par deux, la taille de la fenêtre de congestion (*cwnd*) de TCP Fack reste constante durant toute la phase de *reprise rapide* (*i.e.*, pas d'augmentation d'un MSS par acquittement dupliqué). Mais TCP Fack émet des segments tant que la quantité de données en transit (estimée grâce à *awnd*) reste inférieure à *cwnd*. La quantité de données qui peut être émise durant la phase de *reprise rapide* est donc estimée plus précisément que dans TCP Reno. Cette amélioration contribue à un meilleur lissage du débit d'émission.

Le protocole TCP Fack améliore également l'algorithme de *démarrage lent*. Dans TCP Reno, durant cette phase, lorsque la perte d'un segment est détectée par la réception de trois acquittements dupliqués, la fenêtre de congestion est réduite de moitié. Or, sachant qu'en phase de *démarrage lent* la taille de la fenêtre de congestion double tous les RTTs, après réduction elle revient à la taille qu'elle avait un RTT plus tôt, c'est-à-dire qu'elle revient à la valeur qui a entraîné la perte de segments. Une nouvelle perte sera donc constatée un RTT plus tard, et la fenêtre sera de nouveau réduite de moitié. Pour éviter cela, dans le cas où une première réduction ne ramène pas la taille de la fenêtre de congestion en dessous du seuil *ssthresh*, TCP Fack réduit une seconde fois la taille de la fenêtre de congestion (ce qui revient à une division par quatre).

2.3 Améliorer la gestion de la fenêtre de congestion

2.3.1 Propositions de correctifs pour TCP Reno

De nombreux travaux ont proposé des corrections pour TCP Reno, afin d'en améliorer le contrôle de congestion. Nous résumons ici les plus significatifs.

Dans la RFC 793, il est suggéré d'initialiser la taille de la fenêtre de congestion à un segment (MSS). Mais l'utilisation des acquittements retardés (attente de la réception d'un second segment) peut conduire à l'expiration du chien de garde de retransmission. L'utilisation d'une fenêtre de congestion initale plus grande permettrait d'éviter cette situation. De plus, une augmentation de la taille de la fenêtre de congestion initiale permettrait également aux connexions ayant une petite quantité de donnée à transmettre de réduire significativement leur temps de transmission. À l'inverse, une fenêtre initiale trop grande risquerait d'entraîner l'engorgement du réseau en phase de *démarrage lent*. Les RFC 2414 [4] (expérimentale) et RFC 3390 [5] (en cours de standardisation) proposent de porter la taille initale de la fenêtre de congestion à 4 ko, soit approximativement à 4 segments sur les réseaux classiques.

La RFC 2861 [45] (expérimentale) propose d'ajuster la taille de la fenêtre de congestion (*cwnd*) après une longue période d'inactivité (*i.e.*, supérieure à un RTT, et durant laquelle aucun segment n'est émis). Cette technique permet d'éviter que *cwnd* n'augmente alors que l'émetteur n'utilise pas la totalité de l'espace disponible dans la fenêtre de congestion. En effet, si *cwnd* continue de croître alors que l'émetteur n'utilise qu'une petite partie de l'espace dont il dispose, la taille de la fenêtre peut être surdimensionnée par rapport aux capacités du réseau. Dans une telle situation, si l'émetteur avait subitement une grande quantité de données à transmettre, il saturerait le réseau en utilisant la totalité de la fenêtre de congestion qui serait surdimensionnée. Pour éviter cela, lorsqu'une connexion reste inactive, la taille de la fenêtre de congestion est réduite de moitié pour chaque RTT d'inactivité. Dans le cas où l'émetteur envoie régulièrement des données mais en quantité inférieure à ce que permet la fenêtre de congestion, il est proposé de réduire *cwnd* en tenant compte de la quantité de données émises :

$$cwnd = \frac{cwnd + \text{quantité de données émises durant le dernier RTT}}{2} \tag{2.5}$$

Avant de réduire la taille de la fenêtre de congestion, le seuil de *démarrage lent* (*ssthresh*) est re-calculé en tenant compte de la valeur de *cwnd* :

$$ssthresh = \max(ssthresh, \frac{3}{4} \times cwnd) \tag{2.6}$$

De plus, il est spécifié que l'accroissement de *cwnd* à l'arrivée d'un acquittement ne doit être fait que si la fenêtre de congestion est pleinement utilisée (*i.e.*, s'il y a suffisamment de données à émettre).

Dans [3], il est proposé de modifier la politique d'augmentation de la fenêtre de congestion : au lieu de l'augmenter d'une quantité d'octets fixe à chaque arrivée d'acquittement, elle augmente en fonction du nombre de nouveaux octets acquittés. Durant les phases de *démarrage lent*, à chaque réception d'un acquittement, la taille de la fenêtre de congestion augmente d'autant d'octets qu'il y en a eu d'acquittés, sans dépasser une valeur limite de deux MSS. Cette technique permet d'accélérer la phase de *démarrage lent*, notamment lorsque la technique des acquittements retardés est utilisée. Si la phase de *démarrage lent* est due à une expiration du chien de garde de retransmission, la valeur limite est fixée à MSS, car dans ce cas, une grande quantité de données peut être aqcuittée en une fois. Notons qu'il est préconisé d'utiliser cette politique d'accroissement en parallèle de celle gérant les périodes d'inactivité [45].

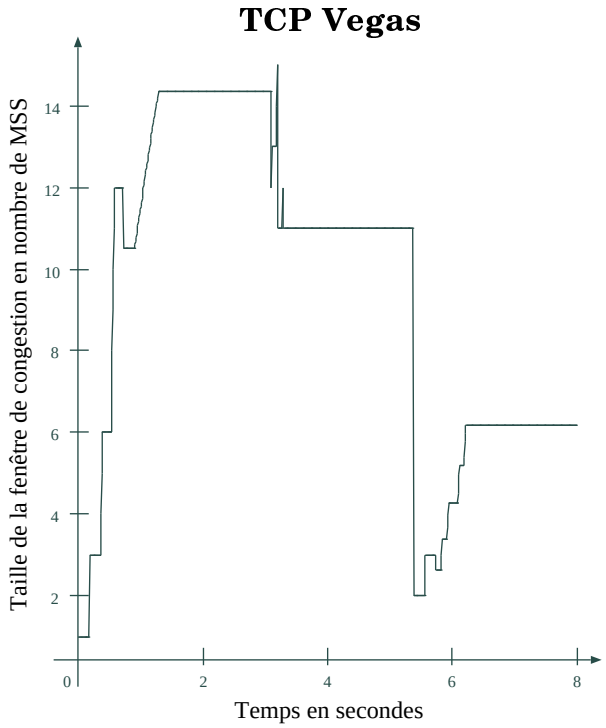

FIG. 2.9 – *Évolution de la taille de la fenêtre de congestion dans TCP Vegas.*

2.3.2 TCP Vegas

Principe. Afin d'améliorer l'utilisation de la bande passante, les recherches sur l'évitement des congestions se sont orientées vers des systèmes préventifs et non plus correctifs comme l'étaient TCP Reno et ses variantes. L'objectif de TCP Vegas [19] est de corriger plusieurs problèmes observés avec TCP Reno. De nouveaux algorithmes de *démarrage lent*, de *reprise rapide* et d'*évitement de congestion* ont été développés dans cette version de TCP. Notons également l'apparition d'un mécanisme qui permet à TCP Vegas d'avoir un débit d'émission plus lissé en supprimant les rafales d'émission. L'estimation des capacités du réseau qu'effectue TCP Vegas permet à la source d'avoir un débit plus stable et lui évite ainsi de créer des congestions. L'évolution de la fenêtre *cwnd* de TCP Vegas est illustrée dans la figure 2.9.

Mise en œuvre. Alors que TCP Reno a besoin de créer une congestion (*i.e.*, perte de segments) pour réussir à estimer la bande passante, TCP Vegas observe la variation du débit. Pour cela, il compare le débit effectif au débit attendu, en utilisant un RTT de référence, RTT_{ref} (plus petit RTT mesuré). Le débit attendu est calculé grâce à la taille de la fenêtre de congestion, censée être émise en RTT_{ref} secondes :

$$\text{débit attendu} = \frac{cwnd}{RTT_{ref}} \tag{2.7}$$

Le débit effectif est calculé grâce au nombre n d'octets envoyés entre la date d'émission d'un segment et la date de réception de son acquittement, RTT secondes plus tard :

$$\text{débit effectif} = \frac{n}{RTT_{inst}} \tag{2.8}$$

En comparant le débit effectif au débit attendu, TCP Vegas ajuste la taille de la fenêtre de congestion. Notons que le débit effectif doit toujours être inférieur au débit attendu. Si ce n'était pas le cas, cela signifierait qu'il faudrait mettre à jour le RTT de référence (RTT_{ref}) avec la valeur courante du RTT (RTT_{inst}). Pour effectuer ce calcul, TCP Vegas a besoin d'un RTT plus précis (de l'ordre du millième de seconde) que celui de TCP Reno, dont la granularité est généralement supérieure à 100 ms.

L'algorithme d'*évitement de congestion* de TCP Vegas cherche à maintenir la taille de la fenêtre de congestion de manière à ce que le débit attendu soit légèrement supérieur à la bande passante disponible. Ce léger dépassement des capacités du réseau permet à TCP Vegas de s'apercevoir d'une éventuelle augmentation de bande passante disponible. En effet, si *cwnd* correspondait exactement aux ressources disponibles du réseau, lors d'une augmentation de bande passante disponible (*i.e.*, des ressources disponibles du réseau), le débit effectif de l'émetteur resterait égal à son débit attendu et donc le surcroît de bande passante utilisable ne serait pas détecté. En revanche, si *cwnd* était légèrement supérieur aux capacités (*i.e.*, débit attendu $>$ débit effectif) du réseau, lors d'une libération de bande passante, l'émetteur pourrait constater une augmentation de son débit effectif et donc accroître la taille de sa fenêtre de congestion *cwnd*. Pour maintenir la taille de la fenêtre de congestion légèrement supérieure à la taille idéale sans congestionner le réseau, TCP Vegas utilise deux bornes entre lesquelles *cwnd* doit se trouver.

Pour éviter les pertes importantes qu'occasionne le *démarrage lent* utilisé par TCP Reno, Vegas utilise la comparaison du débit attendu et du débit effectif. Mais pour que cette comparaison soit possible, il est nécessaire que la fenêtre de congestion reste stable pendant au moins un RTT. Donc, au cours du *démarrage lent*, TCP Vegas augmente *cwnd* exponentiellement mais seulement une fois par RTT (et non à chaque réception d'acquittement comme le fait TCP Reno). Lorsque TCP Vegas détecte que le débit attendu est bien supérieur au débit actuel, il passe dans une phase d'*évitement de congestion*.

Dans TCP Vegas, *cwnd* n'est réduit qu'une seule fois par RTT : la fenêtre de congestion est réduite seulement si le segment perdu a été émis après la dernière réduction de la fenêtre. Ceci permet de diminuer la fenêtre de congestion uniquement si la perte est due au débit actuel et non à un débit qui a déjà été corrigé antérieurement.

De manière générale, dans les autres versions de TCP, tous les segments d'une même fenêtre de congestion sont émis en un temps très inférieur à un RTT (émission en rafales, puis attente des acquittements). La suppression de ces rafales permet à TCP Vegas de lisser le débit d'émission. L'émetteur détermine un délai inter-segments durant lequel il s'autorise à envoyer au maximum deux MSS octets. Notons que ce mécanisme anti-rafales est désactivé durant les phases de *démarrage lent*, car les rafales sont nécessaires pour maintenir une croissance exponentielle de la taille de la fenêtre de congestion.

Enfin, TCP Vegas améliore la gestion des délais de retransmission. Dans TCP Reno, le chien de garde de retransmission n'expire que si aucun acquittement n'est parvenu à l'émetteur depuis RTO secondes. En plus de ce système, TCP Vegas vérifie, pour chaque segment émis mais non acquitté, que son temps de transit n'a pas dépassé le délai de retransmission. Cette modification a pour but de détecter plus rapidement les pertes de segments, ce qui contribue là encore à limiter les variations du débit d'émission.

2.3.3 TCP Westwood

Principe. Dans TCP Westwood [42], la taille de la fenêtre de congestion est calculée en fonction de l'estimation de la bande passante disponible. Seule la politique de réduction de la fenêtre de congestion est modifiée. Dans TCP Reno, une perte de segment (qu'elle soit ponctuelle ou répétée) se solde par une réduction massive de la taille de la fenêtre de congestion. Le protocole TCP Westwood propose de réduire la fenêtre de congestion de manière additive (et non plus multiplicative). Cela permet de sanctionner moins lourdement les dégradations passagères, causées par un lien perturbé ou une congestion ponctuelle (type « *burst* » UDP très court).

Mise en œuvre. Sachant que TCP essaie de déterminer la capacité du réseau en faisant croître le débit jusqu'à provoquer une congestion, on peut en déduire que le débit de la connexion est proche de la bande passante utilisable peu de temps avant l'apparition de la congestion. Pour mesurer la bande passante utilisable ($\hat{b}_k$), TCP Westwood commence par mesurer le débit instantané de la connexion en divisant la quantité de données nouvellement acquittées par le temps séparant les deux derniers acquittements. Notons d_k la quantité de données acquittées dans le dernier acquittement, et Δ_k le temps séparant l'arrivée des acquittements k et $k-1$. La bande passante instantanée b_k se calcule de la manière suivante :

$$b_k = \frac{d_k}{\Delta_k} \tag{2.9}$$

Une valeur lissée de la bande passante utilisable est ensuite calculée de la manière suivante :

$$\hat{b}_k = \alpha_k \times \hat{b}_k + (1-\alpha_k) \times \frac{b_k + b_{k-1}}{2} \qquad \text{avec} \qquad \alpha_k = \frac{2\tau - \Delta_k}{2\tau + \Delta_k} \tag{2.10}$$

Notons que le coefficient α_k, dit de lissage, dépend de la constante τ et du délai inter-acquittements Δ_k. Plus le délai inter-acquittements Δ_k est important (*i.e.*, moins le débit de réception est élevé), plus les deux dernières mesures de la bande passante auront du poids dans l'estimation de la bande passante lissée $\hat{b}_k$.

La politique de croissance du débit de TCP Reno (*i.e.*, croissance exponentielle en phase de *démarrage lent* et linéaire en phase d'*évitement de congestion*) n'est pas modifiée dans TCP Westwood. En revanche, la politique de décroissance de TCP Westwood diffère de celle de TCP Reno. En effet, la taille de la fenêtre de congestion et le seuil de démarrage lent sont ajustés en fonction de la bande passante disponible. Ces ajustements dépendent, comme dans TCP Reno, de la façon dont la congestion est détectée.

Si trois acquittements dupliqués sont reçus, cela signifie que le débit de la connexion a légèrement dépassé la bande passante disponible. Dans ce cas, *ssthresh* et *cwnd* prennent tous deux la valeur :

$$\textit{ssthresh} = \textit{cwnd} = \max\left(2 \times \mathrm{MSS}, \hat{b}_k \times \frac{\textit{RTTmin}}{\textit{taille\_seg}}\right) \tag{2.11}$$

Dans cette équation, *RTTmin* représente la plus petite valeur du RTT, et *taille_seg* la taille moyenne des segments envoyés. Une fois cette mise à jour effectuée, l'émetteur continue d'émettre en phase d'*évitement de congestion*.

Si la perte est détectée par l'expiration du chien de garde de retransmission (*i.e.*, plusieurs segments ont été perdus), cela signifie que $\hat{b}_k$ a dû être surestimée. Dans ce cas, la fenêtre de congestion est réduite à un MSS et le calcul de *ssthresh* reste le même que précédemment. Après avoir réduit *ssthresh* et *cwnd*, l'émetteur entre en phase de *démarrage lent*.

2.3.4 TCP *Learner* (TCP-L)

Principe. TCP-L [29] est une amélioration apportée à TCP qui propose de ne pas augmenter le débit d'émission lorsque l'état du réseau est connu comme étant propice aux congestions. Pour cela, TCP-L « apprend » au cours de la connexion les relations entre le délai de transmission (entre la source et le récepteur), la taille de la fenêtre de congestion, et l'apparition de congestions. Après cette phase d'apprentissage, TCP-L autorisera l'augmentation de la taille de la fenêtre de congestion uniquement si dans les mêmes conditions, cette augmentation n'avait précédemment pas créé de congestion. Cette amélioration permet de stabiliser le débit d'émission. Elle offre ainsi une plus large gamme d'utilisation de TCP (*i.e.*, transport de flux audios, vidéos, *etc.*).

Mise en oeuvre. L'amélioration apportée par TCP-L repose sur :

- l'ajout d'un historique répertoriant les situations (taille de la fenêtre de congestion et état du réseau) pour lesquelles des congestions sont apparues ;
- la possibilité pour TCP de ne pas augmenter la taille de sa fenêtre de congestion lorsqu'il reçoit un nouvel acquittement.

Grâce à ces nouvelles fonctionnalités, lorsque l'émetteur reçoit un nouvel acquittement, il peut choisir de ne pas augmenter la taille de sa fenêtre de congestion. En effet, si l'état dans lequel l'émetteur se trouvera après l'augmentation de la taille de la fenêtre de congestion a déjà été répertorié comme instigateur de congestion, la fenêtre restera inchangée. Sinon la taille de la fenêtre augmentera comme dans les versions classiques de TCP.

Pour connaître l'état de congestion du réseau, TCP-L utilise la taille des files d'attente traversées par les paquets allant de la source à la destination. Cette taille est caractérisée par le temps que passent les paquets dans les files d'attente. Afin de réduire le nombre d'états (pour lesquels une augmentation de la taille de fenêtre de congestion est déconseillée) à mémoriser, TPC-L utilise une borne supérieure pour le temps passé dans les files d'attente. En effet, s'il a été mémorisé que dans l'état $\{W, Q_d\}$ (où W représente la taille de la fenêtre et Q_d le délai d'attente dans les files) le réseau était congestionné, on peut en déduire que pour n'importe quel état $\{W, Q'_d\}$ où $Q'_d > Q_d$, le réseau risque d'être à nouveau congestionné. Il apparaît donc suffisant de conserver une seule valeur du délai d'attente par taille de fenêtre testée. Lorsque les paquets de la connexion changent de chemin pour aller de la source à la destination, TCP-L détecte le changement de route et adapte en conséquence l'historique des états donnant lieu à des congestions.

L'équité entre les flots TCP-L et TCP classiques est conservée, bien que les connexions TCP-L soient légèrement désavantagées (on observe une différence marginale entre les débits de réception des connexions TCP et TCP-L). Notons que l'amélioration de la gestion de la fenêtre de congestion apportée par TCP-L peut facilement être adaptée à n'importe quelle version de TCP (Tahoe, Reno, New Reno, *etc.*).

2.4 Contrôle de congestion basé sur le débit

Avec la croissance des lignes haut débit, les applications utilisant des flots de données temps réel telles que les vidéos, ou les flux audio, se sont rapidement répendues sur Internet. Malheureusement, ces applications supportent mal les ajustements du débit qu'effectue TCP. De ce

fait, elles utilisent le protocole UDP qui n'effectue aucun contrôle des congestions (et donc aucun ajustement du débit). De nouveaux protocoles dont le contrôle de congestion est adapté au transport de flots de données temps réel ont donc vu le jour. Ces nouveaux protocoles, dont le contrôle de congestion n'est pas basé sur une fenêtre d'émission, sont capables de cohabiter équitablement avec TCP (*TCP Friendly*).

2.4.1 *TCP Friendly Rate Control* (TFRC)

Principe. Le protocole TFRC [89, 37, 44] est destiné à être utilisé pour le transport de flots de données qui ne supportent pas les grandes variations du débit de réception (*e.g.*, données multimédia). Ce protocole (en cours de standardisation RFC 3448) tente de résorber les congestions en utilisant le taux de pertes p, le débit de réception D_{rcvd} et le débit TCP-*friendly* D_{friend}, qui est obtenu *via* une modélisation Markovienne de la phase d'*évitement de congestion* de TCP Reno. Le récepteur estime son débit de réception D_{rcvd} ainsi que le taux de pertes de segment p, qu'il transmet à l'émetteur. Initialement, $p = 0$ et l'émetteur TFRC se place en phase de *démarrage lent*. Puis, dès que le taux de pertes devient non nul, il passe en phase d'*évitement de congestion*.

Durant la phase de *démarrage lent*, le débit d'émission D est calculé grâce au débit de réception D_{rcvd}. Alors qu'en phase d'*évitement de congestion*, le débit D est ajusté en fonction du débit TCP-*friendly* D_{friend} (*cf.* équation 2.14).

Mise en œuvre. Contrairement à TCP Reno, qui répond à toute perte de segment par une réduction massive de la fenêtre de congestion, TFRC ajuste son débit en fonction du taux de pertes p. Ce taux doit refléter l'état du réseau, c'est-à-dire le nombre d'événements de pertes ainsi que leur importance. Un événement de pertes représente la perte d'un ou de plusieurs segments consécutifs. Le taux de pertes doit être peu sensible à une perte unique (afin de lisser le débit), tout en variant significativement lorsque plusieurs événements de pertes se succèdent (ou lorsqu'un événement de pertes est important). Pour cela, le taux p est défini comme étant l'inverse de la moyenne pondérée des huit derniers intervalles de pertes (utilisation de l'algorithme WALI[16]). L'intervalle de pertes représente la quantité de données correctement reçues entre deux événements de pertes. Ainsi, plus les intervalles de pertes sont importants (*i.e.*, la quantité de données reçues entre deux événements de pertes est importante), plus le taux de pertes sera faible. Le récepteur TFRC calcule également son débit de réception D_{rcvd} sur une période d'un RTT, et l'envoie à l'émetteur.

En phase de *démarrage lent*, l'émetteur utilise le débit de réception pour éviter de saturer le lien congestionné. Dans cette phase, au lieu de doubler le débit d'émission à chaque RTT (comme le fait TCP Reno), le nouveau débit d'émission est calculé de la manière suivante :

$$D = 2 \times \min(D, D_{rcvd}) \tag{2.12}$$

Notons que chaque segment émis (sauf le premier) contient la valeur courante du RTT, ce qui permet au récepteur de réguler l'émission des acquittements (contenant les valeurs de p et D_{rcvd}). En effet, le récepteur n'émettra un acquittement qu'à l'expiration d'un chien de garde de réponse, armé avec le RTT reçu.

16. *Weighted Average of Loss Interval*.

Lorsque le taux de pertes reçu est non nul, l'émetteur passe en phase d'*évitement de congestion*. Dans cette phase, le débit D est ajusté en fonction du débit TCP-*friendly* D_{friend} (*cf.* équation 2.14) :

$$\begin{array}{l} \text{Si} \quad D < D_{\text{friend}} \quad \text{alors,} \\ \qquad D = \min\left(D + \frac{1}{\text{RTT}}\right) \\ \text{Sinon} \\ \qquad D = D_{\text{friend}} \end{array} \tag{2.13}$$

De cette façon, l'émetteur TFRC ne peut pas être plus agressif qu'un émetteur TCP confronté à la même congestion : il est « *friendly* » avec TCP.

Le débit D_{friend} est calculé *via* l'équation suivante, provenant de la modélisation Markovienne de la phase d'*évitement de congestion* de TCP Reno :

$$D_{friend} \approx \frac{\text{taille d'un segment TCP}}{\sqrt{\frac{2bp}{3}} \times \text{RTT}_{\text{inst}} + \left(3\sqrt{\frac{3bp}{8}} \times p \times (1 + 32p^2)\right) \times \text{RTO}} \tag{2.14}$$

Dans cette équation, RTT_{inst} est le RTT mesuré, p est le taux de pertes, RTO est la valeur du chien de garde de retransmission, et b le nombre de segments acquittés par acquittement TCP.

2.4.2 *Rate Adaptation Protocol* (RAP)

Principe. Le protocole RAP [101] est lui aussi destiné au transport des flots de données temps réel (flux audio, vidéo, *etc.*). Comme TCP, RAP utilise les pertes de segments comme indicateurs de congestion. Mais la détection des pertes qu'effectue RAP lui est propre. En absence de perte, le protocole RAP augmente périodiquement et d'une quantité constante le débit d'émission. Il est divisé par deux en cas de perte. Ce débit est ensuite appliqué à la source en jouant sur le délai inter-paquets.

Mise en œuvre. Chaque segment envoyé par l'émetteur RAP contient un numéro de séquence. Pour chaque segment reçu, le récepteur envoie un acquittement (*i.e.*, pas d'acquittements retardés) qui contient le numéro de séquence du dernier segment reçu, celui du dernier segment non reçu, ainsi que le numéro de séquence du dernier segment reçu avant le segment perdu. Cette redondance d'informations apporte de la robustesse face aux pertes ponctuelles d'acquittements. Pour détecter les pertes de segments (*i.e.*, les congestions), RAP utilise un chien de garde de retransmission, ainsi que les trous dans l'enchaînement des numéros de séquence. Pour chaque segment émis, la source enregistre son numéro de séquence, son heure de départ et son débit d'émission. Avant chaque émission d'un nouveau segment, la source vérifie si le chien de garde de retransmission a expiré pour l'un des segments déjà émis. La détection de pertes grâce aux trous dans l'enchaînement des numéros de séquence est similaire à la détection *via* la réception de trois acquittements dupliqués de TCP Reno. Si l'émetteur reçoit un acquittement pour un segment envoyé trois segments après un segment non encore acquitté, ce dernier est considéré perdu.

Afin d'éviter que le débit ne soit réduit plusieurs fois au cours du même RTT (*i.e.*, perte de segments ayant été émis au même débit), RAP utilise un mécanisme de détection de pertes groupées. Notons $\text{Seq}_{\text{first}}$ le numéro de séquence d'un segment dont la perte a été détectée, et Seq_{last} le numéro de séquence du dernier segment émis au moment de cette détection. Puisque

l'émetteur réagira à la perte du segment $\text{Seq}_{\text{first}}$, toute perte de segments de numéro Seq vérifiant $\text{Seq}_{\text{first}} < \text{Seq} \leq \text{Seq}_{\text{last}}$ sera ignorée en ce qui concerne l'ajustement du débit. Lorsque l'acquittement d'un segment ayant un numéro de séquence supérieur à Seq_{last} sera reçu (*i.e.*, un RTT plus tard), l'émetteur recommencera à prendre en compte les pertes.

En l'absence de pertes, le débit d'émission $\textit{débit}_i$ augmente régulièrement d'une quantité constante α :

$$\textit{débit}_{i+1} = \textit{débit}_i + \alpha \qquad \text{avec} \qquad \alpha = \frac{\text{taille d'un segment}}{C} \tag{2.15}$$

Dans ces équations, C est une constante permettant de jouer sur le pas d'incrémentation du débit. En cas de perte, le débit est réduit de moitié :

$$\textit{débit}_{i+1} = \frac{\textit{débit}_i}{2} \tag{2.16}$$

Les ajustements du débit ne doivent pas être effectués trop souvent sous peine de créer des oscillations dans le débit d'émission. Inversement, des changements trop peu fréquents entraîneraient un temps de réponse trop élevé. Si aucune congestion n'est détectée, RAP ajustera le débit d'émission une fois par RTT. En revanche, si une perte est détectée, le débit sera immédiatement réduit. Le fait d'ajuster le débit d'émission une fois par RTT permet à la source d'observer les réactions du réseau face au nouveau débit. La fréquence de l'augmentation du débit étant basée sur le RTT, plus ce dernier est court, plus la connexion RAP sera agressive envers les flots ayant des RTTs plus longs. Notons que si le débit est mis à jour tous les RTT, et si la constante C est égale à un RTT, l'augmentation du débit d'émission sera d'un segment par RTT, ce qui correspond à la phase d'*évitement de congestion* de TCP. Pour appliquer le débit d'émission $\textit{débit}_i$, la source émet un paquet toutes les $\textit{IPG}$[17] secondes, où $\textit{IPG}$ est déterminé comme suit :

$$\textit{IPG}_{i+1} = \frac{\text{taille d'un segment}}{\textit{débit}_{i+1}}$$

2.4.3 *PAth STatus-based RAte control* (PASTRA)

Principe. Comme TFRC, le protocole PASTRA [113] a été conçu pour transporter des flots de données supportant mal les fortes variations de débit. Ce protocole propose d'ajouter à UDP un mécanisme de contrôle de congestion, en utilisant le concept de « durée relative de trajet aller », le ROTT[18]. Le ROTT est utilisé pour évaluer l'état du chemin emprunté par les segments. L'état du chemin permet ensuite à l'émetteur de choisir le mode de transmission adéquat, c'est-à-dire d'effectuer le contrôle de congestion approprié. Notons que pour calculer son débit d'émission, la source PASTRA utilise une équation modélisant le débit qu'aurait une source TCP dans les mêmes conditions.

Mise en œuvre. L'état du chemin est évalué en fonction du temps que mettent les segments pour aller de l'émetteur au récepteur (ROTT). Ce délai ne peut pas être mesuré de manière exacte car les horloges de l'émetteur et du récepteur ne sont pas synchronisées. Cette inexactitude ne perturbe pas PASTRA car il utilise la variation du délai d'acheminement et non pas le

17. *Inter Packet Gap*.
18. *Relative One way Trip Time*.

délai lui-même. La variation du délai d'acheminement peut également être faussée par la dérive des horloges respectives de l'émetteur et du récepteur. Notons que cette dérive pourra être calculée et compensée grâce à l'algorithme ESRS [114]. À certains moments, les variations du ROTT sont très brusques et des « pics » sont observables. L'enchaînement de brusques variations du ROTT sur une brève période est appelée « trains de pic » [19]. L'étude de la corrélation entre les « trains de pics », les pertes de segments et la réduction du débit a montré que les pertes de données se produisent essentiellement durant les « trains de pics », et ce, quel que soit le débit d'émission de la source. La détection des « trains de pics » permettrait donc de connaître l'état de congestion du réseau.

Pour ajuster son débit, l'émetteur PASTRA a besoin de connaître l'état du chemin emprunté par le flot de données qu'il envoie. À chaque fin de stage [20], le récepteur vérifie l'état du chemin emprunté, et envoie un rapport à l'émetteur. Quatre *états* sont utilisés (*cf.* figure 2.10) pour qualifier l'état du chemin :

- Dans le premier, « Init », le récepteur calcule les paramètres nécessaires à l'estimation du ROTT [114] (*i.e.*, l'état du chemin n'est pas encore connu).
- Dans le second, « Stable », le chemin est considéré comme stable tant que le taux de segments en transit durant les « trains de pics » est inférieur à 30%.
- Le troisième état, « Instable », est atteint si le taux de segments en transit durant les « trains de pics » dépasse 30% ou si PASTRA est incapable d'estimer le ROTT (dans l'état « Init »).
- Le dernier état, « Changé », est utilisé lorsque le chemin entre la source et la destination change (*i.e.*, re-routage). Dans ce cas, le récepteur doit calculer le nouveau ROTT de référence.

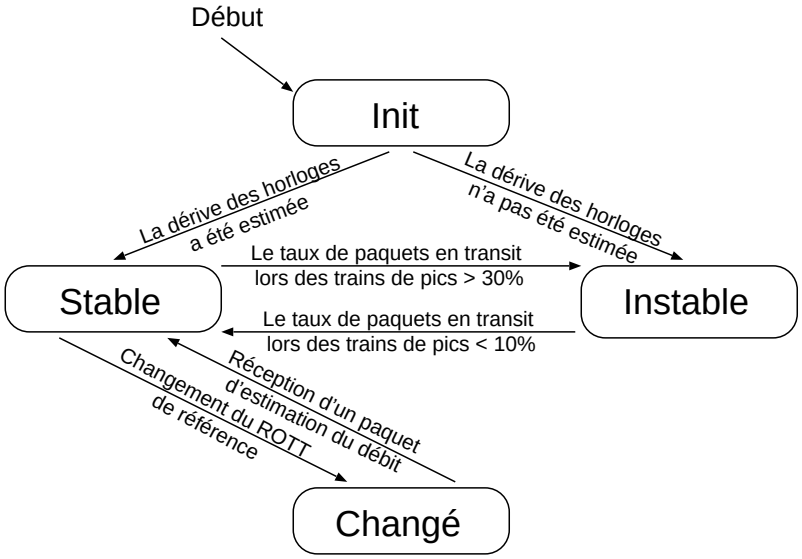

FIG. 2.10 – *Transitions des états du chemin*

19. *spikes trains*.

20. Un stage représente une période de temps durant laquelle on ne change pas la valeur du débit (un stage dure 5 s).

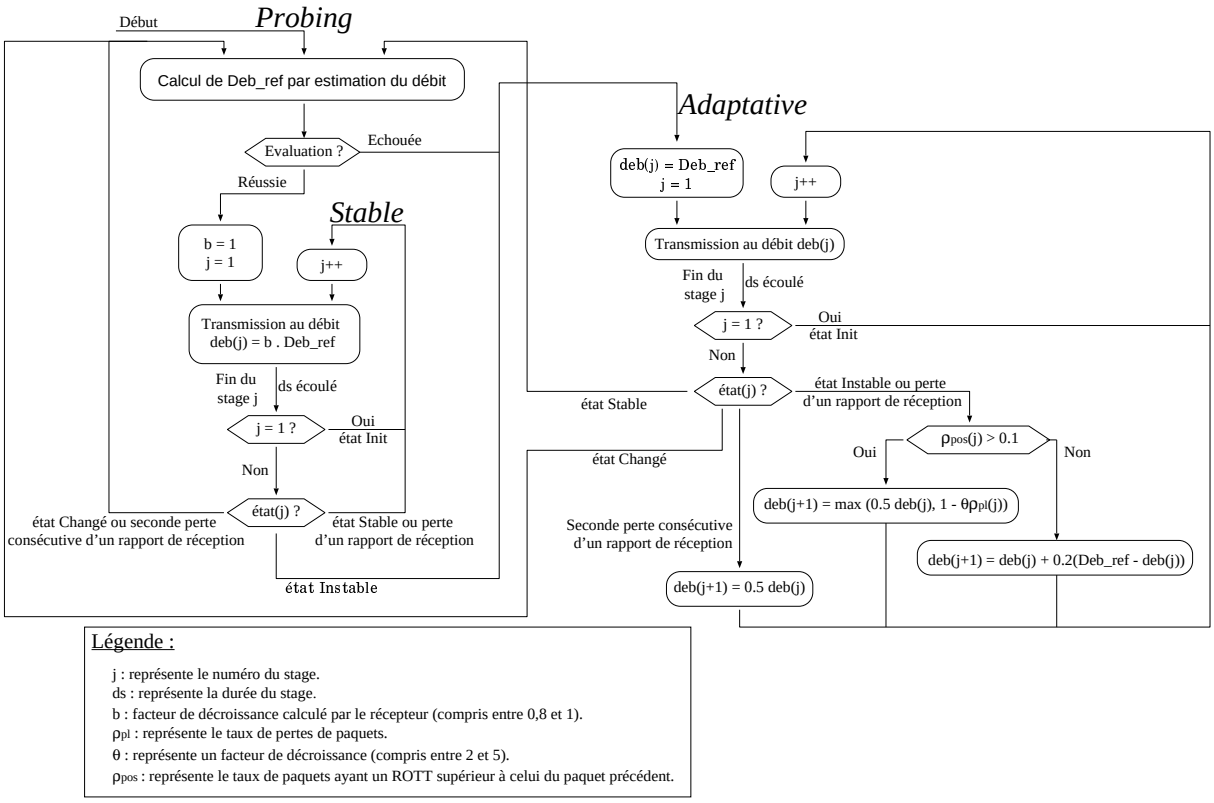

FIG. 2.11 – *Comportement d'un émetteur PASTRA.*

Les différents états du chemin sont utilisés pour déterminer le *mode* de transmission à adopter grâce à un automate (*cf.* figure 2.11) :

– Dans le premier mode, « *Probing* », l'émetteur estime le débit qu'aurait une connexion TCP dans les mêmes conditions. Il s'agit de l'initialisation de la connexion (le chemin est dans l'état « Init ») ou d'un re-routage (le chemin est dans l'état « Changé »).

– Si PASTRA est incapable d'estimer ce débit, la source passe dans le second mode, « *Adaptive* », dans lequel les données sont transmises avec un débit assez faible (*i.e.*, il s'agit d'une transmission en mode dégradé).

– Le dernier mode, « *Stable* » est utilisé lorsque l'état du chemin reste à « Stable ». Dans ce cas, les données sont transmises à un débit proche de celui qu'aurait une connexion TCP.

Notons que l'émetteur peut sortir du mode « *Stable* » pour passer en mode « *Adaptive* » si l'état du chemin passe à « Instable ». Pour sortir du mode « *Adaptive* », l'état du chemin doit être qualifié de « Stable » ou « Changé ». Dans les deux cas, l'émetteur passera en mode « *Probing* ». Notons également que la perte successive de deux rapports de réception (qui indiquent l'état du chemin) alors que l'émetteur était en mode « *Stable* » provoquera son retour en mode « *Probing* ».

2.4.4 *Loss Delay Adaptation Algorithm*(LDA+)

Principe

L'algorithme LDA+ [107, 108] repose sur le protocole RTP[21] [106], qui lui fournit des informations sur les pertes de données et le RTT. L'algorithme LDA+ permet d'adapter le débit de la source avec un mécanisme de type AIMD. Lorsque des pertes surviennent, LDA+ détermine la bande passante disponible pour la connexion grâce à l'équation présentée dans TFRC (*cf.* paragraphe 2.4.1), et ajuste le débit en conséquence. Durant les phases sans congestion, le débit augmente de manière additive, douce et lissée.

Mise en œuvre

Le protocole RTP permet à LDA+ de connaître le RTT de la connexion grâce aux messages RTCP[22] qui sont envoyés périodiquement avec un intervalle minimum de 5 s. Les messages RTCP donnent également des informations sur le taux de pertes. À ces informations, LDA+ ajoute, dans un champ réservé des en-têtes RTCP, la mesure de la bande passante b du goulot d'étranglement (*i.e.*, la bande passante disponible).

Lorsqu'aucune congestion est détectée à l'étape k, le débit D_k est incrémenté avec un pas A_k :

$$D_k = D_{k-1} + A_k \tag{2.17}$$

La valeur du pas d'incrémentation A_k doit évoluer doucement (sans brusque variation) de manière à avoir un débit d'émission lissé. L'évolution du pas d'incrémentation doit également permettre de conserver l'équité entre les différents flots circulant sur le réseau. Le pas A_k est donc défini comme étant le minimum des trois pas d'incrémentation suivants : A_k^{add}, A_k^{exp} et A_k^{TCP}.

Le pas d'incrémentation A_k^{add} est calculé après la réception du $k^{\text{ième}}$ rapport de réception, en utilisant le débit D_{k-1}, calculé à l'étape $k-1$, et la dernière valeur mesurée de la bande passante disponible du goulot b_{k-1} :

$$A_k^{add} = \left(2 - \frac{D_{k-1}}{b_{k-1}}\right) \times A_{k-1}^{add} \tag{2.18}$$

Ce pas d'incrémentation autorise la source à augmenter son débit d'émission (au maximum le doubler) uniquement si le débit qu'elle a actuellement (D_{k-1}) est inférieur à deux fois la bande passante disponible (b_{k-1}).

Le second pas d'incrémentation A_k^{exp}, a pour but de limiter l'augmentation du débit :

$$A_k^{exp} = \left(1 - \exp^{-(1-\frac{D_{k-1}}{b_{k-1}})}\right) \times D_{k-1} \tag{2.19}$$

Il converge vers 0 lorsque le débit de la connexion D_k tend vers la valeur de la bande passante du goulot d'étranglement b_k.

Le dernier pas d'incrémentation A_k^{TCP} a pour but de rendre LDA+ équitable avec les flots TCP. Il est calculé en utilisant l'augmentation du débit d'un flot TCP (exprimé par l'augmentation $\Delta cwnd$ de sa fenêtre de congestion) sur une période de Δt secondes :

$$A_k^{TCP} = \frac{\Delta cwnd}{\Delta t} \tag{2.20}$$

21. *Real-time Transport Protocol*.
22. *RTP Control Protocol*.

Lorsqu'une perte de segment est détectée, le débit d'émission est déterminé grâce à l'intervalle de pertes l (défini à la manière de TFRC, *cf.* paragraphe 2.4.1), au débit d'émission actuel D_{k-1} (*i.e.*, lors de la détection de la perte) et au débit D_{friend} (calculé grâce à l'équation de TFRC) qu'aurait une source TCP dans cette situation :

$$D_k = \max\left(\left(1-\sqrt{l}\right) D_{k-1}, D_{friend}\right) \tag{2.21}$$

2.5 Contrôle de congestion avec une information provenant du réseau

La dernière piste permettant d'améliorer le contrôle de congestion au niveau de la couche transport consiste à ajuster le débit d'émission en fonction d'informations retournées par le réseau.

2.5.1 Messages ICMP *source quench*

Le message ICMP (*cf.* annexe A.3) *source quench* [93, 9] est envoyé à l'émetteur par le récepteur ou des équipements intermédiaires lorsque le débit d'émission est trop important par rapport à leurs capacités. Un tel message signale donc la formation d'une congestion. Dès que l'émetteur reçoit ce message, il doit le transmettre à la couche de transport concernée afin qu'elle ajuste le débit d'émission. Lorsque TCP reçoit ce type de message, la fenêtre de congestion est réduite à sa taille initiale [16] (*cf.* paragraphe 2.1.5). Actuellement cette technique est très peu utilisée car elle a tendance à aggraver les congestions en ajoutant au trafic (déjà trop important) des messages de contrôle qui viennent saturer les routeurs. De plus, le protocole ICMP est parfois bloqué par les pare-feux pour des raisons de sécurité. Les message ICMP *source quench* sont donc rarement utilisés.

2.5.2 *Random Early Detection* (RED)

Différentes politiques de gestion de file d'attente de routeurs ont été étudiées (*e.g.*, Drop Tail, PBS, RIO, *etc.*). Parmi celles-ci, la politique RED [15] permet aux routeurs de détruire aléatoirement des paquets avant que leur file d'attente ne soit pleine. En effet, dès que la taille moyenne de la file dépasse un seuil donné (seuli1 sur la figure 2.12), les paquets arrivant sont détruits aléatoirement, avec une probabilité qui augmente avec la taille de la file. Lorsque la taille moyenne de la file dépasse un second seuil (seuli2 sur la figure 2.12), tous les paquets arrivant sont détruits. Cette destruction préventive des paquets permet aux routeurs d'informer les émetteurs qu'une congestion est en train de se former et ce, sans attendre que leur file ne soit saturée. La détection de la perte d'un paquet par les émetteurs entraînera une réduction de leur débit et enrayera ainsi la formation de la congestion.

2.5.3 *Explicit Congestion Notification* (ECN)

Principe. Plutôt que de détruire aléatoirement les paquets, l'option ECN [98, 97] permet de les marquer. Lorsqu'un routeur utilisant l'option ECN et ayant une politique RED s'aperçoit

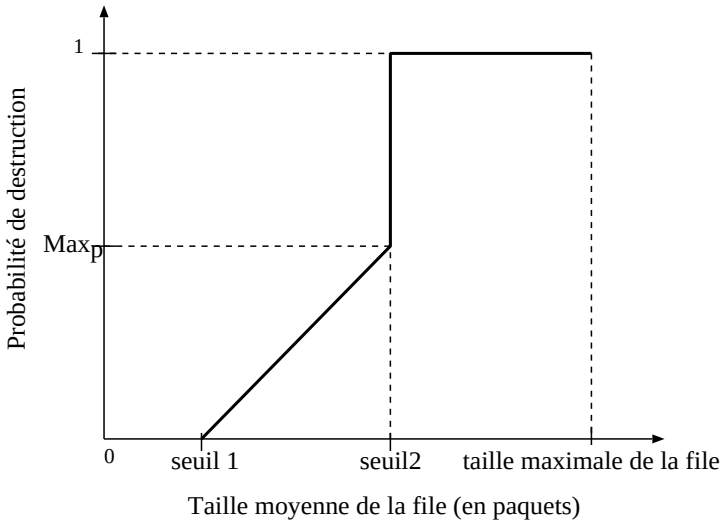

FIG. 2.12 – *Évolution de la probabilité de détruire un paquet dans une file RED en fonction de la taille moyenne de la file d'attente.*

qu'une congestion se forme (*i.e.*, la taille moyenne de la file dépasse le premier seuil), il marque les paquets qui sont *ECN-capable* (label certifiant que l'émetteur comprendra le marquage). Ce marquage, permet d'indiquer explicitement à l'émetteur qu'une congestion est en train de se former, sans avoir à détruire un paquet, et donc sans engendrer de retransmission.

Mise en œuvre. Après négociation entre l'émetteur et le récepteur, si les deux extrémités de la connexion comprennent le marquage ECN, les paquets sont dits *ECN-capable* et labelisés comme tels. En cas de congestion, ils pourront alors être marqués par les routeurs (implémentant une politique ECN), en positionnant le drapeau CE [23] (situé dans l'en-tête) à « vrai » . Lorsque le récepteur TCP reçoit un paquet ayant le drapeau CE à « vrai », il émet un acquittement ayant le drapeau ECE [24] à « vrai ». Afin de rendre l'option ECN robuste face aux pertes d'acquittements, le récepteur émet des acquittements ayant le drapeau ECE à « vrai » jusqu'à ce qu'il reçoive un paquet ayant le drapeau CWR [25] à « vrai ». La réception d'un paquet ayant le drapeau CWR à « vrai » n'assure pas que l'émetteur a bien reçu le message ECE, mais qu'il a réduit son débit d'émission après avoir émis le paquet que le routeur avait marqué.

Lorsqu'un émetteur TCP reçoit un acquittement ayant le drapeau ECE à « vrai », il sait qu'une congestion est en train de se former sur le chemin aller [34, 98]. Cette indication de congestion est traitée comme une perte de segment : la fenêtre de congestion est réduite de moitié, et le seuil de démarrage lent est mis à jour. Le drapeau ECE étant présent dans tous les acquittements jusqu'à ce qu'un segment CWR arrive au récepteur, TCP ne doit pas réagir plus d'une fois par RTT à une série d'indications de congestion. Lorsque l'émetteur TCP réduit la taille de la fenêtre de congestion, quelle qu'en soit la raison, il positionne à « vrai » le drapeau CWR dans le premier paquet de données émis après la réduction. Le paquet contenant le drapeau CWR à « vrai » ne doit jamais être un paquet de retransmission [98].

23. *Congestion Experienced*.
24. *ECN-Echo*.
25. *Congestion Window Reduced*.

2.6 Conclusion

Bilan. Dans ce chapitre, nous avons rappelé le fonctionnement du mécanisme de contrôle de congestion de TCP Reno. Puis, nous avons présenté une synthèse de travaux représentatifs portant sur les nombreuses évolutions et alternatives proposées pour améliorer le contrôle de congestion. Ces travaux ont été classés en quatre catégories : optimisation des acquittements (*cf.* section 2.2), optimisation de la fenêtre de congestion (*cf.* section 2.3), protocoles sans fenêtre (*cf.* section 2.4) et informations sur l'état du réseau (*cf.* section 2.5).

L'étude des différents mécanismes de contrôle de congestion nous permet de tirer quelques conclusions. Pour commencer, il est certain que les nouveaux protocoles de transport devront implémenter un mécanisme de contrôle de congestion [36]. En effet, les progrès attendus aux niveaux des couches réseau et applicative ne pourront que compléter (et non remplacer) le contrôle de congestion de la couche transport.

L'amélioration de la gestion des acquittements est une piste qui a déjà été largement explorée (*cf.* acquittements cumulés, retardés, TCP Sack, TCP Fack, TCP New Reno). Il semble donc peu probable que des avancées significatives soient à attendre de ce côté. Cette piste a permis de réduire la quantité de données inutilement retransmises. En revanche, elle n'a pas apporté d'amélioration en ce qui concerne les réductions de débit qui restent souvent inadaptées à l'état de congestion du réseau.

Le marquage des paquets (ou de leur acquittement) est une piste prometteuse (*cf.* travaux menés sur ECN [32, 43]). Cependant, les techniques liées aux protocoles implémentés sur les équipements intermédiaires (*e.g.*, les routeurs) posent des problèmes au niveau du déploiement. De plus, ce genre de marquage est souvent incompatible avec les mesures de sécurité réseau (*e.g.*, ECN est incompatible avec certains pare-feux).

Le dernier axe d'amélioration possible est donc une meilleure gestion du débit d'émission que ce soit avec ou sans fenêtre d'émission. L'utilisation d'une fenêtre d'émission a l'avantage de réguler automatiquement la quantité de données en transit (*i.e.*, il ne peut y avoir en transit plus de données que n'en contient la fenêtre). En revanche, l'utilisation d'un délai inter-paquets facilite le lissage du débit d'émission (*i.e.*, les paquets ne sont pas envoyés en rafale). Il est cependant possible de lisser le débit d'émission même en utilisant une fenêtre (*cf.* TCP Vegas et TCP-L).

Comme nous l'avons vu dans ce chapitre, les propositions d'alternatives à TCP tentent de lisser le débit d'émission. En effet, les brusques variations du débit qu'impose TCP gênent les flots temps réel (*e.g.*, flux audio, vidéos, *etc.*). De plus, des variations fortes et répétées ont tendance à rendre le réseau instable, ce qui contribue à la sous-utilisation de ses capacités. Ainsi, les nouveaux protocoles, qu'ils utilisent une fenêtre (TCP Vegas TCP Westwood, TCP-L), ou qu'ils soient basés sur le débit (TFRC, RAP, PASTRA et LDA+), tentent tous de lisser au maximum leur débit d'émission.

Des travaux pour adapter le contrôle de congestion aux protocoles *multicast* sont actuellement menés [117, 28]. D'autres travaux cherchent à déplacer le contrôle de congestion du côté du récepteur [96, 102, 46].

Perspectives. Dans cet état de l'art, nous nous sommes concentrés sur un contrôle de congestion *unicast* implémenté par l'émetteur. Dans cette problématique, des améliorations proviendront sans doute encore de l'observation et de l'amélioration des protocoles existants. Mais,

dans le champ étroit des améliorations possibles, c'est certainement l'application de nouveaux modèles mathématiques [83, 55] qui offriront le plus de possibilités.

De plus, des avancées significatives seraient envisageables si l'on faisait abstraction de la compatibilité avec TCP. En effet, TCP Reno impose aux nouveaux protocoles d'avoir un comportement correctif, sous peine de s'approprier toute la bande passante. Cependant, les protocoles préventifs tel que TCP Vegas ont de bien meilleurs résultats [2, 75] que les protocoles correctifs en terme d'équité, de lissage des débits, de taux de bonnes transmissions, *etc.* En se plaçant dans un environnement dédié (réseau local, lien entre téléphone cellulaire et réseau Internet, réseau *ad'hoc etc.*) les protocoles préventifs pourraient être déployés. Notons qu'il semble également possible de rendre temporairement agressif un protocole préventif lorsqu'il est en concurrence avec un protocole correctif [27]. Une telle méthode permettrait donc de déployer des protocoles préventifs sur Internet.

Le développement de protocoles préventifs basés sur des principes issus de la théorie du contrôle des systèmes reste jusqu'à présent une piste peu explorée. Notons que la modélisation continue des phénomènes de congestions sur les réseaux à commutation de paquets apporte une grande simplification du problème. De plus, ce type de modélisation permet, dans certains cas, de ramener le contrôle de congestion à des problèmes largement connus et traités en automatique [26]. L'application au contrôle de congestion des résultats obtenus dans ce domaine paraît être une piste intéressante à suivre. C'est pourquoi nous avons choisi de l'exploiter dans les chapitres suivants.

Pour finir, on peut facilement imaginer que le contrôle de congestion idéal n'est pas universel et qu'il dépend fortement du type de données à transporter. C'est pourquoi, à l'avenir, la technique de contrôle de congestion à employer pourrait être choisie par les applications, comme cela est proposé dans DCCP [64].

Chapitre 3

Cohérence des modélisations continue et discrète d'un réseau à commutation de paquets

Ce chapitre commence à la section 3.1, par la présentation de deux modélisations continues représentatives des études menées actuellement en automatique sur le contrôle de congestion. Ensuite, dans la section 3.2, le modèle expérimental utilisé pour comparer les modélisations continue et discrète est décrit. Puis, la méthode de comparaison des deux modélisations est présentée dans la section 3.3. Dans la section 3.4, les résultats des comparaisons sont présentés et analysés. Enfin, dans la section 3.5, nous soulignons les limites et l'intérêt de l'approximation continue.

Positionnement. Comme nous l'avons vu précédemment, de nombreuses contributions ont œuvré à l'amélioration du contrôle de congestion de TCP, et de nouveaux protocoles mieux adaptés aux flots multimédia ont vu le jour. En parallèle de ces développements, des études tentant de modéliser le transport de flots sur Internet comme des systèmes à entrées-sorties [62] ont été menées. Les entrées sont les flots de données et les sorties leurs acquittements. Les acquittements ne mettant pas un temps fixe pour parvenir à la source, une connexion réseau sera modélisée par un système à entrées-sorties avec un délai incertain. Ainsi, il est possible de définir un bouclage « fictif », qui est réalisé par les acquittements allant de la destination vers la source. L'utilisation de modèles continus permet de réduire la complexité de l'analyse du comportement des réseaux. Ces analyses permettent de dégager de nouveaux résultats qui présentent de gros avantages en termes de stabilité et de robustesse [83, 118, 22].

Les réseaux sont intrinsèquement discrets. Or la plupart des études qui sont menées dans le domaine du continu ne tiennent pas compte de la pertinence des résultats obtenus une fois qu'ils sont discrétisés. La validité de l'approximation continue a donc été étudiée lors de recherches s'appuyant principalement sur des simulations [63, 81, 13, 99]. En effet, il est difficile d'obtenir des comparaisons analytiques de modèles discret et continu. À notre connaissance, le problème de la cohérence entre les réseaux et leurs modélisations continues a été souligné pour la première fois dans [13]. Dans cet article, les auteurs montraient que l'utilisation de l'approximation continue donnait des résultats proches de ceux obtenus en discret dans un grand nombre de scénarios. Les travaux présentés dans ce chapitre étendent les résultats obtenus dans [13, 99].

3.1 Modèles d'automatique

Dans cette section, nous présentons deux modélisations représentatives des recherches actuelles sur l'étude du contrôle de congestion en continu. D'autres modèles sont présentés dans [82]

3.1.1 Modèles d'un réseau à commutation de paquets

Dans [54, 55], Izmailov propose une modélisation déterministe d'une connexion entre une source ayant un mécanisme de contrôle de congestion et un récepteur, reliés par un lien de capacité constante μ. Les deux modèles obtenus par Izmailov sont caractérisés par les deux systèmes d'équations suivants :

$$\begin{cases} \dot{q}(t) = x(t - Df) - \mu \\ \dot{x}(t) = -a\,(q(t - Dr) - q_O) - b\,(q(t - Dr - r) - q_O) \end{cases} \tag{3.1}$$

$$\begin{cases} \dot{q}(t) = x(t - Df) - \mu \\ \dot{x}(t) = -a\,(q(t - Dr) - q_O) - b\,(q(t - Dr - r) - q_O) - c\left(q(t - Dr - \frac{r}{2}) - q_O\right) \end{cases} \tag{3.2}$$

Dans ces équations, q représente la taille de la file d'attente, x le débit de source et q_O la taille optimale de la file d'attente (valeur vers laquelle doit tendre q). Les constantes Df et Dr représentent respectivement le temps de propagation entre la source et la file d'attente, et le temps de propagation entre la file d'attente et la source en passant par le récepteur (*cf.* figure 3.1). Les constantes a et b sont des coefficients d'accroissement et de réduction de débit. Enfin, la variable r est un intervalle de temps de « contrôle ». Cette variable permet de jouer sur la stabilité et la robustesse du système d'équations [22]. Le système d'équations 3.2 est similaire au premier, mais permet d'introduire un degré de liberté supplémentaire pour lequel le paramètre c doit être calculé [23].

En utilisant la nouvelle variable $y(t) = q(t) - q_O$, l'équation modélisant le débit de la source dans le système 3.1 devient une équation du second ordre où $\tau = Df + Dr$:

$$\ddot{y}(t) + ay(t - \tau) + by(t - \tau - r) = 0 \tag{3.3}$$

L'équation modélisant le débit de la source dans le système 3.2 devient

$$\ddot{y}(t) + ay(t - \tau) + by(t - \tau - r) + cy(t - \tau - \frac{r}{2}) = 0 \tag{3.4}$$

Une telle modélisation permet de ramener le contrôle de congestion à un problème de stabilité asymptotique d'un système à retard [22].

3.1.2 Modélisation de TCP

Une modélisation continue du comportement de TCP en phase d'*évitement de congestion* a été proposée par Misra, Gong et Towsley dans [74] et discutée dans [50]. Cette modélisation propose d'envisager le contrôle de congestion comme un problème de contrôle des informations de retour (*i.e.*, les acquittements). Contrairement à la modélisation précédente, le calcul de la

taille de la file d'attente prend en compte plusieurs connexions (N). Le modèle de contrôle de congestion décrit dans [74] est composé d'un système d'équations non linéaire :

$$\begin{cases} \dot{w}_i(t) = \frac{1}{R_i(t)} - \frac{w_i}{2} p(t-\tau) \frac{w_i(t-\tau)}{R_i(t-\tau)} \\ \dot{q}(t) = \begin{cases} -c + \sum_{i=1}^{N} R_i(t) w_i(t) & q > 0 \\ \mathbf{max} \left\{ 0, -c + \sum_{i=1}^{N} R_i(t) w_i(t) \right\} & q = 0 \end{cases} \end{cases} \tag{3.5}$$

Dans ces équations, w_i représente la taille moyenne de la fenêtre de TCP (exprimée en paquets) de la connexion i, q la taille moyenne de la file d'attente (exprimée en paquets), N le facteur de charge du réseau (*i.e.*, nombre de connexions TCP), p la probabilité de perte d'un paquet dans la file d'attente (dont la valeur se situe entre 0 et 1), τ le délai de retour d'informations et R_i le RTT de la connexion i (exprimé en secondes). Le RTT (R_i) de la connexion i est défini de la manière suivante :

$$R_i(t) = \frac{q(t)}{c} + T_p \tag{3.6}$$

Dans cette équation, c représente la bande passante du lien congestionné (exprimé en paquets/secondes), et T_p le temps de propagation (exprimé en secondes). Dans ce calcul du RTT, il est supposé qu'il n'y a qu'une seule congestion sur le chemin allant de la source à la destination. Cependant, notons que ce calcul est généralisable à n congestions.

Lorsqu'un paquet est perdu, l'indication de sa perte parvient à la source environ τ secondes après que le paquet ait été détruit dans la file d'attente. Le délai de retour d'informations (τ_i) de la connexion i est modélisé comme étant la solution du système d'équations suivant :

$$\begin{cases} t = R_i(t') + t' \\ t' = t - \tau_i \end{cases} \tag{3.7}$$

Dans le système d'équations 3.5, la première équation différentielle décrit la dynamique de contrôle de la fenêtre de congestion de TCP en phase d'*évitement de congestion*. En effet, le premier terme ($\frac{1}{R_i(t)}$) décrit la phase d'accroissement additif de la fenêtre de congestion (*Additive Increase*). Le second terme ($\frac{w_i(t)}{2}$) décrit la décroissance exponentielle de la fenêtre de congestion (*Multiplicative Decrease*). Le dernier terme ($p(t-\tau)\frac{w_i(t-\tau)}{R_i(t-\tau)}$) représente la probabilité de perte d'un paquet. Cette probabilité est proportionnelle au débit du flot ($\frac{w_i(t-\tau)}{R_i(t-\tau)}$) et varie en fonction de la taille de la file d'attente ($p(t-\tau)$). En effet, plus un flot occupe une part importante de la bande passante du lien congestionné, plus ses paquets ont de chance d'être détruits dans la file d'attente. Dans [62], Kelly propose un modèle similaire où seule la probabilité de perte d'un paquet (p) est modifiée.

La seconde équation différentielle décrit la taille de la file d'attente du lien congestionné comme étant la différence entre le débit d'arrivée des paquets ($\sum_{i=1}^{N}(\frac{w_i(t)}{R_i(t)})$) et la bande passante du lien congestionné (c).

Ce modèle semble relativement simple, mais son comportement peut être très compliqué, particulièrement si la file d'attente implémente une politique de type *Drop Tail*. En effet, si l'on considère que la probabilité de perte d'un paquet s'exprime par :

$$p(q) = \begin{cases} 1, & \text{si } q > \bar{q} \\ 0, & \text{sinon} \end{cases} \tag{3.8}$$

ce qui traduit une politique *Drop Tail*, alors il a été prouvé dans [115] que le mécanisme de contrôle de congestion de TCP adoptait un comportement chaotique.

Afin de réduire le degré de conservation de la fonction de probabilité de perte, il faut ajouter une hypothèse assez intuitive : moins le débit entrant sur le lien congestionné est important, moins la probabilité (p) de perdre un paquet dans la file sera importante. Réciproquement, plus le débit entrant sur le lien congestionné est important, plus la probabilité de perdre un paquet sera importante. En d'autres termes, il faut supposer que la probabilité de pertes est une fonction croissante, ce qui revient à utiliser une politique RED pour la gestion de la file :

$$p(q) = \begin{cases} 0, & q(t) \leq q_{min} \\ \frac{p_{max}}{q_{max}-q_{min}}(q(t)-q_{min}), & q_{min} < q(t) < q_{max} \\ 1, & q(t) \geq q_{max} \end{cases} \tag{3.9}$$

Dans ces équations q_{min}, q_{max} et $p_{max} \in [0\ ;\ 1]$ doivent être spécifiés et représentent les paramètres d'une file RED.

Notons qu'en se basant sur les simulations de [74, 50], il semble que le modèle (3.5)-(3.6) représente plus précisément la dynamique de la fenêtre de congestion de TCP, lorsque le mécanisme de *Timeout* est ignoré.

3.1.3 Conclusion

Les deux modélisations présentées permettent d'étudier plus simplement le contrôle de congestion dans un réseau à commutation de paquets.

Le premier modèle décrit, grâce à un système de deux équations, l'évolution du débit d'émission (x) et celle de la taille de la file d'attente (q). La transformation de l'équation de débit en une équation du second ordre, permet de ramener le problème de contrôle de congestion à l'étude de la stabilité asymptotique d'un système à retard. Les problèmes de stabilité sont largement connus et traités en automatique, ce qui permet de trouver des solutions exactes à ce type de problème complexe. Malheureusement, dans le modèle d'Izmailov, la « taille idéale » de la file d'attente est supposée connue, ce qui est impossible dans la réalité. En effet, ne sachant pas quels sont les routeurs traversés, ni le nombre de connexions se partageant le lien congestionné, il est impossible de l'estimer. Ces travaux ne semblent donc pas applicables dans un environnement réel.

Le second modèle présenté décrit la dynamique de la fenêtre de congestion d'une source TCP. Cette modélisation permet d'étudier le comportement de TCP dans un environnement continu, où tous les paramètres sont maîtrisés. Cette maîtrise, autorise une analyse plus fine des algorithmes de contrôle de congestion.

Beaucoup d'autres travaux analysant les caractéristiques du mécanisme AIMD[1]. [88, 7, 104, 72, 76, 62, 68] ont été menés. Ces travaux portent principalement sur l'étude du régime stationnaire et de l'équité. Enfin, de récentes recherches sur le contrôle de congestion au niveau de la couche réseau utilisent également des modèles continus [69, 14, 40, 78].

L'ensemble de ces travaux nous amène à nous demander quel est leur domaine de validité (continu, discret). En effet, ces études étant menées dans un environnement continu alors que les réseaux sont intrinsèquement discrets, on peut se demander si les comportements observés

1. *Additive Increase Multiplicative Decrease*.

en continu sont cohérents avec ceux obtenus en discret. Cette question nous a poussés à préciser le domaine de validité de l'approximation continue.

3.2 Modèle expérimental

Dans cette section, nous décrivons la méthode employée pour étudier l'influence de l'« absence de discrétisaton » sur la modélisation des réseaux à commutation de paquets. Pour cela, les comportements discret et continu d'un protocole de transport sont comparés. Les simulations continues seront effectuées avec *Matlab* alors que les discrètes seront effectuées grâce au simulateur *ns* [86] (ce simulateur est présenté dans la section 5.2). De manière à ne pas entacher les résultats d'erreurs dues à des imperfections de modélisations, le protocole étudié admet une modélisation continue exacte.

3.2.1 Contexte de simulation

Le réseau utilisé pour la simulation est composé de n nœuds source S_1 à S_n, d'un seul nœud de congestion C, et d'un seul nœud destinataire D (*cf.* figure 3.1).

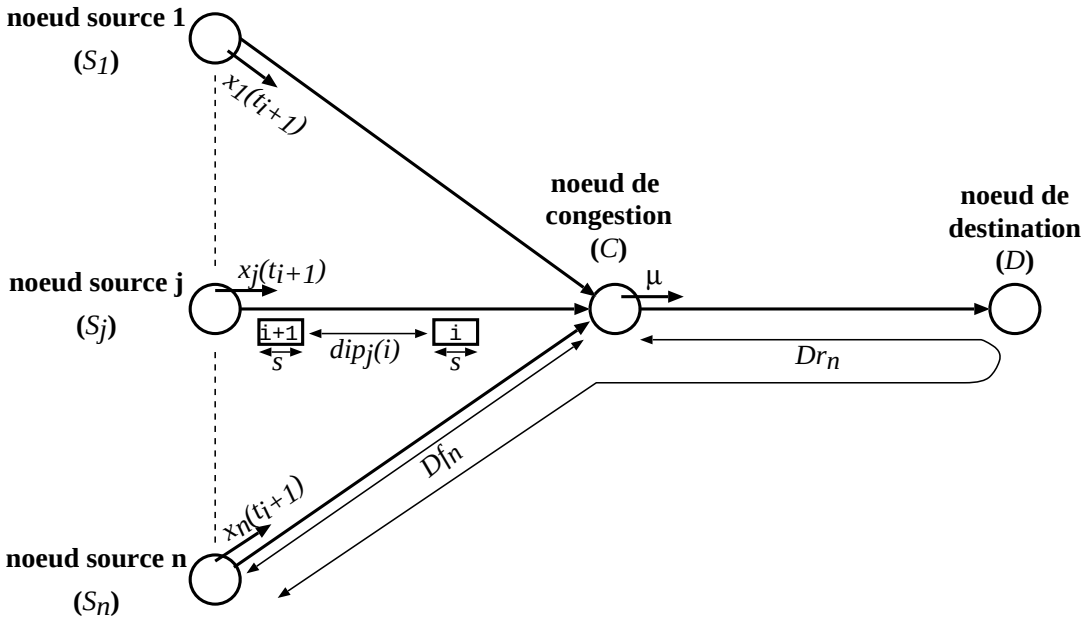

FIG. 3.1 – *Topologie de simulations.*

Notons que, dans le cas simulé, les congestions sont dues à une limitation de la bande passante du lien partagé et non à une puissance de calcul insuffisante des routeurs intermédiaires. De par la topologie utilisée, il est supposé qu'il y a un unique goulot d'étranglement entre la source et la destination, mais qu'il n'y en a aucun dans le sens du retour (entre la destination et la source). Les délais de propagation des liens allant des sources au nœud de congestion et du nœud de congestion aux sources (nommés respectivement Df_j et Dr_j) sont supposés constants (*cf.* figure 3.1).

Le goulot d'étranglement a une bande passante fixée (μ kbits/s) de sorte que : μ soit inférieur à b_{SC} qui représente la somme de la bande passante des liens entrant au nœud de congestion.

La file d'attente utilise une politique FIFO[2] et est de taille infinie (pour éviter les pertes de paquets). L'utilisation d'une file de taille infinie permet de simplifier la comparaison des modèles discret et continu. Mais surtout, elle évite d'entacher les résultats d'erreurs dues à une mauvaise modélisation continue du mécanisme de destruction de paquets des files d'attente (type *Drop Tail*).

3.2.2 Protocole

Le protocole utilisé implémente un schéma de contrôle de congestion de type AIMD. Un nœud source envoie un paquet de données à un nœud destinataire, qui retourne un acquittement à la source. Au début de la communication, le réseau est supposé vide (*i.e.*, aucun paquet ne circule sur le réseau), il n'y a donc pas de congestion. Tous les paquets de données émis par la source sont de taille fixe (notée s) et contiennent dans leur en-tête une estampille indiquant leur heure d'émission. À la réception d'un paquet de données, le destinataire recopie cette estampille dans l'en-tête de l'acquittement avant de l'envoyer à la source. À la réception de cet acquittement, la source n pourra calculer le RTT du $i^{\text{ème}}$ paquet, noté $rtt_n(i)$:

$$rtt_n(i) = \text{heure\_réception} - \text{estampille\_de\_l'ack} \tag{3.10}$$

Le premier RTT calculé alors que le réseau est encore vide, est noté RTT_n^{min} et sera utilisé comme délai d'acheminement de référence. Ce délai de référence représente le RTT le plus petit que pourra atteindre la connexion n. Pour tout paquet i, on a :

$$rtt_n(i) \geq RTT_n^{min} \tag{3.11}$$

Dans ce protocole, le débit d'émission est ajusté en fonction du taux de remplissage de la file d'attente du nœud congestionné. La source estime le taux de remplissage de la file d'attente en comparant le RTT instantané à celui de référence (RTT_n^{min}) : plus la différence constatée est importante, plus la file est pleine. Le taux de remplissage désiré par le protocole pour la file d'attente est représenté par la *tolérance* T exprimée en millisecondes. Par exemple, si $T = s/\mu$ (où μ représente le débit du lien congestionné), la taille requise pour la file d'attente sera d'un paquet. Notons que, plus la tolérance est importante, plus le protocole laissera la file d'attente du nœud congestionné se remplir avant de réduire son débit.

On peut distinguer deux cas pour l'ajustement du débit :

– la taille de la file d'attente est inférieure à celle souhaitée :

$$rtt_n(i) \leq RTT_n^{min} + T \tag{3.12}$$

Dans ce cas, le débit de la source augmente linéairement.

2. *First In First Out.*

– la file d'attente a dépassé la taille souhaitée :

$$rtt_n(i) > RTT_n^{min} + T \tag{3.13}$$

Dans ce cas, le débit diminue exponentiellement.

En discret, la source n contrôle son débit d'émission $x_n(t)$ en ajustant le délai inter-paquets dip_n. Soit t_i l'heure à laquelle le $i^{\text{ème}}$ paquet (de taille s) est envoyé, le délai inter-paquets $dip_n(i)$ séparant l'émission du $i^{\text{ème}}$ et du i+1$^{\text{ème}}$ paquet est calculé de la manière suivante :

$$dip_n(i) = \frac{s}{x_n(t_i)} \tag{3.14}$$

Afin d'éviter les situations de blocage infini, le protocole a un débit d'émission minimale de 3000 bits/s. En effet, si le débit de la source tombait à une valeur proche de 0, le délai inter-paquets tendrait vers l'infini. Dans une telle situation, plus aucun paquet ne serait émis et donc aucun acquittement ne serait reçu. La source ne pourrait donc plus être avertie de la fin de la congestion et la transmission des données serait bloquée.

3.2.3 Modélisation continue

Les simulations en continu ont été effectuées avec *Matlab*, le code sources utilisé se trouve en annexe B.1. Le modèle continu est composé de deux fonctions continues $q(t)$ et $x(t)$:

- $q(t)$ représente la taille de la file d'attente du nœud de congestion C à l'instant t
- $x_n(t)$ donne le débit de la source S_n à l'instant t

Lorsque des données de la source n arrivent au goulot d'étranglement à l'instant t, le débit auquel elles ont été émises est égal à $x_n(t - Df_n)$. En effet, elles ont été envoyées par la source à l'instant $t - Df_n$ (*cf.* figure 3.1). Les données sont ensuite envoyées du nœud de congestion vers la destination. Le débit auquel sont envoyées les données dépend de l'état de la file d'attente :

– si la file d'attente n'est pas vide, les données seront envoyées avec le débit μ du lien congestionné ;

– si la file d'attente est vide et que le débit d'émission des données est inférieur à μ, elles seront émises avec un débit égal à $x_n(t - Df_n)$.

Les conditions précédentes permettent de définir un système d'équations modélisant l'évolution de la taille de la file d'attente (notons que, pour la simplicité du modèle continu, nous considérerons des conditions initiales nulles) :

$$\begin{array}{ll} \text{Si} & \exists\, \epsilon > 0,\ \forall\, t' \in [t-\epsilon,\ t[,\ q(t') = 0 \text{ et } \sum_{j=1}^{n} x_j(t - Df_j) \leq \mu \quad \text{alors} \\ & q(t) = 0 \\ \text{Sinon} & \\ & q(t) = \int_{t_{\textit{file non vide}}}^{t} \left(\left(\sum_{j=1}^{n} x_j(t - Df_j) \right) - \mu \right) dt \end{array} \tag{3.15}$$

Dans ces équations, $\sum_{j=1}^{n} x_j(t - Df_j)$ représente le débit entrant au nœud C à l'instant t, c'est-à-dire, la somme du débit des sources à l'instant $t - Df_j$. La variable $t_{\textit{file non vide}}$ contient l'heure à laquelle la file devient non vide (*i.e.*, $\sum_{j=1}^{n} x_j(t_{\textit{file non vide}} - Df_j) = \mu$).

Les données arrivant à l'instant t dans la file d'attente y passeront $q(t)/\mu$ secondes avant d'en sortir. L'équation donnant le temps passé dans la file d'attente, noté *q_attente*, par un paquet en fonction de son heure de sortie de la file est notée :

$$q\_attente\left(t+\frac{q(t)}{\mu}\right)=\frac{q(t)}{\mu} \tag{3.16}$$

Si un acquittement est reçu par la source n à l'heure t, cela signifie que le paquet correspondant à cet acquittement a quitté la file d'attente à l'instant $t - Dr_n$, ce qui veut dire qu'il est arrivé dans la file d'attente à $t - Dr_n - q\_attente(t - Dr_n)$.

À l'arrivée d'un acquittement à l'heure t, si

$$\frac{q\left(t-Dr_n-q\_attente(t-Dr_n)\right)}{\mu}>T, \tag{3.17}$$

cela signifie que le paquet ayant engendré l'émission de l'acquittement a passé dans la file d'attente un temps supérieur à la tolérance T. Or, si la durée de l'attente tolérée a été dépassée, cela sous-entend que la taille souhaitée pour la file d'attente a également été dépassée. Afin de diminuer la taille de la file d'attente, le débit de la source ayant reçu l'acquittement diminue exponentiellement. Inversement, si le temps passé dans la file d'attente avait été inférieur ou égal à la tolérance T, le débit aurait augmenté linéairement.

Par conséquent, nous avons :

$$\begin{array}{ll}\text{Si} & \frac{q(t-Dr_n-q\_attente(t-Dr_n))}{\mu}\leq T \quad \text{alors}\\ & x_n(t)=deb_n^{min}+\alpha\times(t-t_n^{min})\\ \text{Sinon} & \\ & x_n(t)=\max\left(deb_n^{max}\times\exp\left(\frac{t-t_n^{max}}{\beta}\right),3000\right)\end{array} \tag{3.18}$$

Dans ce système d'équation, t_n^{min} représente l'heure à laquelle la source a détecté qu'elle pouvait de nouveau augmenter son débit, et deb_n^{min} le débit auquel elle émettait à cet instant (*cf.* figure 3.2). De même, la variable $t_n^{\max}$ représente l'heure à laquelle la source a détecté qu'elle devait réduire son débit, et deb_n^{max} le débit auquel elle émettait à cet instant.

Ces quatre nouvelles variables peuvent être décrites formellement :

$$\begin{array}{lll}t_n^{min}=t^{attente\leq T}+Dr_n & \text{et} & deb_n^{min}=x_n(t^{attente\leq T}+Dr_n)\\ t_n^{\max}=t^{attente>T}+Dr_n & \text{et} & deb_n^{max}=x_n(t^{attente>T}+Dr_n)\end{array} \tag{3.19}$$

La variable $t^{attente\leq T}$ (respectivement $t^{attente>T}$) représente l'heure à laquelle la taille de la file du nœud de congestion est repassée en dessous (respectivement au dessus) de la taille souhaitée. Ces deux variables sont définies par le système d'équations suivant :

$$\begin{cases}\exists\,\epsilon>0, \text{tel que } \forall\, t'\in[t^{attente\leq T}-\epsilon,\ t^{attente\leq T}[,\\ \qquad q\_attente(t')>T \text{ et } q\_attente(t^{attente\leq T})\leq T\\ \\ \exists\,\epsilon>0, \text{tel que } \forall\, t'\in[t^{attente>T}-\epsilon,\ t^{attente>T}[,\\ \qquad q\_attente(t')\leq T \text{ et } q\_attente(t^{attente>T})>T\end{cases} \tag{3.20}$$

Initialement, $deb_n^{min}=deb_n^{max}=3000$, $t_n^1=t_n^{min}=t_n^{max}=0$, et la file d'attente est vide ($q(0)=0$).

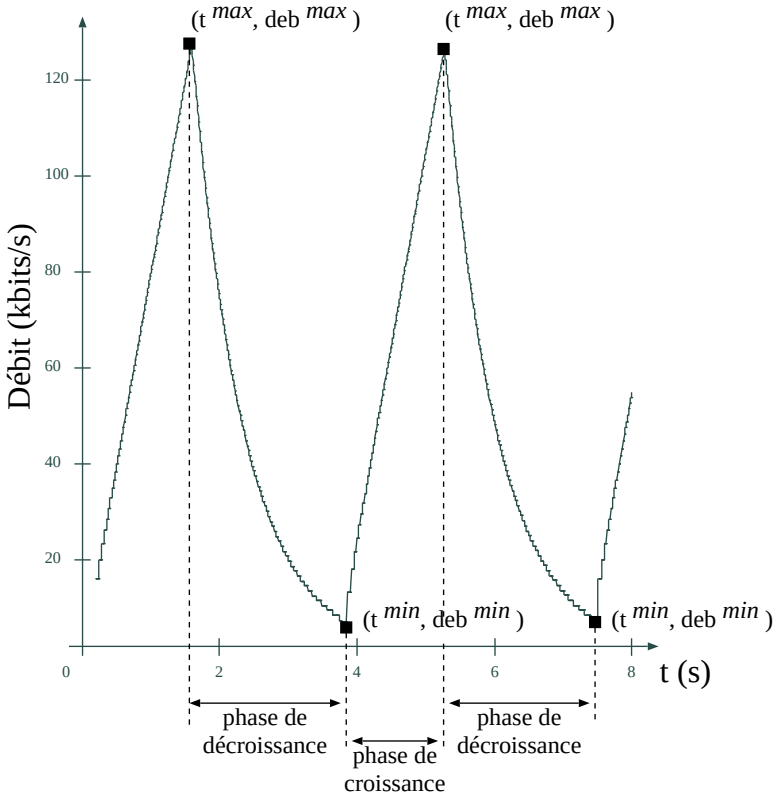

FIG. 3.2 – *Illustration du changement de phase.*

3.2.4 Implémentation discrète

L'implémentation discrète du protocole a été réalisée sous *ns* [86] (*cf.* section 5.2). Comme dans le modèle continu, le protocole utilise trois paramètres (*cf.* équations 3.18) pour ajuster le débit d'émission :

- deux coefficients de correction (le facteur de croissance linéaire α, et le facteur de décroissance exponentielle β),
- un paramètre de réactivité (la tolérance T).

Plus α sera grand, plus le débit augmentera vite dans les phases d'accroissement du débit. Inversement, plus β sera petit, plus le débit diminuera rapidement lors des phases de réduction de débit.

L'algorithme 1 (*cf.* figure 3.3), utilisé pour l'implémentation discrète du protocole se décompose en deux fonctions : *émission* et *réception*.

La fonction d'émission permet d'envoyer les paquets de données et d'en calculer le débit d'émission. Dans chaque paquet envoyé, la taille du paquet (tp octet) ainsi que son heure d'émission (t) sont indiquées. Comme en continu, le débit d'émission se calcule de manière différente suivant que l'émetteur se trouve en phase d'augmentation de débit (*i.e.*, la taille de la file d'attente est inférieure à celle souhaitée) ou en phase de réduction de débit (*i.e.*, la taille souhaitée a été dépassée).

En phase de réduction de débit (*cf.* lignes 14 à 20 de l'algorithme 1 : phase décroissante) le débit instantané est calculé grâce à β, à la dernière mesure du débit maximal *débit_max* (mesuré

au début de la phase de réduction actuelle), et à l'heure *t_max* à laquelle a commencé cette phase.

De même, en phase d'augmentation de débit (*cf.* lignes 21 à 28 de l'algorithme 1) le débit instantané est calculé grâce à α, à la dernière mesure du débit minimal *débit_min* (mesuré au début de la phase d'accroissement actuelle), et à l'heure *t_min* à laquelle a commencé cette phase.

La variable *phase* permet de distinguer si le changement de phase vient d'avoir lieu et, si c'est le cas, le protocole récupérera les valeurs des variables *t_min* et *débit_min* ou *t_max* et *débit_max*, liées au changement de phase.

La fonction de réception est appelée à chaque arrivée d'un paquet. Elle permet :

- d'envoyer un acquittement (*cf.* lignes 33 et 34 de l'algorithme 1) si le paquet reçu contenait des données. Dans ce cas, l'acquittement envoyé contiendra dans son en-tête une copie de l'heure d'émission du paquet reçu, ainsi que sa taille (ces deux informations étant indiquées dans l'en-tête du paquet reçu) ;
- de calculer le RTT si le paquet reçu était un acquittement (*cf.* lignes 36 à 41 de l'algorithme 1). Dans ce cas, le protocole commence par tester s'il s'agit du premier acquittement reçu (*i.e.*, première mesure du RTT). Si c'est le cas, la valeur minimale du RTT utilisée pour estimer la taille des files d'attente est enregistrée (*RTT_min = t - date_émission*). Sinon, le RTT instantané est mesuré (*rtt = t - date_émission*).

Le code source utilisé pour l'implémentation sous *ns* est disponible en annexes B.2.

3.3 Méthode de comparaison

3.3.1 Critère de comparaison

Pour comparer le comportement des modèles continu et discret, un indicateur numérique a été défini. Nous avons remarqué que, pour la plupart des cas simulés, le débit des sources était périodique. En effet, l'alternance des phases d'augmentation et de réduction de débit est périodique pour le modèle continu et quasi périodique pour le modèle discret (*cf.* figure 3.4).

De par les caractéristiques du protocole étudié, nous savons que l'amplitude d'une courbe de débit est dépendante de sa période (*i.e.*, l'amplitude est une fonction du temps et du changement de phase donc de la période). On peut affirmer que, si deux courbes ont la même période, elles auront la même amplitude et seront par conséquent équivalentes. Si les sources ayant généré ces courbes de périodes identiques possèdent les mêmes paramètres α, β et *T*, on peut considérer qu'elles ont un comportement identique. La période semble donc être un bon indicateur pour comparer le comportement des modèles continu et discret.

Notons que cette méthode de comparaison ne tient pas compte du déphasage de courbes, mais cela ne fausse pas les résultats obtenus. En effet, si deux courbes de débit ont des périodes très légèrement différentes, à l'issue d'une longue simulation, cette différence finira par induire un déphasage important. Or, en utilisant une technique de comparaison de type moindre carré ou soustraction des deux courbes, le déphasage important de la fin de simulation indiquerait que ces deux sources ont un comportement totalement différent. Cette indication serait erronée car le comportement d'un protocole est représenté par la manière globale dont le débit varie et non par les dates auxquelles les changements de phase (augmentation et réduction) sont effectués.

Algorithme 1 : **Protocole(α, β, T)**

```
1   phase = croissante
2   débit_min_absolu = 3000
3   débit_min = débit_min_absolu
4   t_min = 0
5   débit_max = débit_min_absolu
6   t_max = 0
7   débit = débit_min
8   RTT_min = 0
9   rtt = rtt_min
10  tp = 1000
11  émission() :
12      t = date()
13      envoyer( tp octets, t ) au nœud destinataire
14      si rtt > RTT_min + T alors
            ▷ Le débit d'émission doit être réduit.
15          si phase == croissante alors
                ▷ Entrée en phase décroissante.
16              phase = décroissante
17              débit_max = débit
18              t_max = t
19          fin si
20          débit = max( débit_max × exp( (t_min - t) / β ) , débit_min_absolu)
21      sinon
            ▷ Le débit doit augmenter.
22          si phase == décroissante alors
                ▷ Entrée en phase croissante.
23              phase = croissante
24              débit_min = débit
25              t_min = t
26          fin si
27          débit = débit_min + α × (t - t_min)
28      fin si
29      appeler émission() à la date t + tp / débit
30  réception() :
31      t = date()
32      si paquet == données alors
            ▷ Réception d'un paquet de données par le nœud destinataire.
33          recevoir( tp octets, date_émission )
34          envoyer( date_émission ) au nœud émetteur
35      sinon
            ▷ Réception d'un acquittement par le nœud source.
36          recevoir( date_émission )
37          si RTT_min == 0 alors
38              RTT_min = t - date_émission
39          sinon
40              rtt = t - date_émission
41          fin si
42      fin si
```

FIG. 3.3 – *Algorithme utilisé pour l'implémentation du protocole dans ns*

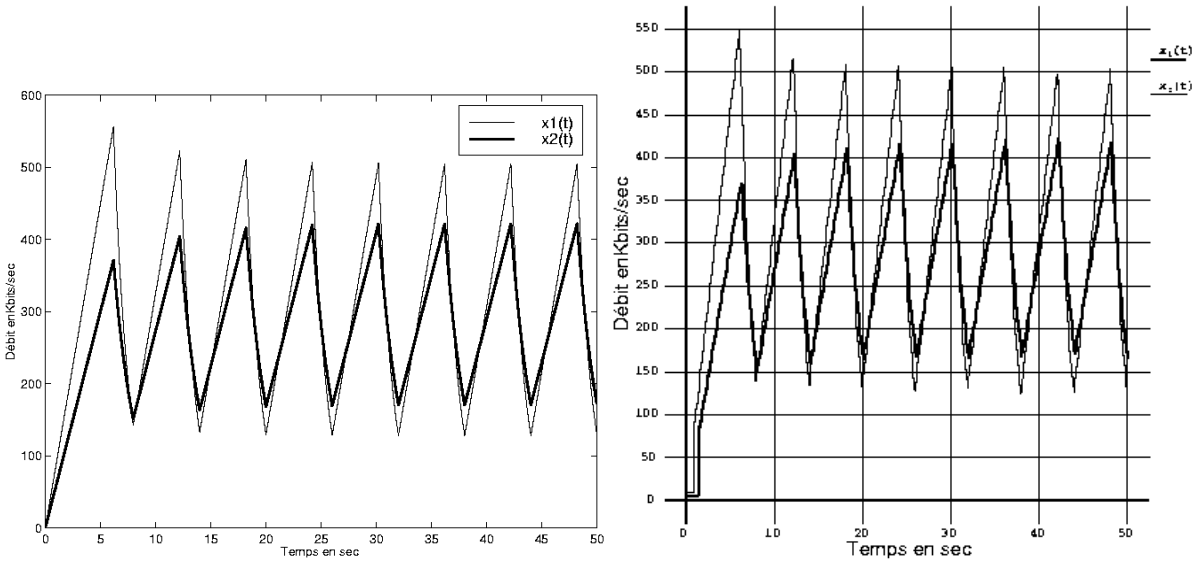

FIG. 3.4 – *Les courbes de débits obtenues avec le modèle continu (à gauche) et le modèle discret (à droite) représentent les débits* $x_1(t)$ *et* $x_2(t)$ *de deux sources.*

La période est un indicateur simple qui fournit des résultats cohérents. Elle sera donc utilisée pour comparer le comportement du protocole dans des environnements continu et discret.

Lors des premières simulations, nous nous sommes aperçus que le modèle discret n'était pas rigoureusement périodique. En effet, sur une simulation de 300 s, il est possible d'observer de faibles variations de la période. Afin que la comparaison des périodes continue et discrète ne soit pas faussée par les variations minimes de la période du modèle discret, nous utiliserons la valeur *période discrète* qui représente la moyenne des périodes calculée sur 300 s (entre 30 et 75 périodes suivant les paramètres de simulation) :

$$\textit{période discrète} = \frac{\sum \textit{durée période}}{\textit{nb périodes}} \tag{3.21}$$

Pour le modèle continu, les courbes de débit étant complètement périodiques, il suffit d'utiliser une mesure de la période pour comparer les deux modèles.

Afin de faciliter la lecture des résultats, l'indicateur *écart-CD* (pour écart Continu Discret) a été défini. Il représente (sous forme de pourcentage) l'écart constaté entre la période obtenue avec le modèle continu et celle obtenue avec le modèle discret :

$$\textit{écart-CD} = \left| 100 - \left(\frac{\textit{période discrète}}{\textit{période continue}} \times 100 \right) \right| \tag{3.22}$$

3.3.2 Description des simulations

Hypothèses de simulation. Pour étudier la cohérence entre les modèles continu et discret, plusieurs ensembles de simulations ont été nécessaires. Une attention particulière a été prêtée à l'influence de certains paramètres (du réseau ou du protocole lui-même) sur la cohérence des deux modèles. Cependant, il est difficile d'isoler un paramètre des autres, car ils sont souvent

corrélés. De plus, il est impossible de simuler toutes les situations existantes. Des choix pertinents étaient donc indispensables. Ces choix ont été faits aux vues des résultats obtenus lors de simulations préliminaires.

Les simulations ont permis de différencier plusieurs classes de comportements. Par exemple, le couple de valeurs $\alpha = 60$ et $\beta = 2$ (respectivement $\alpha = 800$ et $\beta = 0.75$) correspond à des variations « lissées » (respectivement « fortes ») du débit d'émission. De la même manière, le protocole peut être plus ou moins sensible à la détection des congestions. En effet, plus la tolérance *T* est faible, plus le protocole détecte rapidement l'apparition d'une congestion. Lorsque la tolérance est faible, on dit du protocole qu'il est réactif. La valeur la plus couramment utilisée dans les simulations a été la valeur minimale qui est égale au nombre de sources émettant simultanément.

Proposition 1 *La tolérance T ne peut pas être inférieure à $n - 1$ paquets, n représentant le nombre de sources. En effet, dans le cas contraire, il y aurait toujours au moins une source qui agirait comme s'il y avait congestion (i.e., absence de périodes sans congestion).*

La topologie utilisée pour les simulations est composée de n nœuds sources, d'un routeur et d'un nœud de destination (*cf.* figure 3.1).

Simulations en rapport avec les paramètres du réseau

À l'exception de la taille de la file d'attente (qui est de taille infinie), tous les paramètres liés aux caractéristiques du réseau ont variés afin d'en évaluer l'influence sur la cohérence des comportements du protocole dans des environnements continu et discret. Les conditions des différentes simulations sont maintenant détaillées.

Variation des délais de propagation. Pour cet ensemble de simulations, le réseau est composé de deux sources et d'une seule destination. Les deux connexions ont des temps de propagation hétérogènes (les délais d'aller retour minimum sont différents). Cette situation favorise l'inéquité entre les flots et devrait donc accentuer les éventuelles différences entre les modélisations fluide et discrète.

Nous sommes repartis de la première étude menée sur la cohérence des modèles discret et continu [13] dans laquelle la congestion était située au milieu du chemin. Nous avons ajouté à ce cas d'étude un second contexte de simulations où la congestion ne se trouve plus au milieu du chemin. Afin d'étudier l'importance des délais de propagation, deux cas ont donc été distingués :

- **Liens à délais de propagation symétriques**. Dans ce cas, le lien entre la source (S_n) et le nœud de congestion (C) a le même temps de propagation dans les deux sens (*i.e.*, il est symétrique). En utilisant un lien $S \rightarrow C$ ayant un temps de propagation symétrique, la relation suivante est obtenue : $Df = Dr - 200$ (200 ms représentant le temps aller retour du lien $C \rightarrow D$, *cf.* figure 3.1). Pour ce premier cas, le délai de propagation allant de la première source au nœud de congestion a été fixé à $Df_1 = 100$ ms (*i.e.*, $Dr_1 = 300$ ms). En revanche, le délai de propagation Df_2 a varié de 100 ms à 1,6 s (et Dr_2 de 300 ms à 1,8 s).
- **Liens à délais de propagation asymétriques.** Dans ce cas, en terme de temps de propagation, le nœud de congestion est situé au milieu du chemin $S_n \rightarrow C \rightarrow D \rightarrow C \rightarrow S_n$, ce qui signifie que $Df = Dr$. Les temps de propagation de la première connexion sont tous

deux fixés à 330 ms (*i.e.*, $Df_1 = Dr_1 = 330$ ms). Alors que les temps de propagation Df_2 et Dr_2 ont varié de 330 ms à 1,66 s.

La bande passante du lien partagé (*i.e.*, le goulot d'étranglement) est fixée à 750 kbits/s, et la bande passante des autres liens est de 1 Mbit/s. Le lien allant du nœud de congestion au récepteur est symétrique et a un délai de propagation de 100 ms (200 ms aller retour). Les paramètres du protocole ont été fixés pour l'ensemble de la simulation : $\alpha = 60$, $\beta = 2$, et $T = 2$.

Variation du rapport de bande passante. Afin de mesurer l'influence que peut avoir le rapport de bande passante sortante/entrante au nœud de congestion (*i.e.*, bande passante du lien congestionné divisée par la somme des bandes passantes des liens arrivant au nœud de congestion) sur la cohérence entre les modèles continu et discret, il nous a paru intéressant d'évaluer l'influence du rapport de bande passante avec deux types de variation de débit. Ces deux paramètres (type de variation de débit et rapport de bande passante) sont fortement corrélés. En effet, plus le goulot d'étranglement est « serré », plus la congestion se forme rapidement. De même, plus les variations de débit sont brusques, plus les congestions se forment rapidement. Nous avons donc distingué deux ensembles de simulations en fonction du type de variation de débit :

- **Variations de débit lissées**. Le protocole est paramétré de telle sorte ($\alpha = 60$, $\beta = 2$) que les variations de débits de la source soient lissées.
- **Variations de débit brusques**. Inversement, dans cet ensemble le protocole est paramétré ($\alpha = 800$, $\beta = 0.75$) de façon à ce que les variations de débits de la source soient fortes.

Dans les deux situations, on a considéré une réactivité maximale. La tolérance a été fixée à 0 paquet dans la file, la détection de congestion sera donc rapide voire prématurée.

Le réseau utilisé pour ces ensembles de simulations est composé d'une unique source et la bande passante du lien entrant sur le nœud de congestion (lien $S_1 \rightarrow C$) est de 1,5 Mbits/s. La bande passante du lien congestion μ varie de 100 kbits/s à 900 kbits/s. Les temps de propagation des liens ont été fixés de sorte que $Df = Dr = 330$ ms.

Variation du nombre de sources. Cet ensemble de simulations cherche à déterminer l'influence de l'interaction des connexions lorsque leur nombre augmente sur la cohérence des modèles continu et discret. Le nombre de connexions se partageant un même lien congestionné varie de 1 à 25. Les autres paramètres du réseau sont fixés pour l'ensemble de simulations. Le rapport entre la bande passante du lien congestionné et la somme des bandes passantes des liens entrants (ayant un débit de 1 Mbit/s) au nœud C est de 50% (*i.e.*, la valeur de la somme des bandes passantes des liens entrants est 2 fois plus importante que celle du lien congestionné). Les temps de propagation des liens ont été fixés de sorte que $Df = Dr = 330$ ms. Les paramètres du protocole sont également fixés $\alpha = 60$, $\beta = 2$, $T = n$.

Simulations en rapport avec les paramètres du protocole

Un deuxième de groupe de simulations dans lequel l'influence des paramètres du protocole sur la cohérence des modèles continu et discret a été testé. Afin d'estimer l'impact des paramètres du protocole, les paramètres du réseau (à l'exception du nombre de sources) ont été fixés.

Coefficients de correction du débit (α et β). Afin d'évaluer l'influence des coefficients de correction (*i.e.*, le degré de réaction du protocole) sur la cohérence entre les modèles continu et discret, des simulations ont été effectuées sur des réseaux composés de 1, 2 et 10 sources. Nous avons jugé pertinent d'évaluer la corrélation qu'il pouvait y avoir entre le nombre de connexions se partageant un même lien et le type de variation de débit qu'elles utilisent. Les autres paramètres du réseau ont été fixés de la manière suivante : $Df = Dr = 330$ ms et $\mu = b_{SC}/2$. La plage de valeurs utilisées pour les coefficients de correction α et β a permis de tester le protocole avec de fortes et de faibles variations du débit d'émission (*i.e.*, α a varié de 20 à 1600, et β de 6 à $0,25$). Afin que le protocole conserve un comportement cohérent, il faut s'assurer que la rapidité d'augmentation du débit est proportionnelle à celle de réduction. Il faut donc que les variations des coefficients α et β soient inversement proportionnelles : lorsque le débit augmente rapidement (*i.e.*, la valeur d'α est importante) en phase croissante, il doit diminuer rapidement en cas de congestion (*i.e.*, la valeur d'β est faible).

Taille des paquets (s). Dans cet ensemble de simulations, l'influence de la taille des paquets sur la cohérence entre les modèles continu et discret a été évaluée. Un réseau composé de dix sources a été utilisé pour réaliser cette évaluation. Les temps de propagation ont été fixés à $Df = Dr = 300$ ms et le rapport bande passante sortante/entrante à 50%. Les paramètres du protocole ont également été fixés à $T = 10$ paquets pour la réactivité, $\alpha = 60$ et $\beta = 2$ pour les coefficients de correction. Intuitivement, on peut penser que plus les paquets sont petits, plus le modèle discret se rapprochera du modèle continu (*i.e.*, plus les paquets sont petits, plus le flot de données entre l'émetteur et le récepteur sera « continu »). Il est donc intéressant d'observer l'impact de la taille des paquets sur la cohérence des modèles continu et discret. Pour ce faire, nous avons fait varier la taille des paquets de 0,1 ko à 50 ko.

Réactivité (T). Le dernier paramètre observé afin d'étudier la cohérence entre les modèles discret et continu est la réactivité du protocole (*i.e.*, le seuil de détection de congestion). Ce paramètre a été testé sur des réseaux composés de 2, 3, 5, 10 et 25 sources. Les temps de propagation ont été fixés à $Df = Dr = 300$ ms et le rapport bande passante sortante/entrante à 15%. La réactivité du protocole T a varié du nombre de sources moins un $(n - 1)$ (plus petite valeur acceptée) à 60 paquets. Les autres paramètres du protocole ont été fixés à $\alpha = 60$ et $\beta = 2$.

Ces différents ensembles de simulations multi-critères devraient nous autoriser à estimer les limites de la validité de l'approximation continue.

3.4 Résultats de simulations

Dans cette section, nous présentons les résultats des comparaisons des modèles discret et continu.

3.4.1 Paramètres du réseau

Délais de propagation. Dans cet ensemble de simulations, nous cherchons à estimer l'influence des délais de propagation sur la cohérence des comportements en continu et en discret.

Nous avons distingué deux cas dans lesquels la congestion se trouve ou ne se trouve pas au milieu du chemin. Notons que dans les deux cas, les temps de propagation des deux connexions sont hétérogènes (*cf.* paragraphe 3.3.2).

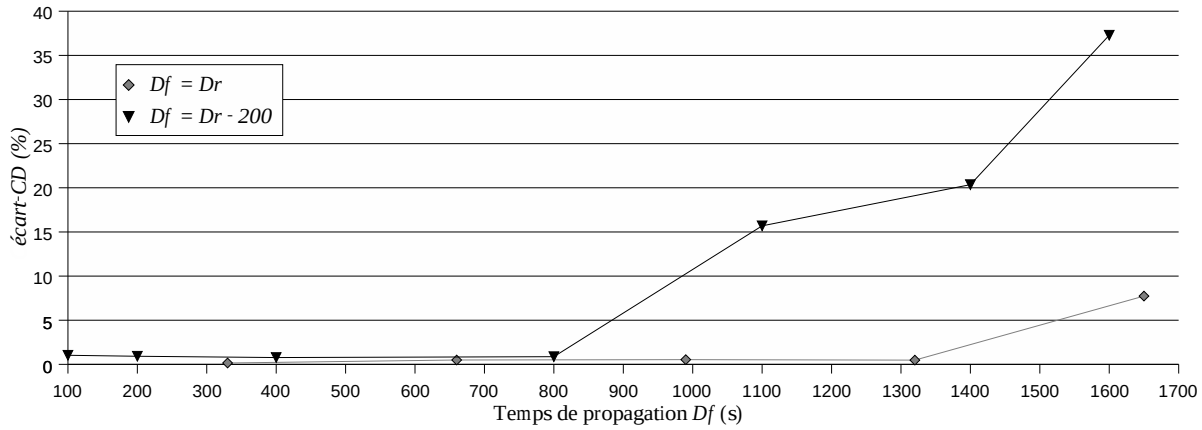

FIG. 3.5 – *Influence des temps de propagation.*

Les résultats obtenus pour les deux cas testés sont présentés dans la figure 3.5. Le temps de propagation des sources ne semble pas avoir une grande importance tant que la différence entre les RTTs minimaux (*i.e.*, les temps de propagation) des deux sources n'excède pas 1,4 s si la congestion n'est pas située au milieu du chemin (et 2,64 s si elle l'est).

Cet ensemble de simulation nous montre que, lorsque le temps de propagation n'est pas trop différent entre les connexions, la modélisation continue est une bonne approximation des réseaux à commutation de paquets (ici l'écart constaté entre les modèles continu et discret est inférieur à 1%). En revanche, de fortes différences entre les deux modèles sont observées lorsque les temps de propagation sont très différents. Notons que sur des réseaux réels, il semble peu probable qu'une connexion ait un temps de propagation dépassant une seconde et demie. Les différences observées ne devraient donc pas poser de problèmes si les réseaux modélisés utilisent des temps de propagation réalistes.

Bande passante du lien congestionné. Dans cet ensemble de simulation, nous cherchons à évaluer l'impact du rapport de la bande passante sur la cohérence des comportements continu et discret d'un réseau à commutation de paquets. Cet ensemble de simulations a été testé avec deux types de correction de débit : brusque et lissé (*cf.* paragraphe 3.3.2).

Les résultats obtenus avec une variation lissée du débit d'émission sont présentés figure 3.6. On peut remarquer que la cohérence des comportements obtenus par les deux modèles est généralement « assez bonne » (*écart-CD* toujours inférieur à 13%), et qu'elle est « excellente » (*écart-CD* inférieur à $3,5\%$) pour un rapport bande passante sortante/entrante supérieur à 13,33% (μ supérieur à 200 kbits/s).

Les résultats obtenus avec un forte variation du débit d'émission sont présentés dans la figure 3.7. Il est important de noter qu'ici, lorsque le rapport bande passante sortante/entrante est « petit », la différence entre les modèles discret et continu est plus marquée (pour un rapport de 6,66%, *écart-CD* $>$ 250%). Cette constatation était prévisible, car en augmentant le risque de congestion et l'amplitude des variations de débit, la situation obtenue est plus propice aux discontinuités qui sont mal restituées par la modélisation continue.

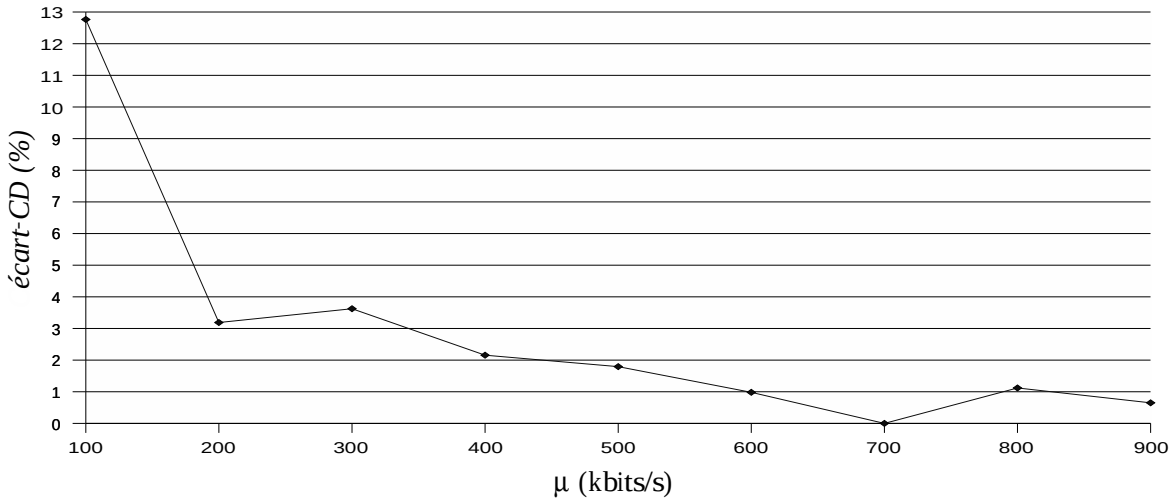

FIG. 3.6 – *Influence de la bande passante avec une variation de débit lissée ($\alpha = 60$, $\beta = 2$).*

Quel que soit le type de variation de débit, la différence entre les deux modèles devient négligeable lorsque le rapport bande passante sortante/entrante est supérieur à 13,33%.

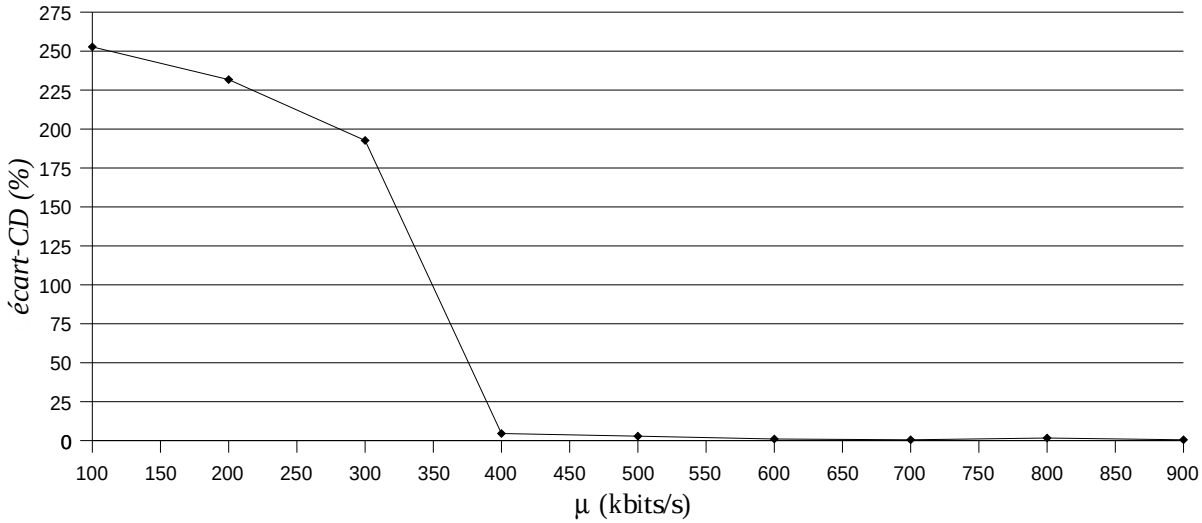

FIG. 3.7 – *Influence de la bande passante avec une forte variation de débit ($\alpha = 800$, $\beta = 0.75$).*

Les résultats obtenus dans ces simulations montrent que le rapport entre la bande passante sortante et entrante au nœud de congestion est un paramètre important pour la cohérence des modèles continu et discret. En effet, plus ce rapport est élevé, moins la différence entre les deux modèles est importante. Notons que ce phénomène est accentué lorsque le protocole utilise des coefficients α et β impliquant de fortes variations de débit. Les brusques variations paraissent favoriser les discontinuités et donc creusent l'écart entre les deux modèles.

Nombre de sources. Cet ensemble de simulations a été exécuté en utilisant le réseau présenté figure 3.1 avec un nombre de nœuds sources n variant de 1 à 25. Les autres paramètres du

réseau, ainsi que, ceux du protocole, étaient fixés (*cf.* paragraphe 3.3.2). Les résultats obtenus sont résumés dans le tableau figure 3.8.

nombre de sources	1	2	3	5	10	25
écart-CD (%)	1,21	0,16	0,00	0,31	0,61	0,12

FIG. 3.8 – *Impact du nombre de sources sur la cohérence des modèles continu et discret ($\alpha = 60$, $\beta = 2$, $T = n$).*

Le plus gros écart constaté entre les deux modèles est de 1,21% lorsque le réseau est composé d'une source. Lorsque le réseau est composé de plusieurs sources, l'écart mesuré entre les deux modèles est vraiment très faible (inférieur à 1%). Cet ensemble de simulations laisse supposer que le nombre de sources n'a pas de réelle influence sur la cohérence des modèles continu et discret.

3.4.2 Paramètres du protocole

Coefficients de correction (α et β). Afin d'évaluer l'influence du type de variation de débit (brusque ou lissé), différents couples de coefficients de correction ont été testés. Ces simulations ont été effectuées avec une topologie composée de 1, 2 et 10 sources, afin de déterminer s'il y a une corrélation entre le nombre de sources et le type de variation de débit (*cf.* paragraphe 3.3.2).

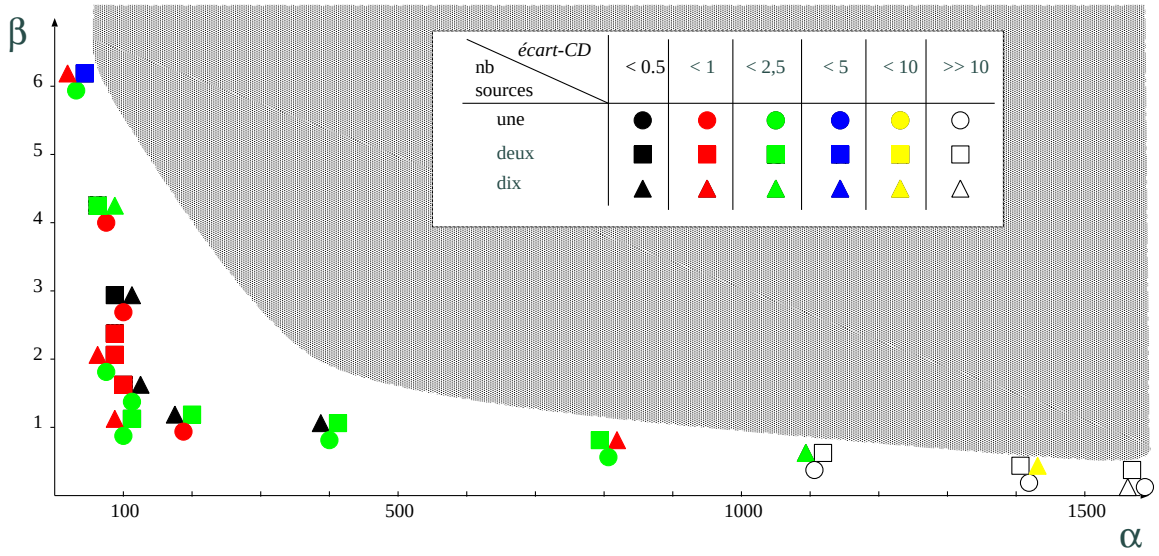

FIG. 3.9 – *Influence des coefficients de correction α et β.*

Les résultats présentés figure 3.9 montrent que la cohérence entre les modèles continu et discret est proportionnelle au lissage du débit d'émission. En effet, plus le débit est lissé (*i.e.*, α petit et β grand), plus les différences constatées entre les deux modèles sont faibles. Inversement, les fortes variations de débit accentuent l'écart entre le modèle discret (propice au régime

discontinu) et le modèle continu (inadapté aux discontinuités). Notons que plus le nombre de sources est élevé, moins l'écart entre les deux modèles est important. Ce phénomène est notamment observable lorsque les variations de débits sont brusques. On peut supposer que plus le nombre de sources est élevé, plus le flot de données en transit sur le lien de congestion sera « continu » et donc facilement restituable par un modèle continu.

Remarque. Il est intéressant de noter que les résultats obtenus (pour 1, 2 et 10 sources), confirment ceux présentés dans [13] pour un réseau composé d'une seule source et vérifiant la condition : $\alpha \times \beta = \mu$.

Taille des paquets (*s*). Dans cet ensemble de simulations, nous avons évalué l'influence de la taille des paquets sur la cohérence des comportements observés en continu et discret. Pour ce faire, nous avons fait varier la taille des paquets de 0,1 ko à 50 ko (*cf.* paragraphe 3.3.2).

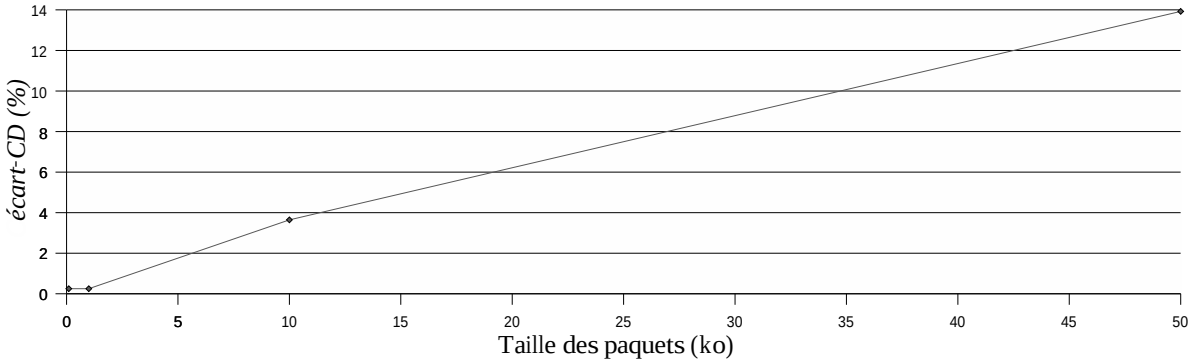

FIG. 3.10 – *Influence de la taille des paquets,* $\alpha = 60$ *et* $\beta = 2$.

Les résultats obtenus (*cf.* figure 3.10) montrent que plus les paquets sont « gros », plus la différence entre les modèles continu et discret est marquée. Cette constatation n'est pas étonnante, car plus les paquets sont petits, plus le comportement du modèle discret s'approche de celui du modèle continu. En effet, lorsque les paquets ont une petite taille, le flot de données en transit sur le modèle discret ressemble à un flot continu : les délais inter-paquets sont très petits.

Réactivité (*T*). Ce dernier ensemble de simulations nous a permis d'étudier l'influence de la réactivité du protocole (*i.e.*, le seuil de détection de congestion) sur la cohérence entre les modèles discret et continu (*cf.* paragraphe 3.3.2). De plus, nous avons souhaité observer s'il y avait une éventuelle corrélation entre le nombre de sources et leur réactivité.

Les résultats de simulations présentés figure 3.11, montrent que lorsque les variations de débit sont lissées ($\alpha = 60$ et $\beta = 2$) et que le rapport bande passante sortante/entrante est raisonnable (supérieur à 15%), la tolérance ne semble pas avoir une grande influence sur la cohérence entre les modèles continu et discret. En effet, l'*écart-CD* ne dépasse jamais 1,35%, ce qui est très faible. On peut également noter que, quel que soit le nombre de sources utilisées, lorsque la tolérance dépasse 40 paquets, l'*écart-CD* paraît converger vers 0,7%. On peut supposer que, pour le modèle discret, avec une tolérance de 40 paquets, la file d'attente n'est jamais vide. Si c'est le cas, le flux de données en transit sur le lien de congestion est quasi continu, et donc le modèle discret s'approche de l'approximation continue.

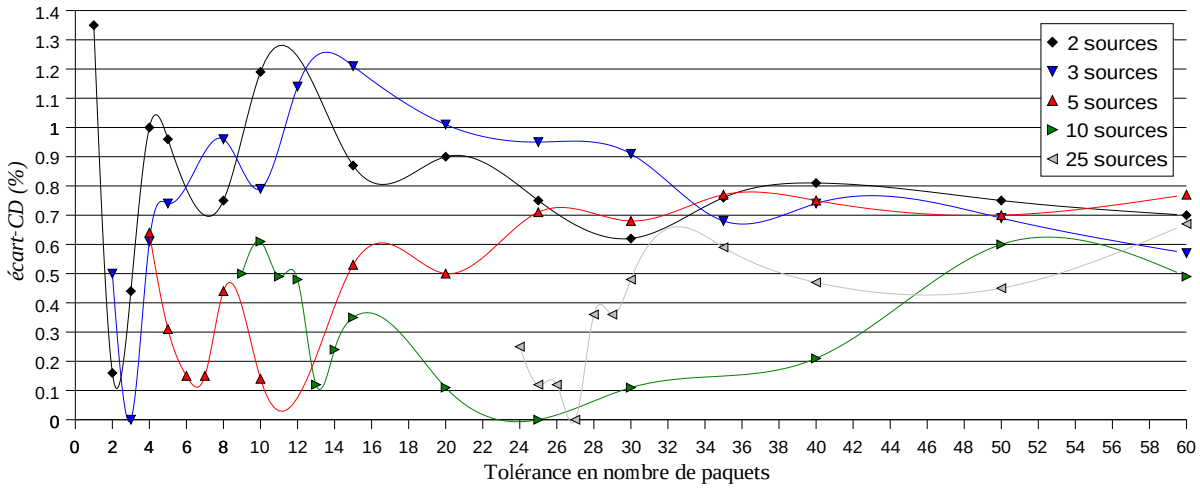

FIG. 3.11 – *Influence de la réactivité,* $\alpha = 60$ *et* $\beta = 2$.

3.5 Conclusion

Bilan. Dans ce chapitre, nous avons étudié la cohérence entre les modèles discret et continu. Cette étude s'appuie sur la comparaison de simulations discrètes et continues d'un protocole de transport simplifié dans différentes situations. Les résultats de ces simulations montrent que la modélisation continue est pertinente pour une grande plage de valeurs des différents paramètres du réseau et du protocole.

- **Bande passante**. Le rapport entre la bande passante sortante/entrante du nœud de congestion est un paramètre important. Plus ce rapport est grand (*i.e.*, moins le goulot d'étranglement est serré) moins l'écart entre les deux modèles est important. Inversement, lorsque le rapport de bande passante est faible, et que de fortes variations de débit sont imposées (*i.e.*, α grand et β petit), les deux modèles ont des comportements assez différents.
- **Délais de propagation**. Les délais de propagation (s'ils sont réalistes) n'ont pas une grande influence sur les résultats obtenus en discret et en continu. En effet, lorsque le RTT de la connexion est inférieur à 1,5 s, les différences constatées entre les modèles discret et continu sont négligeables (*écart-CD* $< 1\%$).
- **Type de variation du débit**. La rapidité de la variation du débit est un paramètre important. Pour garantir une équivalence entre les deux modèles, il faut que les variations de débit soient douces. Cependant, les fortes variations de débit sont mieux acceptées à mesure que le nombre de sources augmente.
- **Réactivité**. La réactivité du protocole ne semble pas avoir une grande influence sur la cohérence entre les modèles continu et discret. Notons que lorsque le rapport bande passante sortante/entrante est faible, une faible réactivité (*i.e.*, une forte tolérance T) peut réduire l'écart entre les modélisations continue et discrète.
- **Taille des paquets**. La taille des paquets n'a pas une influence significative si ces derniers ont une taille réaliste ($s < 10$ ko).

Perspectives. Ces résultats ont été obtenus avec un protocole simple de type AIMD, mais nous pensons qu'ils peuvent être extrapolés à d'autres protocoles de transport de type AIMD. Cette étude a permis de déterminer certaines limites autorisant une utilisation pertinente de la modélisation continue de réseaux à commutation de paquets.

Notons que l'intérêt principal de la modélisation continue réside dans la simplification de l'étude des réseaux qu'elle apporte. En effet, un certain nombre de problèmes de réseaux à commutation de paquets (comme le contrôle de congestion) peut être réduit à une étude de systèmes dynamiques (en boucle fermée ou non) équivalents. La cohérence des résultats obtenus dans les domaines du continu et du discret devrait permettre d'élargir la piste des recherches visant à améliorer le contrôle de congestion. En effet, les résultats obtenus en continu [83, 118, 22] pouvant être transposés en discret sans que les comportements (observés en continu) ne se détériorent, de nouvelles perspectives s'ouvrent pour le contrôle de congestion dans les protocoles de transport.

L'approximation continue devrait également rendre possible l'exécution de simulations de plus en plus complexes. En effet, dans [67], les auteurs montrent que lorsque le nombre de paquets générés devient important (*i.e.*, nombre de nœuds et de sources élevé), les simulations utilisant un modèle continu sont plus performantes que celles effectuées en discret. La modélisation continue devrait donc autoriser des simulations à large échelle, qui sont actuellement irréalisables (en terme de puissance de calcul nécessaire) avec des simulateurs à événements discrets tel que *ns*.

Chapitre 4

Le protocole Primo : Proportionnel intégral modifié

Dans ce chapitre, un mécanisme de contrôle de congestion basé sur un régulateur provenant de la théorie du contrôle est développé. Pour commencer, le type de correcteur ainsi que les variables de contrôle à utiliser seront choisis (*cf.* section 4.1). Ensuite (dans les sections 4.2 à 4.5), le correcteur de débit d'émission est construit pas à pas et validé par des simulations dans le domaine du continu sous *matlab*. Une fois le correcteur construit et validé, il sera intégré à un protocole de transport (dans la section 4.6) intrinsèquement discret sous l'hypothèse de cohérence des modèles continu et discret [21] (*cf.* chapitre 3).

Positionnement. Comme nous l'avons vu dans le chapitre 2, les améliorations apportées à TCP lui permettent d'ajuster son débit d'émission afin d'éviter les congestions tout en utilisant au mieux les ressources disponibles sur le réseau. Dans TCP, les informations sur l'état du réseau sont fournies par les acquittements qui permettent :

- de déterminer si les segments émis ont bien été reçus,
- de calculer le temps d'aller retour des paquets (le RTT),
- de calculer le chien de garde de retransmission (le RTO).

La détection d'une perte de segments entraîne sa ré-émission et la réduction du débit d'émission.

Comme nous l'avons vu dans le chapitre 3, une telle connexion peut être modélisée par un système dynamique à retard avec un délai de retour incertain. Les sorties du système sont véhiculées (par les acquittements) jusqu'au contrôleur (la source) qui les utilise pour ajuster l'entrée du système (le débit d'émission). Dans le cas du contrôle de congestion, le réseau est vu comme une « boîte noire » qui ne peut fournir à la source aucune information explicite sur son état de congestion. Il est donc exclu de concevoir un système en boucle fermée *explicite*. En revanche, les pertes de paquets, le débit de réception ainsi que la durée du RTT fournissent des informations implicites sur l'état du réseau et permettent donc de construire un modèle en boucle fermée *implicite*. En utilisant le langage de la théorie du contrôle des systèmes [71], le contrôle de congestion peut être vu, d'une certaine manière, comme un problème de *stabilité* du système en boucle fermée.

Objectifs. La construction d'un correcteur simple (issu des techniques de contrôle) ainsi que sa validation permettront de privilégier certaines pistes et de les approfondir afin d'améliorer les techniques de contrôle de congestion. Le correcteur ainsi créé devra répondre aux spécifications suivantes :

- **Rapidité.** Le correcteur doit permettre à la source d'atteindre un débit optimal (utilisation complète de la bande passante disponible) en un minimum de temps. Cette caractéristique permet de garantir que les ressources du réseau seront utilisées au mieux.
- **Prévention.** Le correcteur doit avoir un comportement préventif), ce qui signifie que le débit des sources doit être réduit avant que les files d'attente des routeurs ne soient saturées. Cette caractéristique permet de réduire le nombre de pertes de paquets et donc de retransmissions (entraînant une sous-utilisation du réseau).
- **Constance du débit.** Le correcteur doit permettre à la source d'émettre à un débit variant le moins possible lorsque le réseau se trouve en régime permanent (la charge du réseau est stabilisée). En effet, les oscillations du débit d'émission peuvent engendrer des variations du débit de réception et donc affecter la qualité de réception dans le cas de la transmission d'un flot temps réel. De plus, les oscillations du débit d'émission nuisent à la stabilité du réseau.
- **Réaction aux perturbations.** Une fois le régime permanent atteint, le correcteur doit être capable de réagir à des perturbations (*i.e.*, arrivée d'une nouvelle connexion). Cette réaction doit être assez rapide (de manière à ne pas saturer le réseau) et adaptée à la perturbation (afin d'éviter des réductions de débit inutiles). Autrement dit, le correcteur doit permettre aux sources d'adapter leur débit à la charge du réseau.

Ces spécifications donneront lieu à la création d'un correcteur multi-critère. Un tel correcteur a deux objectifs globaux : optimiser l'utilisation des ressources du réseau (rapidité, réaction aux pertubations, prévention), et accroître la qualité de réception du destinataire (rapidité, stabilité, réaction aux perturbations).

4.1 Principe d'utilisation du correcteur

4.1.1 Correcteur

Choix du correcteur. Il existe de nombreuses architectures de commande qui sont plus ou moins complexes et performantes. Néanmoins, quatre grandes familles de correcteurs classiquement utilisés en automatique [26, 41] peuvent être distinguées : Proportionnel, Proportionnel Dérivé, Proportionnel Intégral, Proportionnel Intégral Dérivé. Chacun de ces correcteurs utilise une ou plusieurs des actions suivantes :

- **Action « proportionnelle ».** Elle permet de corriger l'entrée du système proportionnellement à l'erreur constatée entre la consigne et la valeur d'entrée du système. Dans le cas du contrôle de congestion, le RTT peut être utilisé comme information sur l'état du réseau (variable de sortie). Une fois le RTT consigne estimé (RTT pour lequel le réseau n'est pas congestionné), l'erreur constatée entre le RTT instantané et la consigne permettra de définir l'état de congestion du réseau. Elle permettra également d'adapter le débit de la source proportionnellement à l'erreur. L'action « proportionnelle » semble bien adaptée

aux objectifs fixés car la source doit plus ou moins réduire son débit en fonction de la gravité de la congestion.

– **Action « dérivée »**. Elle permet d'anticiper les variations du système en se basant sur le sens et la rapidité de variation de la sortie. Cette composante permet d'améliorer le temps de réponse d'un système lors d'un changement de consigne ou lors de l'apparition d'une perturbation. L'inconvénient de cette composante est qu'elle engendre fréquemment des dépassements de consigne et donc des oscillations de la variable d'entrée (le débit d'émission dans le cadre du contrôle de congestion). Les oscillations du débit d'émission pouvant entraîner l'instabilité du réseau, la composante « dérivée » ne sera pas utilisée.
– **Action « intégrale »**. Elle permet de tenir compte de l'historique des variables de sortie du système. Cette composante permet d'améliorer la stabilité du système au détriment de la rapidité de réaction. Dans le cas du contrôle de congestion, la composante « intégrale » permettra à la source d'émettre avec un débit plus lissé : les erreurs (écart entre le RTT instantané et la consigne) de courte durée auront moins d'influence sur le débit d'émission. Cette action semble bien adaptée au contrôle de congestion, car elle permet d'accroître la stabilité du système.

Les deux composantes « proportionnelle » et « intégrale » semblent adaptées au contrôle de congestion. En effet, la composante « dérivée » risquant d'introduire de l'instabilité et ayant une sensibilité accrue aux erreurs de modélisation (surtout lorsqu'il y a des retards), elle ne sera pas utilisée. Le correcteur construit sera donc de type PI (Proportionnel Intégral). Dans un premier temps, seule l'action « proportionnelle » sera mise en œuvre, puis l'action « intégrale » sera ajoutée de manière à améliorer les performances obtenues.

4.1.2 Variables

Le choix des variables pour construire un correcteur PI dans le cadre du contrôle de congestion est assez restreint. En effet, il n'existe qu'une seule variable de commande, le débit d'émission et trois variables de sortie, le marquage des paquets en cas de congestion, le délai d'acheminement, et le débit de réception.

Description de ces variables :

– **Débit d'émission**. Pour la source, le seul moyen d'agir sur une congestion est d'ajuster son débit d'émission. En pratique, ce débit peut être ajusté de deux manières, en jouant sur :
  – la taille d'une fenêtre d'émission (à la manière de TCP)
  – la durée du délai inter-paquets (à la manière de RAP ou de TFRC).
– **Marquage en cas de congestion**. L'information apportée par l'option ECN est incompatible avec l'utilisation d'un correcteur proportionnel car elle ne permet pas de « quantifier » l'importance de la congestion.
  En effet, l'option ECN étant intrinsèquement discrète, elle permet uniquement de spécifier si le réseau est congestionné ou non sans en indiquer l'état précis. Cette variable ne sera donc pas utilisée pour la construction du correcteur PI.

– **Délai d'acheminement**. Le RTT est généralement utilisé pour représenter le délais d'acheminement des données. De cette variable, on peut déduire l'état de congestion du réseau ; plus ce délai est important (*i.e.*, plus les files d'attente des routeurs sont pleines), plus le réseau est congestionné.

  Malheureusement, le RTT ne permet pas de différencier les congestions se trouvant dans le sens retour (chemin emprunté par les acquittements) de celles se trouvant dans le sens aller (partie du chemin sur laquelle la source peut avoir une influence). La source ne pouvant agir (en ajustant son débit d'émission) que sur les files d'attente du chemin dans le sens aller, elle a besoin de pouvoir différencier les congestions se trouvant dans le sens aller de celles se trouvant dans le sens retour.

  À la manière du ROTT de PASTRA (*cf.* paragraphe 2.4.3), nous utiliserons une variable dérivée du RTT : le FTT (*Forward Trip Time*). Cette variable représente le temps que met un paquet pour aller de la source à la destination et est mesurée par le récepteur. Le FTT permet donc d'estimer l'état de congestion du chemin emprunté pour aller de l'émetteur au récepteur et ainsi de s'affranchir des erreurs d'estimation dues au sens retour du chemin.

  Pour mesurer le FTT, il faut que la source indique dans les paquets leur heure d'émission. À la réception de ces paquets, la machine destinataire soustrait l'heure de réception à celle d'émission pour obtenir le FTT. La source récupérera la valeur du FTT *via* les acquittements.

– **Débit de réception.** Comme le FTT, le débit de réception est calculé par le récepteur et fourni à la source *via* les acquittements. De cette variable, on peut déduire l'état de congestion du chemin emprunté dans le sens aller. En effet, plus le débit de réception diminue (*i.e.*, le nombre de paquets de différentes connexions contenus dans les files d'attente augmente), plus le réseau est congestionné.

Le marquage des paquets n'étant pas compatible avec l'action « proportionnelle », les variables de sorties utilisées pour la construction du correcteur seront le délai d'acheminement (*i.e.*, le FTT) et le débit de réception.

Remarque 1 *Les horloges des deux machines (source et destinataire) n'étant pas synchronisées, le FTT ne peut pas être mesuré de manière exacte. Cette imprécision ne pose pas de problème car les correcteurs se basent sur les variations et non sur les valeurs. En revanche, si les horloges n'ont pas la même précision, la mesure du FTT risque de croître ou de décroître (suivant que l'horloge du récepteur est plus rapide ou moins rapide que celle de l'émetteur) monotonement. Mais cette dérive des horloges est négligeable dans la plupart des cas. Notons tout de même qu'elle peut être calculée et compensée grâce à un algorithme de type ESRS [114].*

4.1.3 Principe de fonctionnement

À l'initialisation de la connexion, l'émetteur commence par envoyer deux paquets avec un délai inter-paquets nul (*i.e.*, débit infini).

Lorsque le récepteur reçoit le premier paquet, il calcule le FTT initial et le renvoie à l'émetteur *via* un acquittement. À l'initialisation, le réseau est supposé non congestionné, le FTT initial représente donc la plus petite valeur que pourra atteindre le FTT de cette connexion. Si,

après l'initialisation, un FTT inférieur au FTT initial venait à être mesuré (cas où le réseau était congestionné lors de l'initialisation), le FTT initial prendrait pour valeur cette dernière mesure.

Lorsque l'émetteur reçoit l'acquittement contenant le FTT initial, le FTT consigne (noté FTT-cons), est calculé. À la manière de TCP Vegas [18], un FTT consigne légèrement supérieur au FTT initial permettra de détecter tout accroissement de la bande passante disponible. En effet, supposons que deux connexions aient des FTTs consignes idéaux (de sorte que leurs paquets ne soient jamais stockés dans les files d'attente), si l'une de ces deux connexions s'arrête, le FTT instantané de l'autre restera le même et la bande passante libérée ne sera pas réutilisée. En revanche, si le FTT consigne avait été légèrement supérieur à sa valeur idéale (de sorte qu'au moins l'une des files du chemin emprunté ne soit pas complètement vide), à l'arrêt de l'une des deux connexions, le FTT instantané de l'autre aurait diminué. Or, une réduction du FTT instantané signale une diminution de la taille de l'une des files d'attente et donc une libération de la bande passante. Cette libération de bande passante doit donc entraîner l'augmentation du débit d'émission et une utilisation totale de la bande passante disponible.

À la réception du second paquet, le récepteur calcule le débit de réception (noté $y(t)$) de la manière suivante :

$$y(t) = \frac{\text{taille d'un paquet}}{\text{délai inter-paquets mesuré à la date } t} \tag{4.1}$$

Le calcul initial du débit de réception permet d'estimer quelle est la bande passante initialement disponible. En effet, la mesure du délai séparant la réception des deux premiers paquets reflète le temps qu'ils ont passé dans les files d'attente (lié au temps de mise sur le réseau ou au taux de remplissage des files) et donc la bande passante disponible.

À la réception du premier acquittement (contenant le FTT initial ainsi que la première mesure du débit de réception), la source calcule le FTT consigne et commence à émettre des données. Les données sont émises avec un débit d'émission égal au débit de réception contenu dans l'acquittement (noté y-reçu$_t$). L'emploi du débit de réception comme débit d'émission permet d'utiliser immédiatement toute la bande passante disponible.

Une fois la phase d'initialisation terminée, chaque réception d'un paquet de données entraîne l'émission d'un acquittement contenant la dernière mesure du FTT et du débit de réception. Ces valeurs contenues dans l'acquittement sont ensuite utilisées par l'émetteur qui régulera son débit en fonction de l'état de congestion du réseau (*i.e.*, du FTT instantané noté FTT-reçu$_t$ et du FTT consigne).

4.2 Construction d'un correcteur proportionnel basé sur le FTT

Dans cette section, un correcteur proportionnel basé uniquement sur le FTT est construit (l'introduction de la composante « intégrale » se fera ultérieurement *cf.* section 4.5).

4.2.1 Principe

Dans cette première version du correcteur, le calcul du débit d'émission se fait uniquement grâce à l'erreur constatée entre le FTT instantané et le FTT consigne. Ce correcteur proportion-

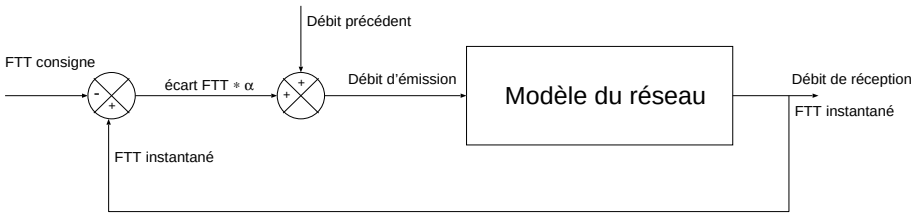

FIG. 4.1 – *Système en boucle fermée implicite intégrant un correcteur proportionnel.*

nel peut être représenté très simplement (*cf.* figure 4.1). Dans la figure 4.1, le *gain* α représente le coefficient de proportionnalité (qui est constant). Plus ce gain sera élevé (en valeur absolue), plus les réactions aux variations du FTT engendreront de forts ajustements du débit d'émission. Le débit d'émission est donc obtenu de la manière suivante :

$$\text{Débit d'émission} = \text{Débit précédent} + \alpha \times (\text{FTT instantané} - \text{FTT consigne}) \times K \quad (4.2)$$

Le paramètre K représente une constante exprimée en kbit/s^2. Cette constante est obtenue en divisant le débit de réception initial par le FTT initial et elle permet d'obtenir une équation homogène. Le « Débit précédent » représente le débit qu'avait la source un RTT plus tôt. L'utilisation du « Débit précédent » permet de corriger le débit ayant entraîné la mesure du FTT instantané (et donc l'estimation de l'état de congestion du réseau). Dans le cas où une congestion est détectée, une des idées est d'adapter le débit d'émission en fonction du débit ayant créé cette congestion (débit de la source un RTT plus tôt). De manière plus formelle, l'équation obtenue est la suivante :

$$x(t) = x(t - \text{RTT}_t) + K \times \alpha \times (\text{FTT-reçu}_t - \text{FTT-cons}) \quad (4.3)$$

4.2.2 Évaluation

Pour étudier les performances de ce correcteur, des simulations utilisant la topologie présentée figure 4.2 ont été effectuées. Afin de simplifier l'étude, dans le cadre de simulations sous *matlab* (en continu), les files d'attente seront considérées de taille infinie.

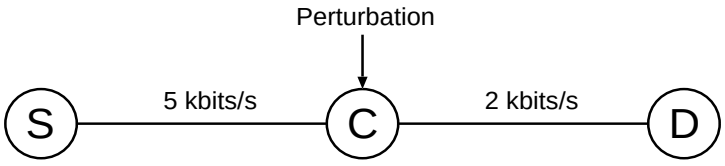

FIG. 4.2 – *Topologie utilisée pour les simulations avec S : nœud source, C : nœud de congestion, D : nœud destinataire.*

Simulations sans perturbation. Dans ces premières simulations, la perturbation arrivant sur le nœud congestionné est nulle. En utilisant un FTT consigne dépassant de 3% le FTT initial et un gain α de -1, les résultats obtenus sont particulièrement mauvais (*cf.* figure 4.3). En effet, le

débit d'émission devrait théoriquement se stabiliser autour de 2 kbits/s. Or, dans cette simulation, il oscille très fortement entre 0 kbit/s et 4 kbits/s. Ces oscillations sont dues à la consigne qui a une valeur trop élevée (dépassement de 3% du FTT initial) ainsi qu'au gain qui est trop grand.

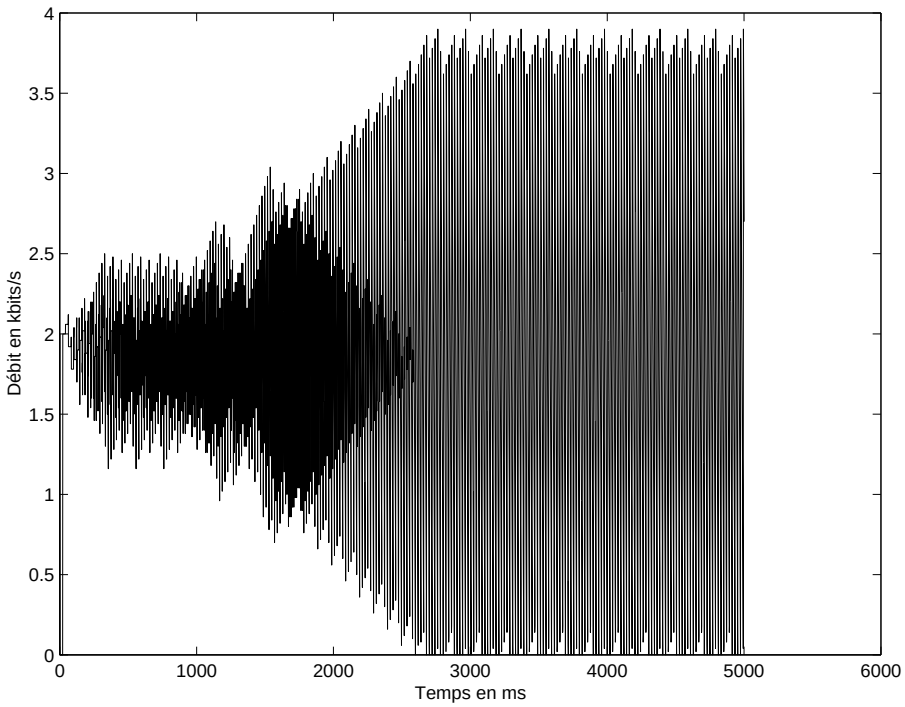

FIG. 4.3 – *Première simulation utilisant un correcteur proportionnel avec une consigne ayant un dépassement de 3% et un gain de -1.*

En utilisant un FTT consigne ayant la même valeur que le FTT initial (soit 0% de dépassement), le débit d'émission est totalement stable et vaut 2 kbits/s tout au long de la simulation (*cf.* figure 4.4). L'utilisation d'un FTT consigne sans dépassement va entraîner des problèmes de détection de bande passante disponible. Il faudra donc employer une autre technique pour découvrir les libérations de bande passante.

Simulation avec perturbation. L'introduction d'une perturbation constante de 1 kbit/s permet de tester facilement les capacités d'adaptation du correcteur. Normalement, l'introduction d'une telle perturbation devrait conduire la source à réduire son débit puis à le stabiliser à 1 kbit/s. Les résultats obtenus en utilisant un gain assez petit pour éviter de trop fortes oscillations montrent que le correcteur ne parvient pas à stabiliser le débit d'émission. En effet, le débit d'émission oscille autour de 1 kbit/s (*cf.* figure 4.5). Mais ces oscillations ont une amplitude de plus en plus importante au cours du temps. Ce phénomène rend le correcteur inefficace lorsque le réseau reste stable suffisamment longtemps. Aux vues de ces premières simulations, le nombre de paramètres du correcteur est insuffisant pour pouvoir contrôler efficacement le débit d'émission.

Cette première série de simulations nous a montré qu'il était impératif d'utiliser un FTT consigne égale au FTT initiale. Cette contrainte impose la définition d'une nouvelle méthode

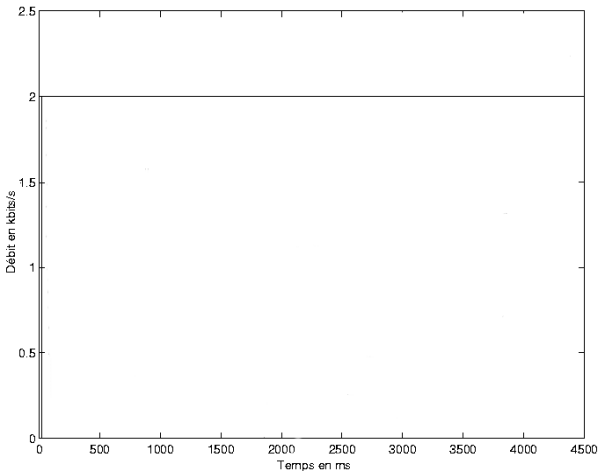

FIG. 4.4 – *Simulation utilisant un correcteur proportionnel avec une consigne sans dépassement et un gain de -1.*

permettant de détecter les augmentations de bande passante disponible (*i.e.*, libération de bande passante à l'arrêt d'une connexion). Ces simulations ont également montré que l'utilisation du FTT comme seule variable de contrôle était insuffisante. En effet, avec ce correcteur, il est impossible de stabiliser le débit d'émission lorsqu'une perturbation apparaît.

4.3 Introduction du débit de réception comme variable de contrôle

Dans cette section, le correcteur précédent est complété par l'ajout d'une seconde variable de contrôle : le débit de réception.

4.3.1 Principe

Lorsque le FTT instantané devient supérieur au FTT consigne, cela signifie qu'au moins l'une des files d'attente du chemin emprunté n'est pas vide. Or, si au moins une file d'attente contient des paquets, le débit de réception représente la bande passante disponible pour la connexion (*i.e.*, le débit auquel les paquets de la connexion sortent de la file non vide). Le débit d'émission de la source pourrait donc prendre la valeur du débit de réception, ce qui permettrait d'utiliser au mieux la bande passante disponible sans saturer le réseau. L'introduction du débit de réception pour calculer le débit d'émission en phase de congestion semble donc judicieuse.

Dans cette évolution du correcteur, l'écart entre le FTT instantané et la consigne (l'erreur) sera toujours utilisé pour estimer l'état de congestion. Dans le cas où l'erreur est positive (*i.e.*, une congestion est en train de se former), le débit d'émission sera calculé grâce à l'erreur constatée entre le débit de réception et le débit d'émission un RTT plus tôt. En cas de congestion, le débit d'émission sera donc obtenu de la manière suivante :

$$\text{Débit d'émission} = \text{Débit précédent} + \alpha_{\text{débit}} \times (\text{Débit précédent} - \text{Débit réception}) \quad (4.4)$$

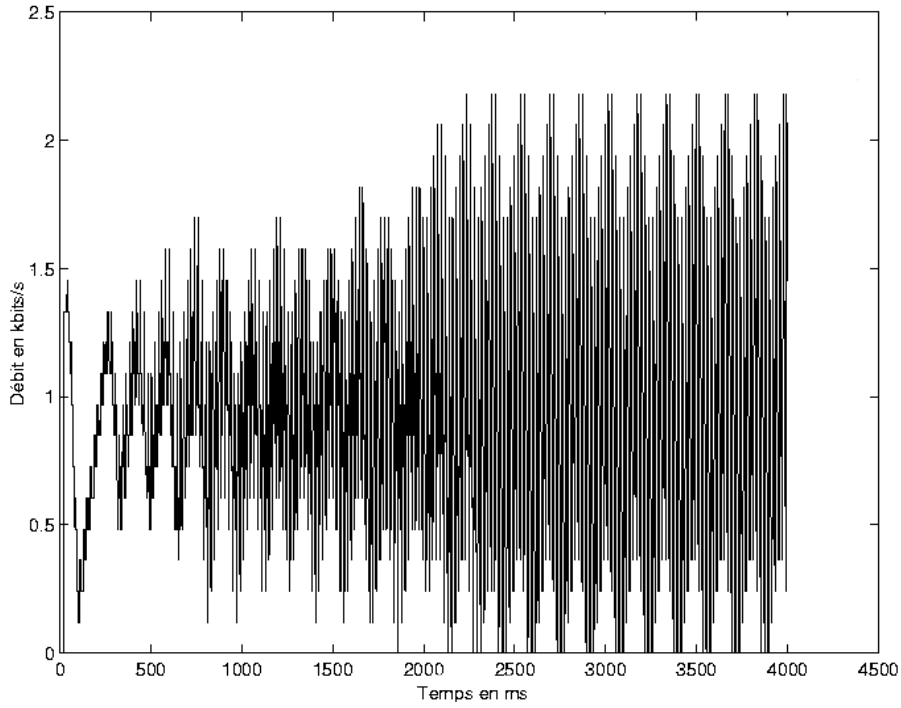

FIG. 4.5 – *Simulation avec une perturbation de 1 kbit/s et un correcteur proportionnel ayant une consigne sans dépassement et un gain de -1.*

Le *gain* $\alpha_{\text{débit}}$ permet de faire converger plus ou moins rapidement le débit d'émission vers le débit de réception. De manière plus formelle, l'équation obtenue est la suivante :

$$x(t) = x(t - RTT_t) + \alpha_{débit} \times (x(t - RTT_t) - y\text{-}reçu_t) \tag{4.5}$$

Dans le cas où le réseau n'est pas congestionné (*i.e.*, l'erreur entre le FTT instantané et la consigne est nulle), le débit sera calculé comme dans le correcteur précédent (*cf.* équations 4.2 et 4.3).

4.3.2 Évaluation

Afin d'étudier l'impact de l'introduction du débit de réception dans le calcul du débit d'émission, des simulations ont été effectuées dans le même contexte que précédemment : une perturbation constante de 1 kbit/s commence et arrête d'émettre en même temps que la source utilisant le correcteur.

Les résultats obtenus (*cf.* figure 4.6) avec ce correcteur semblent au premier abord excellents : le débit de la source se stabilise rapidement à 1 kbit/s.

Cependant, on peut remarquer que le débit de la source n'est jamais inférieur à 1 kbit/s, ce qui signifie que la file ne se videra jamais. En effet, au début de la connexion, la source émet à 1,3 kbits/s et la perturbation émet à 1 kbit/s, le débit du lien congestionné étant de 2 kbits/s et la file d'attente se remplit. Lorsque le débit de la source atteint 1 kbit/s, la taille de la file d'attente cesse d'augmenter, mais ne diminue pas car la somme des débits entrants au nœud congestionné est équivalente au débit de sortie (2 kbits/s). Pour que la file puisse se vider, il faudrait que la

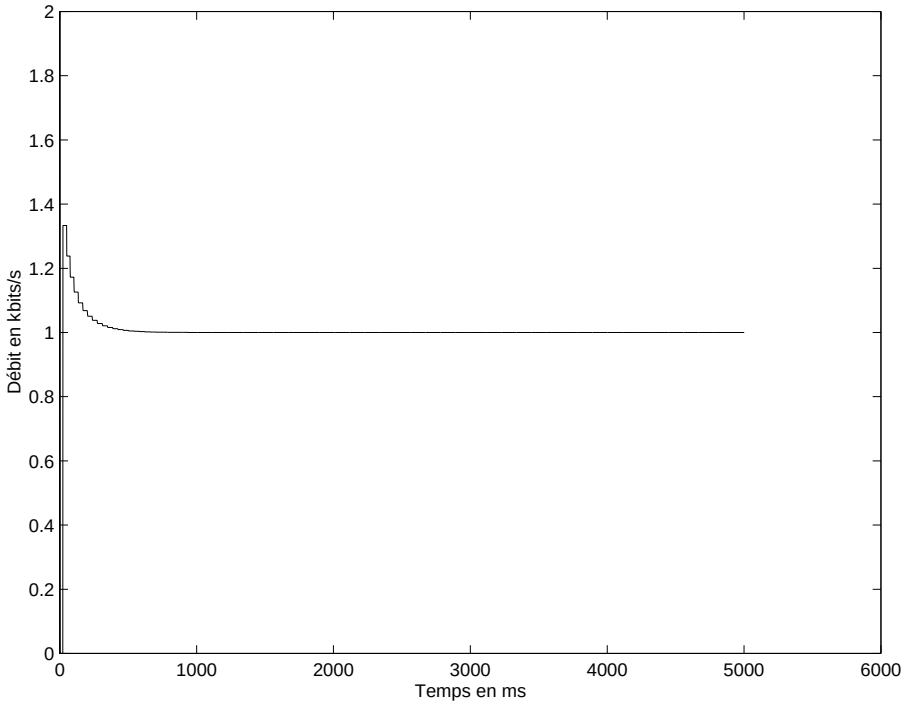

FIG. 4.6 – *Simulation utilisant le correcteur proportionnel introduisant le débit de réception dans le calcul du débit.*

somme des débits entrant soit inférieure au débit de sortie du nœud congestionné (*i.e.*, que le débit de la source soit inférieur à 1 kbit/s).

Le fait de ne jamais vider la file d'attente pose des problèmes évidents de saturation du réseau. À chaque arrivée d'une nouvelle connexion, la taille de la file va augmenter avant de se stabiliser ou de saturer. Cette situation n'est donc pas envisageable dans un environnement réel où les files d'attente ont une taille finie. L'ajout du débit de réception permet de stabiliser le débit de la source lorsqu'une perturbation apparaît, mais le correcteur doit encore être modifié afin de répondre aux contraintes d'un environnement réel (*i.e.*, files d'attente de taille finie).

4.4 Réduction de la taille de la file d'attente après une congestion

Dans cette section, nous proposons une solution au problème du remplissage de la file d'attente.

4.4.1 Principe

Pour résoudre le problème du remplissage de la file d'attente, il suffit de faire converger le débit d'émission vers une valeur légèrement inférieure à la bande passante disponible.

Sachant que lors des phases de congestion, le débit de réception représente la bande passante disponible, il suffit que la source ait un débit d'émission inférieur à celui de réception pour que

la file d'attente se vide. Une fois la file d'attente vidée, le débit d'émission doit augmenter de manière à utiliser toute la bande passante disponible. Pour cela, le débit d'émission doit converger vers une valeur légèrement supérieure au débit de réception.

Notons que cette augmentation du débit d'émission permet également de résoudre le problème de détection de bande passante nouvellement disponible engendré par l'utilisation d'un FTT consigne sans dépassement (*cf.* section 4.2).

L'augmentation du débit d'émission risque d'entraîner la création de « petites congestions » (*i.e.*, léger dépassement du FTT consigne) et de nouvelles réductions du débit d'émission. Pour éviter que les oscillations du débit d'émission ne soient trop fortes, l'importance de la congestion doit être prise en compte lors de la réduction du débit.

À la manière de TCP Vegas, pour maintenir le débit d'émission légèrement supérieur à la bande passante disponible, trois bornes ($\text{erreur}_{\text{inf}}$, $\text{erreur}_{\text{sup1}}$ et $\text{erreur}_{\text{sup2}}$) relatives à l'erreur entre le FTT instantané et le FTT consigne sont définies. Ces trois bornes sont toutes supérieures à zéro. Partant de là, on peut distinguer quatre cas :

- Lorsque l'erreur constatée entre le FTT consigne et le FTT instantané est inférieure à la borne $\text{erreur}_{\text{inf}}$, le débit d'émission augmentera.
- Inversement, si l'erreur constatée entre le FTT consigne et le FTT instantané est comprise entre les bornes $\text{erreur}_{\text{sup1}}$ et $\text{erreur}_{\text{sup2}}$ (*i.e.*, légère congestion), le débit d'émission sera légèrement réduit.
- De même, si l'erreur constatée est supérieure à la borne $\text{erreur}_{\text{sup2}}$ (*i.e.*, forte congestion), le débit sera fortement réduit.
- Enfin, si l'erreur constatée est comprise entre les bornes $\text{erreur}_{\text{inf}}$ et $\text{erreur}_{\text{sup1}}$, le débit d'émission restera inchangé.

Ces quatre cas sont résumés d'un manière formelle dans le système d'équations suivant :

$$\text{adaptation débit} = \begin{cases} \text{Débit précédent} - \text{coefficient de forte réduction} \times \text{Débit réception} & \text{Si } cond1 \\ \text{Débit précédent} - \text{coefficient de faible réduction} \times \text{Débit réception} & \text{Si } cond2 \\ \text{Débit précédent} - \text{coefficient d'augmentation} \times \text{Débit réception} & \text{Si } cond3 \\ 0 & \text{Sinon} \end{cases}$$

Avec :
$cond1$: Forte congestion
$cond2$: Faible congestion
$cond3$: Risque de sous utilisation du réseau

$$\text{Débit} = \text{Débit précédent} + \alpha_{\text{débit}} \times \text{adaptation débit} \tag{4.6}$$

Le « degré de congestion » du réseau permet de distinguer le cas où la congestion constatée est due à une recherche de bande passante disponible par l'une des sources (*i.e.*, faible congestion) du cas où elle est due à une réelle surcharge du réseau (*i.e.*, forte congestion). Les valeurs des bornes $\text{erreur}_{\text{sup1}}$ et $\text{erreur}_{\text{sup2}}$ permettent de définir à partir de quels seuils on considère qu'une congestion est faible ou forte. La valeur de ces seuils conditionne le degré de prévention du correcteur. En effet, plus ces seuils sont bas plus le correcteur est préventif.

Dans le système d'équations précédent, les coefficients de faible et de forte réductions sont des constantes comprises entre 0 et 1. Ces deux constantes permettent de faire converger le débit d'émission vers une valeur inférieure au débit de réception.

Le coefficient de faible réduction ($R1$) est supérieur à celui de forte réduction ($R2$), de sorte que la réduction de débit ne soit pas trop forte dans le cas d'une faible congestion (*i.e.*, dépassement de erreur$_{\text{sup1}}$ mais pas de erreur$_{\text{sup2}}$) et qu'elle soit importante dans le cas d'une forte congestion (*i.e.*, dépassement erreur$_{\text{sup2}}$). Le coefficient d'augmentation ($A1$) est une constante supérieure à 1 qui permet d'augmenter le débit d'émission lorsque les files d'attente du réseau sont vides.

Le système d'équations précédent s'écrit plus explicitement de la manière suivant :

$$\Delta = \begin{cases} x(t - \text{RTT}_t) - R2 \times \text{y-reçu}_t & \text{Si } cond1 \\ x(t - \text{RTT}_t) - R1 \times \text{y-reçu}_t & \text{Si } cond2 \\ x(t - \text{RTT}_t) - A1 \times \text{y-reçu}_t & \text{Si } cond3 \\ 0 & \text{Sinon} \end{cases}$$
$$\text{Avec :}$$
$$cond1 : (\text{FTT-reçu}_t - \text{FTT-cons}) \geq \text{erreur}_{\text{sup2}} \qquad (4.7)$$
$$cond2 : \text{erreur}_{\text{sup1}} < (\text{FTT-reçu}_t - \text{FTT-cons}) < \text{erreur}_{\text{sup2}}$$
$$cond3 : (\text{FTT-reçu}_t - \text{FTT-cons}) \leq \text{erreur}_{\text{inf}}$$

$$x(t) = x(t - \text{RTT}_t) + \alpha_{\text{débit}} \times \Delta$$

Dans cette version du correcteur, l'erreur entre le FTT consigne et le FTT instantané n'est plus utilisée dans le calcul du débit d'émission. Cette variable sert uniquement à l'estimation de l'état de congestion du réseau.

4.4.2 Évaluation

Le contexte de simulation précédent ne permettant pas d'évaluer la totalité des améliorations apportées au correcteur (*i.e.*, découverte de bande passante disponible), un nouveau contexte a donc été utilisé. Ce nouveau cas d'étude permet d'apprécier la réduction de la taille de la file d'attente ainsi que la détection de la bande passante disponible. Au lieu de démarrer et de s'arrêter en même temps que la source utilisant le correcteur proportionnel, la perturbation constante de 1 kbit/s démarre 1 s après la source (à t=1000) et s'arrête 2 s avant la source (à t=3000).

Les résultats obtenus (*cf.* figure 4.7) sont assez satisfaisants. La source réagit assez rapidement à l'apparition de la congestion sans trop réduire son débit. En effet, le débit de la source est légèrement inférieur à 1 kbit/s, ce qui permet à la file d'attente de se vider. Une fois la file d'attente vidée, la source tente d'augmenter son débit d'émission, ce qui entraîne la création d'une « petite congestion » immédiatement résorbée (*cf.* figure 4.7, oscillation du débit d'émission située juste avant t = 3000).

Le surcroît de bande passante disponible (à l'arrêt de la perturbation) est immédiatement détecté par la source et le débit augmente jusqu'à atteindre 2 kbits/s. De fortes oscillations sont visibles durant la période transitoire. En effet, lorsque la source cherche à utiliser pleinement la bande passante disponible, son débit d'émission varie fortement. En régime permanent (*i.e.*, lorsque le réseau est stable), de petites oscillations du débit d'émission sont observables. Ces oscillations sont dues aux tentatives d'augmentation de débit.

Les améliorations apportées au correcteur permettent de vider les files d'attente après une période de congestion et donc de répondre aux contraintes d'un environnement réel (*i.e.*, files

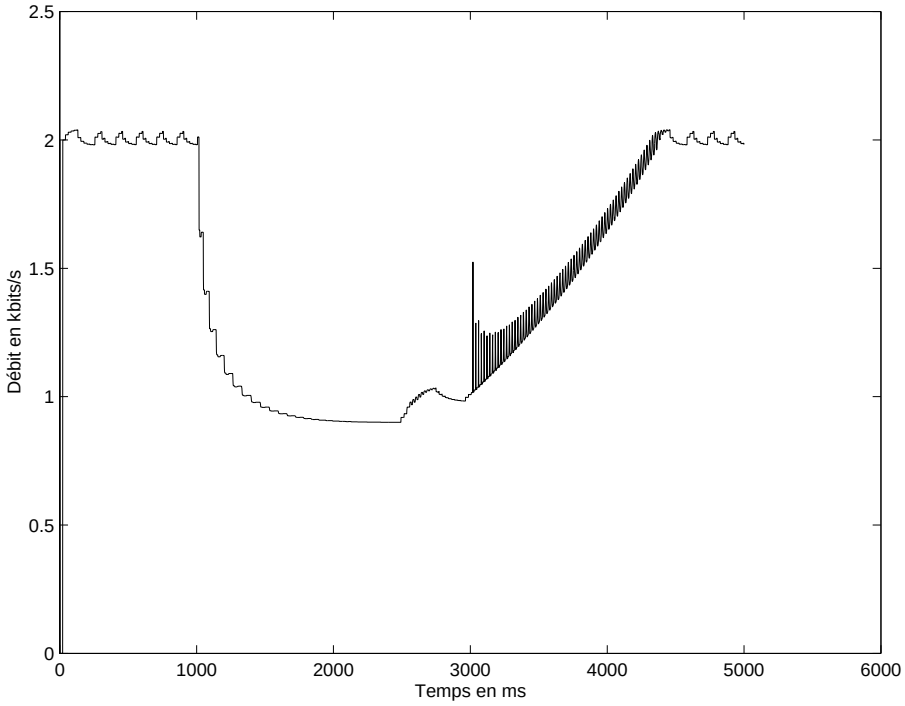

FIG. 4.7 – *Simulation utilisant le correcteur proportionnel prenant en compte la réduction de la taille de la file d'attente en cas de congestion.*

d'attente de taille finie). Ces améliorations ont également résolu le problème de détection de l'augmentation de la bande passante disponible. Cependant, afin de réduire les fortes oscillations constatées lors de la récupération de bande passante, le correcteur doit être encore modifié.

4.5 Introduction de la composante « intégrale »

Afin d'améliorer le lissage du débit d'émission, notamment lors des phases de récupération de bande passante, l'historique de la connexion sera utilisé par le correcteur. Dans ce sens, nous allons introduire l'action « intégrale ».

4.5.1 Principe

Comme précisé dans le paragraphe 4.1.1, l'action « intégrale » permet de prendre en compte l'historique de la connexion. Dans le cas du contrôle de congestion, le passé de l'état de congestion du réseau (*i.e.*, l'erreur constatée entre le FTT instantané et le FTT consigne) peut être utilisé afin de lisser le débit d'émission.

L'action « intégrale » s'appliquera donc à la mesure de l'erreur entre le FTT consigne et le FTT instantané. Les systèmes d'équations 4.6 et 4.7 restent donc les mêmes ; seul le test (FTT instantané - FTT consigne) utilisé pour connaître l'état de congestion du réseau sera modifié. Le FTT instantané utilisé pour effectuer ce test sera celui récupéré par la source il y a $\tau \times$ RTT secondes. La constante τ représente le *retard pur*, qui permet de prendre en compte les informations avec plus ou moins de retard. Le fait d'utiliser des informations avec un certain

retard permet à la source de réagir moins fortement aux variations de courtes durées. L'introduction de la composante « intégrale » devrait donc apporter de la stabilité au réseau.

4.5.2 Évaluations

Pour estimer l'amélioration apportée par l'ajout de la composante « intégrale », des simulations ont été effectuées dans le même cas d'étude que précédemment.

Les résultats obtenus (*cf.* figure 4.8) montrent que l'ajout de l'action « intégrale » a pour effet de lisser le débit d'émission. En effet, dans la simulation précédente (*cf.* figure 4.7), des oscillations du débit d'émission étaient présentes lors de la récupération de bande passante libérée. En utilisant la composante « intégrale », ces oscillations disparaissent. En revanche, les oscillations constatées, lorsque le débit est stabilisé, sont quasiment identiques que l'action « intégrale » soit utilisée ou non. L'action « intégrale » permet donc de lisser le débit dans certains cas sans toutefois résoudre tous les problèmes d'oscillations. Notons que les oscillations restantes sont très faibles et ne devraient donc pas perturber la stabilté du réseau.

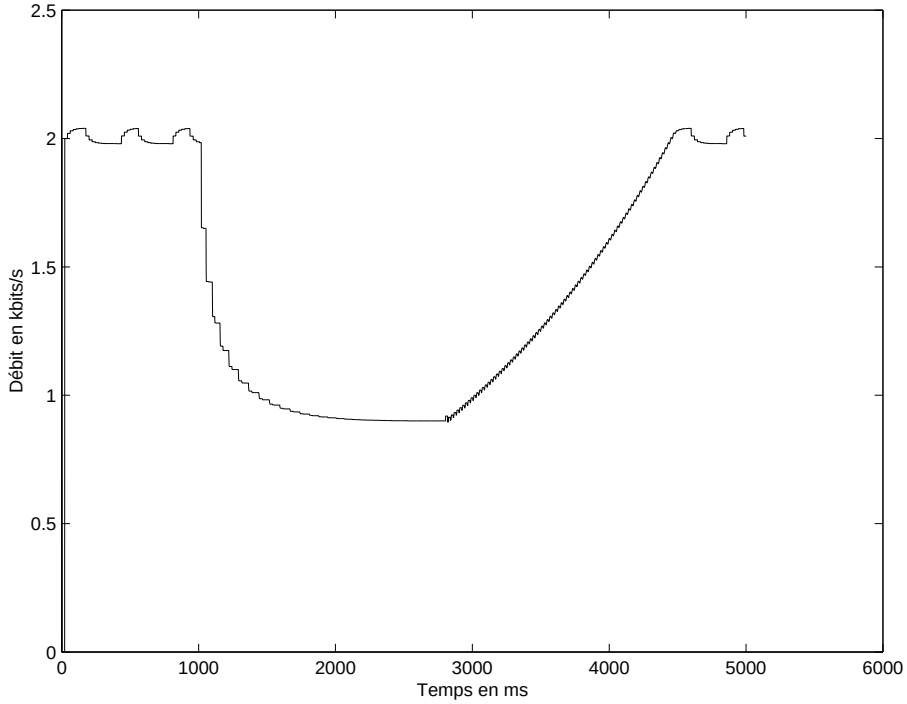

FIG. 4.8 – *Simulation utilisant le correcteur proportionnel intégral.*

Le code source (*matlab*) du correcteur obtenu est présenté dans l'annexe C.1.

4.6 Discrétisation

Dans cette section, nous décrivons les modifications apportées au correcteur continu afin de l'intégrer à un protocole de transport (dont le code source est disponible en annexe C.2).

Le principe de fonctionnement général reste le même. Seules quelques différences liées aux caractéristiques des réseaux à commutation de paquets sont notables.

La gestion du débit d'émission se fera grâce à un délai inter-paquets (dip). Lors de l'émission du $i^{\text{ème}}$ paquet, si le débit de la source est x_i, alors le délai séparant le $i^{\text{ème}}$ du $i+1^{\text{ème}}$ paquet (noté dip_i) sera calculé de la manière suivante :

$$dip_i = \frac{\text{taille du } i^{\text{ème}}}{x_i} \tag{4.8}$$

4.6.1 Initialisation

En continu, lors de l'initialisation, la source envoie deux paquets avec un délai inter-paquets nul afin d'estimer la bande passante disponible. Dans un réseau à commutation de paquets, il est risqué d'estimer la bande passante disponible en n'envoyant que deux paquets. En effet, il est possible que deux paquets d'un même flot (envoyés avec un délai inter-paquets nul) traverse toutes les files d'attente sans jamais qu'un seul paquet d'un autre flot (empruntant le même chemin) ne s'intercale entre eux. À la réception de ces deux paquets, le destinataire aura l'impression d'être seul sur le chemin emprunté. Il estimera donc une part de bande passante disponible égale à la bande passante du plus petit lien traversé. Cette surestimation de la bande passante disponible risque d'entraîner la saturation des files d'attente lors du passage en régime permanent (utilisation du débit de réception initial comme débit d'émission). Inversement, si plusieurs paquets d'un même flot s'intercalent entre les deux paquets initialement émis, le récepteur sous-estimera la bande bande passante disponible. Cette sous-estimation de la bande passante disponible risque d'entraîner une situation d'inéquité entre les flots.

Pour réduire les risques d'une mauvaise estimation de la bande passante disponible, l'émetteur enverra une rafale de quatre paquets avec un délai inter-paquets nul. Notons que le nombre de paquets envoyés dans cette rafale ne doit pas être trop important, car cela risquerait de saturer les files d'attente si plusieurs connexions démarraient simultanément.

Lorsque les quatre paquets de la rafale sont reçus, le récepteur calcule le débit de réception initial (*i.e.*, la bande passante disponible pour le flot). Le débit de réception initial représentera la moyenne des débits (*i.e.*, des délais inter-paquets) auxquels ont été reçus les quatre paquets. L'utilisation d'une moyenne permet de réduire les risques de sur et/ou sous-estimation de la bande passante disponible.

Pour estimer la bande passante disponible, le récepteur commence par calculer le temps écoulé entre la réception du premier et du dernier paquet de la rafale (noté temps-écoulé) :

$$\text{temps-écoulé} = \text{heure de réception du } 4^{\text{ème}} \text{ paquet} - \text{heure de réception du } 1^{\text{er}} \text{ paquet} \tag{4.9}$$

La bande passante disponible est ensuite calculée en divisant la quantité de données reçues entre la réception du premier et du dernier paquet (trois paquets) par le temps écoulé entre ces deux réceptions :

$$\text{Bande passante disponible} = \frac{3 \times \text{taille d'un paquet}}{\text{temps-écoulé}} \tag{4.10}$$

Une fois la bande passante disponible estimée par le récepteur, ce dernier envoie à l'émetteur un acquittement. Cet acquittement contient dans son en-tête la bande passante disponible ainsi que le FTT initial calculé lors de la réception du premier paquet. À la réception de cet acquittement, la source initialise le FTT consigne, ainsi que le débit d'émission avec les valeurs contenues dans l'en-tête.

La réception de cet acquittement permet également de calculer le RTT initial et ainsi d'initialiser le «retard» (RET) utilisé pour l'action « intégrale » (*cf.* section 4.5) :

$$\text{RET} = \tau \times \text{RTT initial} \times 1000$$

Comme nous l'avons vu précédemment, l'action « intégrale » a besoin de connaître la valeur qu'avait la variable de contrôle (*i.e.*, l'erreur constatée entre le FTT instantané et le FTT consigne) RET ms plus tôt. Un tableau permettant de stocker temporairement les « RET » dernières valeurs de l'erreur est donc créé.

Durant la phase d'initialisation, aucune perte n'est tolérée. Ainsi si l'un des paquets ou l'acquittement est perdu, cette phase sera intégralement relancée.

4.6.2 Régime permanent

Comme en continu, en régime permanent, la réception d'un paquet de données entraîne le calcul du débit de réception ainsi que celui du FTT. Ces deux informations sont ensuite transmises à la source *via* l'en-tête de l'acquittement correspondant au paquet reçu. Notons que tous les paquets reçus engendrent l'émission d'un acquittement (*i.e.*, pas d'acquittements cumulés).

À la réception de cet acquittement, la source calcule l'erreur constatée entre le FTT instantané (contenu dans l'acquittement) et le FTT consigne. La valeur ainsi obtenue est stockée dans le tableau conservant l'historique des « RET » dernières erreurs constatées.

À chaque émission d'un paquet de données, le débit d'émission de la source est ajusté grâce au correcteur précédemment défini. L'algorithme 2 décrit le comportement du correcteur discrétisé.

Dans cet algorithme, l'émetteur commence par récupérer la valeur de l'erreur constatée entre le FTT instantané et le FTT consigne il y a RET ms. Ensuite, comme en continu, l'erreur est comparée aux trois seuils ($n \times \text{erreur}_{\text{sup}}$, $\text{erreur}_{\text{sup}}$, $\text{erreur}_{\text{inf}}$), ce qui permet de déterminer le degré de congestion du réseau.

Une fois le degré de congestion déterminé, le nouveau débit d'émission est calculé proportionnellement à la différence constatée entre le débit de réception (récupéré dans le dernier acquittement reçu) et le débit d'émission qu'avait la source lors de l'envoi du paquet acquitté (*i.e.*, un RTT auparavant). Comme en continu, en fonction du degré de congestion du réseau, la différence mesurée entre le débit d'émission et de réception est plus ou moins accentuée grâce aux coefficients de réduction ($0 < \text{réduction\_1} < \text{réduction\_2} < 1$) ou d'augmentation ($1 < \text{augmentation\_1}$). L'importance de la variation du débit entre le nouveau débit calculé et celui qu'avait la source lors de l'émission du paquet acquitté est contrôlée par le coefficient α. Plus α est grand, plus les variations du nouveau débit pourront être importantes.

On peut noter qu'en cas de congestion, une fois le nouveau débit calculé, il est comparé à l'ancien débit (**si** nouveau débit $>$ débit **alors**..). Cette comparaison permet d'éviter d'augmenter le débit de la source alors que le réseau est congestionné. En effet, dans certains cas, il arrive

que le débit de réception mesuré soit supérieur au débit d'émission. Cela entraîne une augmentation à la place d'une réduction du débit. Ces mesures erronées du débit de réception sont dues à un effet de *Clustering* dans les files d'attente. Il peut arriver que des paquets envoyés avec un délai inter-paquets important se retrouvent placés de manière contiguë dans une file d'attente (*i.e.*, aucun paquet ne s'est intercalé entre eux et la file n'est pas vide). Si le débit du lien situé après cette file est supérieur au débit d'émission des paquets, leur délai inter-paquets à la sortie de la file sera inférieur à celui qu'ils avaient lors de leur émission. Cette réduction du délai inter-paquets engendrera une mesure erronée du débit de réception (qui sera plus important que celui d'émission). Ces cas de figure sont assez rares mais lorsqu'ils surviennent, les performances du contrôle de congestion sont fortement dégradées. De cette manière, lorsqu'une mesure erronée du débit de réception est détectée, elle est ignorée et le débit d'émission reste inchangé. Notons qu'inversement, une mesure erronée du débit de réception peut entraîner une réduction du débit d'émission alors qu'une libération de bande passante avait été détectée. Le même mécanisme visant à ignorer les mesures erronées a donc été mis en place lors des phases d'augmentation de débit (**si** nouveau débit $<$ débit **alors** ...)

Lors d'une augmentation de la bande passante disponible, il peut arriver que le débit de réception soit très supérieur à celui d'émission (*cf.* effet de *Clustering*). Un mécanisme de limitation a donc été mis en place (**si** nouveau débit $>$ 1,1 $\times$ débit **alors** ...) afin d'éviter un augmentation trop importante et inappropriée du débit d'émission pouvant entraîner l'instabilité du réseau. Ce mécanisme permet de plafonner l'augmentation du débit d'émission : le nouveau débit d'émission ne pourra pas dépasser de plus de 10% l'ancien débit d'émission. Cette limitation évite de fortes oscillations du débit d'émission lors de la détection d'une augmentation de la bande passante disponible.

En discret, le lissage du débit d'émission calculé par le correcteur étant moins bon, un second mécanisme de lissage a été ajouté. En effet, une moyenne pondérée du débit d'émission est calculée en utilisant le débit actuel (débit qu'avait la source lors de l'émission du dernier paquet), le nouveau débit ainsi qu'un coefficient de pondération (β). Notons que plus β sera grand ($0 \geq \beta \geq 1$), plus les variations de débit seront atténuées et le débit d'émission lissé.

4.6.3 Chien de garde

Contrairement aux protocoles basés sur une fenêtre d'émission, Primo ne possède pas de dispositif qui lui permette de bloquer l'émission des données en cas de non réception d'acquittements. Il a donc fallu en créer un. En effet, afin d'éviter que Primo ne continue d'engorger le réseau en cas de grave congestion (cas où il ne reçoit plus d'acquittement), un système de chien de garde a été implémenté. Ce chien de garde se déclenche lorsque l'émetteur n'a pas reçu d'acquittement depuis un certain temps.

Durant la phase d'initialisation, ce chien de garde est armé à une valeur arbitraire de 3 s, ce qui signifie qu'après l'émission des quatre premiers paquets, si le premier acquittement n'arrive pas dans un délai de 3 s, la phase d'initialisation est relancée. Une fois le premier acquittement reçu, le chien de garde est armé à deux fois la valeur du premier RTT mesuré. Entre la réception du premier et du second acquittement, il s'écoule au minimum un RTT. En effet, lors de l'initialisation, seulement quatre paquets sont émis, puis la source arrête d'émettre durant un RTT (*i.e.*, jusqu'à l'arrivée du premier acquittement).

Algorithme 2 : **Primo(deb, t, deb_recv)**

```
1   erreur FTT = tab_dev_FTT[t-RET]
2   si erreur FTT ≥ erreur_sup2 alors
      ▷ Cas d'une congestion importante
3     nouveau débit = débit + α × (débit - (réduction_1 × débit réception))
      ▷ Forte réduction du débit d'émission
4     si nouveau débit > débit alors
5       nouveau débit = débit
6     fin si
7   sinon
8     si erreur FTT ≥ erreur_sup1 alors
        ▷ Cas d'une petite congestion
9       nouveau débit = débit + α × (débit - (réduction_2 × débit réception))
        ▷ Réduction du débit d'émission
10      si nouveau débit > débit alors
11        nouveau débit = débit
12      fin si
13    sinon
14      si erreur FTT < erreur_inf alors
          ▷ Risque de sous utilisation du réseau
15        nouveau débit = débit + α × (débit - (augmentation_1 × débit réception))
          ▷ Augmentation du débit d'émission
16        si nouveau débit > 1,1 × débit alors
17          nouveau débit = 1,1 × débit
18        sinon
19          si nouveau débit < débit alors
20            nouveau débit = débit
21          fin si
22        fin si
23      fin si
24    fin si
25    nouveau débit = β × débit actuel+ (1 - β) × nouveau débit
```

À la réception du second acquittement, le régime permanent est pleinement lancé (*i.e.*, il n'y a plus d'arrêt dans l'émission des données). L'émission ininterrompue des données permet de réduire la durée du chien de garde à la valeur du FTT consigne additionné du délai inter-paquets. Cette réduction de l'attente avant l'expiration du chien de garde permet de diminuer la durée des émissions saturant le réseau en cas de rupture d'un lien ou de congestion extrême. Notons que le chien de garde est réarmé à chaque réception d'acquittement.

Si le chien de garde vient à expirer alors que le régime permanent est atteint, le débit d'émission sera réduit de moitié. Quel que soit le régime dans lequel se trouve le protocole, à la suite d'une expiration du chien de garde, la durée de son attente est doublée et un délai d'attente aléatoire (ne pouvant dépassé une seconde) y est ajouté. Ce délai aléatoire permet d'éviter d'éventuels problèmes liés à une synchronisation des sources. Enfin, notons que, quelle que soit la phase dans laquelle se trouve l'émetteur, au bout de quatre expirations successives du chien de garde (*i.e.*, aucune réception d'acquittement entre chaque expiration du chien de garde), la connexion sera arrêtée.

4.6.4 Résumé des paramètres

Les valeurs attribuées aux différents paramètres du protocole ont été obtenues par ajustements successifs (grâce à des simulations) durant la phase de développement :

- le gain α = -0,5 ;
- réduction_1 = 0,95 ;
- réduction_2 = 0,99 ;
- augmentation_1 = 1,05 ;
- RET = RTT intial ;
- erreur$_{sup1}$ = 0,17 $\times$ FTT consigne ;
- erreur$_{sup2}$ = 0,34 $\times$ FTT consigne ;
- erreur$_{inf}$ = 0,15 $\times$ FTT consigne ;
- β = 0,9 ;
- durée initiale de l'attente du chien de garde de retransmission = 3 secondes ;
- nombre d'expiration successives du chien de garde avant fermeture de la connexion = 4.

Notons que ces paramètres peuvent être ajustés en fonction du type de réseau sur lequel évolura Primo. Notons également que ces paramètres pourraient s'ajuster automatiquement aux conditions du réseau grâce à une loi d'adaptation.

4.7 Conclusion

Les protocoles de transport basent essentiellement le contrôle de congestion sur des règles empiriques et la plupart de ces protocoles utilisent les pertes de données pour s'informer sur l'état de congestion du réseau [6, 56, 42]. Cette méthode de détection n'est pas préventive ; elle induit la création de congestions au lieu de les éviter. L'une des méthodes permettant de créer un contrôle de congestion préventif est d'utiliser la modélisation continue. L'utilisation d'un modèle continu autorise une étude simplifiée du comportement du réseau.

Le correcteur construit dans cette section répond aux objectifs que nous nous étions fixés :

- **Rapidité** : grâce à l'estimation de la bande passante initialement disponible, la source atteint un débit d'émission utilisant la totalité de la bande passante disponible en un RTT. En comparaison à des mécanismes tel que celui du *slow start*, le temps que met la source Primo à atteindre le débit optimal est très court. Cette rapidité d'estimation de la bande passante permettra d'améliorer le temps de transfert de petits fichiers.
- **Prévention** : en utilisant le temps d'acheminement comme indicateur sur l'état de congestion du réseau, le correcteur est capable d'ajuster le débit de la source avant que des pertes de paquets ne surviennent. Cette caractéristique permet de réduire le nombre de pertes de paquets et donc de mieux utiliser les ressources disponibles.
- **Stabilité du débit** : grâce à la composante « intégrale », le correcteur parvient à lisser le débit d'émission et, par conséquent, celui de réception. Le lissage du débit d'émission améliore la stabilité du réseau et permet à ce protocole d'être utilisé pour le transport de flux temps réel (ex : visio-conférence).
- **Réaction aux perturbations** : l'adaptation du débit étant proportionnelle à la charge du réseau (*i.e.*, aux congestions), les perturbations sont automatiquement prises en compte dans l'ajustement du débit d'émission.

L'utilisation de la composante « proportionnelle » assure que le correcteur aura une réaction (*i.e.*, ajustement du débit d'émission) adaptée au niveau de congestion du réseau, ce qui n'est pas le cas pour un protocole tel que TCP, qui réagit de la même manière quelle que soit la perturbation rencontrée (réduction de 50% du débit d'émission puis accroissement linéaire). La cohérence du niveau de réaction par rapport au niveau de congestion est un élément déterminant pour utiliser au mieux les capacités d'un réseau. En effet, si les actions engendrées par la détection d'une congestion sont disproportionnées, la bande passante disponible sera fréquemment inutilisée. Inversement, si les réductions de débit sont insuffisantes, le réseau sera fréquemment saturé. C'est pourquoi, un correcteur réagissant proportionnellement à l'état du système et tenant compte de son historique (pour éviter des variations de débit inutiles) semble bien adapté au contrôle de congestion.

Perspectives. Le correcteur décrit dans ce chapitre pourrait donner lieu à la définition d'un correcteur « adaptatif », dont la loi d'adaptation serait définie par l'état du réseau. Ce type de correcteur permet d'ajuster la valeur des coefficients d'augmentation et de réduction du débit d'émission en fonction de l'état du réseau (*i.e.*, de la loi d'adaptation).

Le protocole de transport Primo présenté dans ce chapitre n'implémente pas de mécanisme de fiabilité. En effet, actuellement lorsqu'un paquet est perdu il n'est pas retransmis. Dans le cadre d'une transmission vidéo, cette fonctionnalité n'a pas d'importance. En revanche, si l'on souhaite utiliser Primo pour transmettre des fichiers, il faudra ajouter un mécanisme fiabilisant le transport des données. Notons également que, pour l'instant, Primo n'implémente pas de mécanisme de *Piggybacking*. Un tel mécanisme permet d'envoyer des données dans les acquittements et ainsi d'optimiser la transmission des données.

Si Primo devait être implémenté dans un environnement réel, ces deux fonctionnalités seraient indispensables. Mais dans le cadre de notre étude, portant uniquement sur les mécanismes de contrôle de congestion, ces aspects ne sont pas nécessaires et, par conséquent, ils n'ont pas été implémentés.

Chapitre 5

Méthode d'évaluation des performances

Dans ce chapitre, nous développons une méthode permettant d'évaluer des performances du contrôle de congestion d'un protocole de transport. Pour commencer, différents modes d'évaluation sont décrits dans la section 5.1, les avantages et les inconvénients de chacun y sont soulignés. Puis, dans la section 5.2, les origines, la structure et le fonctionnement du simulateur *ns* sont brièvement décrits. Les paramètres de simulations sont ensuite détaillés dans la section 5.3. Pour finir, la section 5.4 décrit les critères retenus pour évaluer les performances des mécanismes de contrôle de congestion.

Positionnement. Actuellement, nous n'avons recensé aucune méthodologie de référence permettant une comparaison multi-critères des protocoles de transport. En effet, chaque étude visant à comparer les mécanismes de contrôle de congestion propose ses propres scénarios d'utilisation ainsi que ses propres critères d'efficacité. Cette situation donne naissance à des résultats relativement contradictoires. Ainsi dans [1] et [12], les résultats obtenus lors de l'évaluation de TCP Vegas le décrivent comme peu efficace, alors que dans [25] le contraire est présenté. En effet, suivant les critères de performance choisis ou selon la situation dans laquelle est placé un protocole, les résultats peuvent être très différents. Il est donc indispensable de définir une méthode d'évaluation des performances objective. Elle devra être multi-critères afin d'évaluer l'ensemble des performances d'un protocole et non uniquement un point précis. Cette méthode d'évaluation devra également permettre d'étudier le protocole dans des situations variées afin d'évaluer les performances de manière générale et non dans un environnement restreint. Notons que le projet TANGO [111], en cours de développement, proposera, à terme, une méthodologie pour la conception, le développement et l'évaluation des nouveaux protocoles.

Un protocole peut être évalué de plusieurs manières. La première venant à l'esprit est le test en environnement réel, mais ce mode d'évaluation n'est pas toujours techniquement réalisable. D'autres modes d'évaluation telle que l'émulation ou la simulation sont fréquemment employés (*cf.* section 5.1).

Lorsque le mode d'évaluation utilisé est la simulation, il faut choisir le simulateur à utiliser. Le simulateur le plus utilisé actuellement pour des recherches liées au contrôle de congestion des protocoles de transport est sans conteste *ns* (*cf.* section 5.2). Ce simulateur est celui qui a été utilisé pour comparer les performances de Primo à celles d'autres protocoles de transport.

La seconde étape, la plus complexe [38], pour évaluer un protocole *via* la simulation est de définir les différents paramètres de simulations employés (topologies, paramètrages du réseau,

protocoles à comparer, *etc.*).

La dernière étape consiste à définir des critères de performances, ainsi que les mesures nécessaires à leur évaluation (*cf.* section 5.4). Les mesures n'étant pas directement exploitables, le *post* traitement qui leur est appliqué est présenté. Ce *post* traitement permet d'obtenir des indicateurs numériques autorisant une comparaison objective des performances des protocoles testés.

5.1 Mode d'évaluation

Dans cette section nous décrivons les différents modes d'évaluation existant en soulignant leurs avantages et inconvénients.

5.1.1 Environnement réel

Actuellement, l'expérimentation en environnement réel [25, 2] est le seul moyen d'inclure dans les tests tous les effets dus aux aléas de l'implémentation d'un protocole. En effet, l'interprétation des spécifications fournies dans les RFCs peut être différente d'une implémentation à l'autre. Ces différences ont pour effet de modifier le comportement du protocole et donc l'évolution du trafic auquel il contribue. Les tests en environnements réels ont donc l'avantage de confronter un nouveau protocole à ce qu'il rencontrera réellement sur un réseau.

Les tests en environnement réel peuvent être de deux types :

- **Tests sur un réseau isolé**. Ce premier type de tests permet d'éprouver dans un premier temps l'implémentation d'un nouveau protocole sur un réseau local. Cette méthode consiste à envoyer plusieurs flux de données entre différentes machines du réseau local et à observer les comportements des protocoles testés. Cette technique de test a l'avantage de soumettre les protocoles à un environnement réel dont les paramètres (de par leur quantité et leur diversité) sont non reproductibles en simulations. En revanche, dans un tel environnement, l'évaluation des résultats est plus complexe car les paramètres à mesurer sont moins accessibles que dans un simulateur. Outre les difficultés rencontrées pour la récupération des résultats, la rigidité de ce genre de tests pose également problème. En effet, le nombre de topologies utilisées (lié au type de réseau et au nombre de machines disponibles), ainsi que les types de protocoles testés (liés aux systèmes d'exploitation installés sur les machines) sont assez restreints. Les résultats obtenus dans un tel environnement ne sont donc pas généralisables à l'ensemble des réseaux.
- **Tests sur Internet**. L'étape ultime dans l'évaluation d'un protocole consiste à le tester sur Internet. Ce genre de tests est difficile à réaliser car il faut disposer d'un éventail représentatif (configuration et localisation) de machines connectées à Internet. Cependant, le projet NIMI[1] [85] (ou son équivalent Européen RIPE[2] [103]) permet d'administrer un ensemble d'hôtes répartis principalement sur le territoire américain dont la fonction est de servir de stations de mesures. Cette infrastructure permet de mesurer différents paramètres d'un flux de données comme le débit, le taux de pertes de paquets, l'équité,

1. *National Internet Measurement Infrastructure.*
2. Réseaux IP Européens.

etc. Les résultats obtenus lors de tests sur Internet sont généralement difficiles à exploiter, tant le nombre de paramètres pouvant influer sur un protocole est important. Cependant, afin d'éviter les graves conséquences qu'aurait le déploiement d'un « mauvais » protocole sur Internet, les tests à large échelle (Internet) sont indispensables.

Le test en environnement réel est donc un mode d'évaluation apportant beaucoup de précisions et garantissant l'exactitude des résultats obtenus. Cependant si ces tests sont uniquement effectués sur un réseau privé, les résultats obtenus ne seront valables que pour ce réseau. En effet, les conditions rencontrées sur un réseau privé ne sont pas forcément représentatives de celles rencontrées dans l'ensemble des réseaux privés et encore moins à celles d'Internet. De plus, ce mode d'évaluation est difficile à mettre en œuvre, car la phase de développement qu'il nécessite est vraiment conséquente (remplacement d'une couche protocolaire dans un système d'exploitation). De plus, dans le cas de tests sur un réseau à grande échelle (*i.e.*, Internet), les résultats sont difficiles à exploiter car ils dépendent d'un grand nombre de paramètres non maîtrisés. Ce genre de tests est donc généralement effectué après avoir validé le nouveau protocole par simulations ou par émulations (*i.e.*, dans des environnements où les paramètres influents sont maîtrisés).

5.1.2 Simulations

La simulation consiste à recréer un environnement de tests artificiel grâce à un logiciel. Cet environnement artificiel permettra d'effectuer diverses expériences irréalisables dans un environnement réel (coût financier, maîtrise des paramètres, niveau d'abstraction du tests, *etc.*). Cette méthode est de loin la plus utilisée dans les études visant à évaluer les performances des protocoles. Les simulateurs les plus répandus sont *OpNET* [87], *x-sim* [17] et *ns* [86]. Ils sont activement supportés et de nombreuses améliorations y sont régulièrement apportées. Ce type d'évaluation présente tout d'abord un avantage économique car elle ne nécessite qu'une simple machine et un logiciel de simulation (qui dans le cas de *ns* est gratuit). Ce genre d'évaluation permet également de tester un grand nombre de modèles : topologies, capacités des équipements, protocoles utilisés, *etc.* Cette souplesse dans la création des scénarios de simulation permet d'envisager un grand nombre de situations, ce qui n'est pas le cas avec le mode d'évaluation précédent. De plus, la récupération et l'exploitation des résultats sont généralement très simples et permettent une interprétation aisée (tous les paramètres de simulation étant maîtrisés). La simulation semble donc être une solution idéale permettant de modéliser relativement aisément des situations qui peuvent apporter des preuves pertinentes de l'efficacité d'un protocole.

Cependant, il est nécessaire de nuancer ce tableau qui semble parfait. Malgré les avantages indéniables de la simulation, il apparaît que cette méthode souffre de certains inconvénients. Par exemple, les protocoles utilisés dans les simulateurs ne reflètent pas exactement le comportement des protocoles implémentés dans les systèmes d'exploitation (*cf.* interprétation de RFC). De plus, de par la diversité des situations que l'on peut trouver sur Internet, il est impossible de définir des scénarios de simulations (topologies, paramètres réseau, traffic, *etc.*) reflétant exactement la réalité [38]. Les résultats obtenus par la simulation doivent donc être pris comme une indication pertinente du comportement d'un protocole et non comme une représentation exacte de son comportement en environnement réel. Ce mode d'évaluation a donc de nombreux avantages, mais il faut garder à l'esprit qu'une simulation ne représentera jamais parfaitement la réalité.

5.1.3 Émulation

Une application courante de l'émulation est de transformer une machine disposant de plusieurs interfaces réseau en un routeur grâce à des outils logiciels. Il est alors possible de connecter cette machine à plusieurs réseaux pour qu'elle agisse exactement comme le ferait un véritable routeur. Ce genre de manipulation apporte un contrôle total du fonctionnement de la machine émulée (changement de la politique des files d'attente, changement de leur taille, *etc.*), ce qui offre une certaine souplesse pour faire des tests [25, 2]. L'installation d'une machine émulée dans un réseau étant complètement transparente pour les autres machines, les tests effectués dans cette configuration reflètent parfaitement le comportement des protocoles implémentés sur les différentes machines du réseau.

L'émulation peut être utilisée sous deux formes :

– **Émulation dans un environnement réel**. Ce type d'émulation permet, par exemple, de créer un réseau local (LAN[3]) ayant le comportement d'un réseau à grande échelle (WAN[4]). Dans [2], les auteurs montrent qu'en appliquant des *patches* à six machines reliées entre elles par des liens Ethernet, ils sont parvenus à étudier le comportement qu'aurait TCP Vegas sur un réseau à grande échelle. Cette méthode permet, avec de faibles moyens, de réaliser des tests sur des réseaux ayant les mêmes caractéristiques que des réseaux à grande échelle.

– **Émulation dans un environnement de simulation**. L'émulation peut également être utilisée en collaboration avec la simulation en injectant par exemple des flux réels dans un simulateur. Dans ce cas, le simulateur fait partie intégrante d'un réseau réel, ce qui lui permet d'exploiter de vraies données. Dans cette configuration, un trafic réel est inséré dans le simulateur. Ce trafic est traité dans le réseau virtuel du simulateur avant d'être réinjecté dans le réseau réel dont l'émulateur fait partie. Dans un pareil cas, le simulateur utilisé contient nécessairement une interface capable de convertir les données réelles en données virtuelles et inversement (le simulateur *ns* dispose de cette interface).

L'émulation est donc une méthode de mesure à mi-chemin entre la simulation et les tests en environnement réel. En effet, elle allie à la fois *software* et *hardware,* en tirant avantages des deux : une certaine liberté dans les tests couplée à l'assurance de résultats très proches de la réalité. Les avantages de ce mode d'évaluation sont également ses défauts. En effet, l'émulation n'apporte pas autant de souplesse et de simplicité de mise en œuvre que la simulation, ni l'exactitude des résultats obtenus lors de tests en environnement réel.

5.1.4 Conclusion

Bien que les tests en environnement réel permettent de coller à la réalité, ce mode d'évaluation n'est pas le plus pertinent pour le contrôle de congestion. En effet, il permet uniquement de se placer dans un environnement actuel sans pouvoir se projeter dans l'avenir. De plus, les topologies et le trafic sur Internet sont complexes. Il est donc quasiment impossible d'analyser le comportement d'un mécanisme de contrôle de congestion dans un environnement de ce type. L'émulation se confronte au même genre de problème. La simulation apparaît donc comme le

3. *Local Aera Network.*
4. *Wide Aera Network.*

mode d'évaluation le plus adapté pour l'étude de mécanisme de contrôle de congestion [38]. Cela offre aussi une grande souplesse et permet ainsi d'évaluer un protocole dans un grand nombre de situations (qu'elles soient représentatives des conditions actuelles ou non). De plus, il est reconnu [38] que dans le cadre de l'étude du contrôle de congestion, les scénarios de simulation les plus simples (*i.e.*, un environnement simple que seule la simulation peut fournir) sont souvent les plus pertinents.

5.2 Simulateur ns

Pour que les résultats obtenus lors de diverses évaluations de performances soient comparables, il faut que ces évaluations aient toutes utilisées le même simulateur. En effet, l'utilisation d'un simulateur commun permet de s'assurer que l'implémentation des protocoles testés (*i.e.*, le comportement) est la même et ainsi autorise la comparaison des résultats. Acutellement, le simulateur *ns* fait office de référence dans le domaine de la recherche en réseau. Ce simulateur est donc celui qui a été choisi pour le développement de cette méthode d'évaluation.

5.2.1 Origines et structure

Origines. En 1988, le simulateur *REAL* [100] vit le jour. Il était destiné à étudier la dynamique des flots ainsi que les mécanismes de contrôle de congestion dans les réseaux à paquets commutés. Ce simulateur était composé d'une trentaine de modules (écrits en C) émulant le fonctionnement des principaux protocoles de contrôle de congestion (tel que TCP).

À ses débuts, le simulateur *ns* était une variante de *REAL*, puis il a fortement évolué durant ces dernières années. En 1995, le développement de *ns* a été subventionné par l'organisme DARPA[5] à travers le projet VINT[6] [116]. Le projet VINT avait pour but de construire un simulateur de réseau qui permettrait l'étude de l'interaction des protocoles dans les réseaux actuels et futurs. Pour construire ce simulateur, cinq axes de développement ont été mis en avant :

- **Architecture modulaire**. Le simulateur devait fournir une structure autorisant le développement de modules de simulations sur toutes les couches du modèle OSI. Une telle structure représente la modularité de l'architecture d'Internet, et permet rapidement de développer et d'évaluer de nouveaux protocoles.
- **Niveau d'abstraction**. Dans le cadre de la simulation d'un réseau réel, il est difficile d'extraire de la masse de données récupérées les informations pertinentes. Il a donc été spécifié que le simulateur devait offrir la possibilité de faire varier le niveau d'abstraction. Cette caractéristique permet à l'utilisateur d'isoler un phénomène pertinent grâce à des simulations de haut niveau d'abstraction, puis d'approfondir les résultats obtenus en effectuant des simulations plus détaillées.
- **Outils de visualisation**. Le simulateur développé devait être accompagné d'outils de visualisation. En effet, certains comportements peuvent rester invisibles si l'on se contente d'utiliser des résultats statistiques. Les outils de visualisation sont donc indispensables à l'élaboration de simulations pertinentes.

5. *Defense Advanced Research Projects Agency*.
6. *Virtual InterNetwork Testbed*.

- **Interface d'émulation**. Le simulateur devait être capable de se connecter à un réseau réel et de l'utiliser comme une partie de la topologie de simulation. Cette fonctionnalité permet d'améliorer le code source d'un protocole avant de le déployer sur un réseau.
- **Librairies de topologies et de trafic**. Le simulateur devait fournir un grand nombre de topologies et de styles de trafic afin que les protocoles puissent être testés dans diverses situations.

Le simulateur *ns* a été développé en suivant les spécifications du projet VINT [116]. Actuellement, ce simulateur à événements discrets fait office de référence et la plupart des nouveaux protocoles sont évalués avec ce simulateur.

Structure de *ns*. Le simulateur *ns* utilise deux langages de programmation : *C++* et *OTCL*[7] (*TCL* orienté objet). Ce simulateur a besoin d'un langage qui autorise l'implémentation de protocoles ayant des algorithmes parfois assez lourds tout en manipulant efficacement les octets, les en-têtes de paquets, *etc.* Pour toutes ces tâches, le temps d'exécution est très important, et le temps passé à compiler et à déboguer l'est moins. Le simulateur *ns* utilise donc le *C++*, car ce langage est rapide en exécution, mais plus compliqué à modifier (modification du code + compilation). D'un autre coté, la recherche en réseau induit de nombreuses modifications des paramètres de configuration. En effet, un ensemble de simulations est souvent composé de différentes topologies et de différents scénarios. Dans ces conditions, la facilité à modifier le code et à le ré-exécuter prime sur le temps d'exécution. Le simulateur *ns* utilise donc des scripts *OTCL* qui rendent la modification des paramètres simple et rapide mais qui sont plus lents pour l'exécution.

En résumé, pour le développement d'un protocole, il sera préférable d'utiliser le langage *C++*, et pour évaluer le protocole dans différentes situations, les scripts *OTCL* seront plus adaptés à la création des scénarios. Notons que les protocoles (développés en *C++*) et les scripts de simulations (écrits en *OTCL*) ont la possibilité de partager des variables. De plus, il est également possible de faire des appels aux instances *OTCL* dans le code *C++*, et réciproquement les scripts *OTCL* ont la possibilité d'appeler des fonctions *C++*.

Les fonctionnalités des langages orientés objet, notamment l'héritage, sont largement utilisées dans *ns*. En effet, la plupart des classes sont héritières d'une autre classe. Par exemple, il existe une classe TCP qui regroupe toutes les fonctionnalités classiques d'une implémentation de TCP Tahoe. L'implémentation des autres versions de TCP (Reno, Vegas, *etc.*) héritent de la classe TCP (Tahoe). De plus, l'orientation objet de ce simulateur facilite l'intégration de nouveaux protocoles. Il est en effet très simple de réutiliser les classes existantes pour en développer de nouvelles.

5.2.2 Scripts de simulations

Les scripts de simulation *OTCL* sont assez simples à mettre en œuvre, et peuvent être décomposés en cinq parties :

- **Environnement de simulation**. Pour pouvoir effectuer des simulations avec *ns*, il faut commencer par créer un nouvel objet « simulateur » dans le script *OTCL* :

```
set ns [new Simulator]
```

7. *Object TCL*.

Cette commande entraîne la création d'un objet simulateur dont le nom sera `ns`. Cet objet est indispensable à la création et à l'exécution d'une simulation.

– **Création de la topologie**. La création de la topologie se déroule en deux phases. Il faut commencer par créer les nœuds (`n0` et `n1`) :

```
set n0 [$ns node]
set n1 [$ns node]
```

Une fois les nœuds créés, il faut définir les liens les reliant. Les liens sont définis par leur type (uni ou bidirectionnel), un nœud de départ et d'arrivée, leur bande passante, leur temps de propagation et le type de la file d'attente qui les précède :

```
$ns duplex-link $n0 $n1 10Mb 15ms RED
```

Les trois lignes de scripts précédentes créent un réseau composé de deux nœuds reliés par un lien bidirectionnel, ayant une bande passante de 10 Mbits/s, un temps de propagation de 15 ms et dont les files d'attente sont de type RED (*cf.* paragraphe 2.5.2).

– **Création des flots de données**. Dans *ns*, les flots de données sont générés par un trafic ou une application et sont transportés grâce aux agents. Les agents représentent les protocoles de transport (TCP, UDP, *etc.*) et sont attachés à un nœud :

```
set tcp0 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $tcp1
```

Dans cet exemple, l'agent `tcp0` est une source implémentant le protocole TCP Reno qui se situe sur le nœud `n0`, alors que l'agent `sink0` est un récepteur implémentant le protocole TCP qui se situe sur le nœud `n1`. Pour pouvoir communiquer, les deux agents doivent être connectés :

```
$ns connect $tcp0 $sink0
```

Le mode de transport des données étant défini, il ne reste plus qu'à spécifier le type de trafic ou d'application générant les données à transporter :

```
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

Ici, l'application FTP (`ftp0`) utilisera la source `tcp0` pour transmettre les données au récepteur `sink0` (situés respectivement sur les nœuds `n0` et `n1`).

– **Échéancier.** Dans *ns*, les différentes actions (ici, démarrage et arrêt du transfert de fichers) sont lancées sous forme d'événements discrets et datés. L'échéancier gère également la durée de la simulation :

```
$ns at 1.0 "$ftp0 start"
$ns at 15.0 "$ftp0 stop"
$ns at 20.0 exit
```

Ces lignes indiquent que le transfert de fichiers entre les deux nœuds commencera après une seconde de simulation et s'arrêtera au bout de 15 s. La simulation, quant à elle, prendra fin au bout de 20 s.

– **Lancement de la simulation**. Enfin, pour effectuer la simulation, il faut la lancer à la fin du script :
```
$ns run
```

Les scripts de simulation sont donc très simples à créer, et facilement modifiables. De plus, il existe des librairies de topologies et de trafics simplifiant encore la mise en œuvre des scripts. Ces librairies donnent la possibilité de tester les protocoles dans un grand nombre de situations. Il est également possible d'approfondir les résultats obtenus avec des simulations de haut niveau d'abstraction en spécifiant certains paramètres telles que la taille maximale de la fenêtre de congestion de TCP, la technique de routage employée, *etc.*

5.2.3 Résultats de simulations

Par défaut *ns* propose deux types de résultats donnant la possibilité d'analyser les simulations. Le premier type de résultats fourni par *ns* est un fichier d'animation donnant la possibilité de visualiser l'évolution du réseau. La lecture de ce fichier se fait grâce à nam[8] (associé à *ns*). Cet utilitaire affiche schématiquement le réseau, les paquets y circulant ainsi que les éventuelles files d'attente (*cf.* figure 5.1). Il offre également la possibilité d'observer dynamiquement l'évolution de certaines variables liées au protocole ou aux files d'attente.

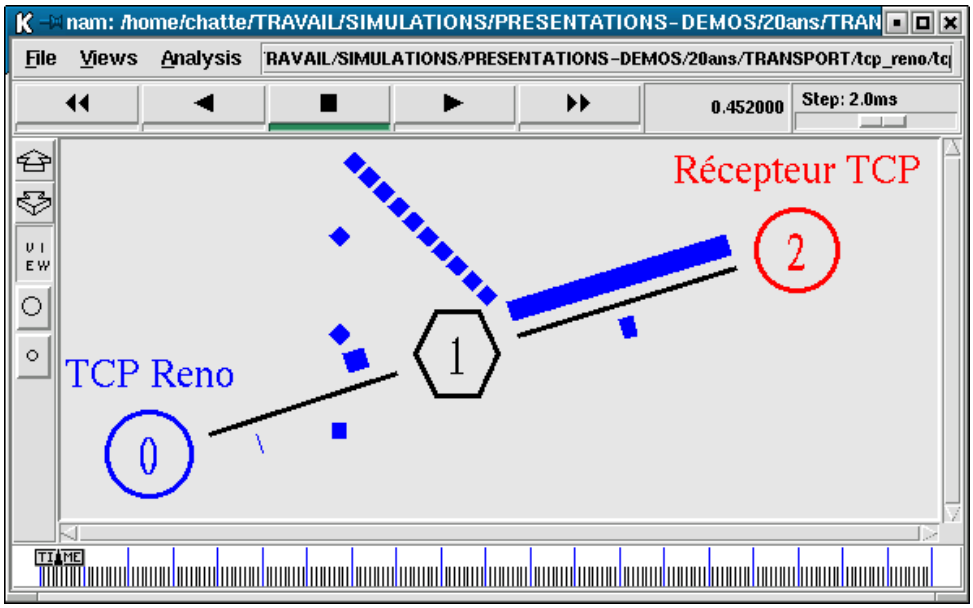

FIG. 5.1 – *Utilitaire de visualisation nam.*

Le simulateur *ns* peut également fournir une trace de simulation qui répertorie tous les événements qui ont eu lieu. Chaque ligne de cette trace représente un événement, et contient 17 champs (type d'événement, date, nœuds source et destination, numéro du paquet, type d'application ou de trafic, taille du paquet, *etc.*) :
```
+ 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
r 1.002336 0 2 cbr 210 ------- 0 0.0 3.1 0 0
```

8. *network animator*

```
d 1.002336 2 3 cbr 210 ------- 0 0.0 3.1 0 0
+ 1.002336 2 3 tcp 210 ------- 0 0.0 3.1 0 0
- 1.00375 0 2 ack 210 ------- 0 0.0 3.1 1 1
```

Ces traces sont très complètes mais difficiles à exploiter en l'état. Il est également possible de générer des traces ne comportant qu'un type d'événement pour un nœud précis (ex : arrivée d'un paquet sur le nœud i). Ces dernières contiennent moins d'informations, mais sont plus simples à exploiter. Enfin, on peut également conserver une trace de certaines variables liées aux protocoles (ex : taille de la fenêtre de congestion pour TCP). Les traces ne comportant qu'un seul type d'événement ou de paramètre lié au protocole sont visualisables sous la forme de courbes en utilisant l'application xgraph (*cf.* figure 5.2).

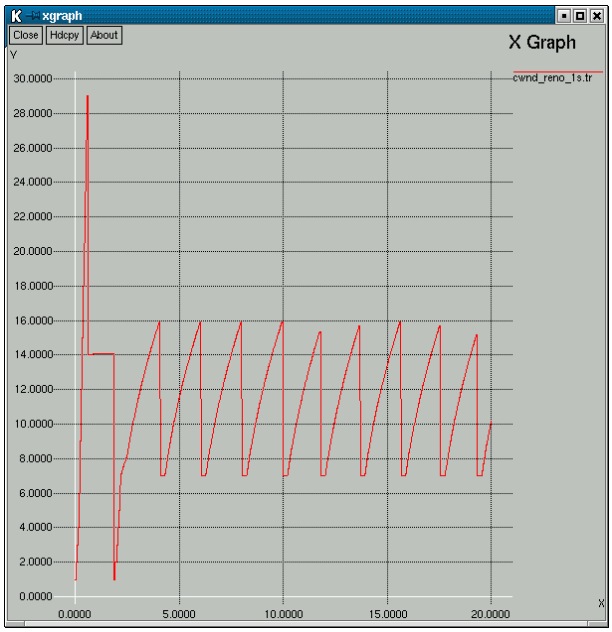

FIG. 5.2 – *Utilitaire de visualisation xgraph.*

5.3 Paramètres de simulations

Dans cette section, nous définissons les différents paramètres du réseaux qui seront utilisés pour évaluer les mécanismes de contrôle de congestion.

5.3.1 Topologies

Les topologies rencontrées dans la littérature faisant référence à l'évaluation des protocoles de transport sont peu nombreuses. Elles peuvent être classées en deux grandes familles :

– Les topologies composées de *n* sources, un lien congestionné (situé entre deux routeurs) et *n* récepteurs (*cf.* figure 5.3). Il existe cependant plusieurs variantes de cette famille de topologies. Par exemple, dans [49, 10] le réseau est simplement composé d'une paire source/destination reliée par l'intermédiaire de deux routeurs. Lorsqu'il n'y a qu'une

source, il arrive également que la topologie se résume à trois nœuds : une source, un routeur et une destination [31]. De même, il existe des réseaux composés de *n* sources, un routeur et une destination [30, 66]. Mais la topologie la plus fréquemment rencontrée est composée de deux à *n* paires source/destination, et de deux routeurs [47, 75, 19, 27, 48]. En bornant le nombre de sources à deux, l'étude de l'impact des différents paramètres du réseau (bande passante, délai de propagation, *etc.*) est simplifiée tout en permettant d'observer l'équité entre les flots. Mais, il est également intéressant de pouvoir faire varier le nombre de sources afin d'analyser l'interaction des connexions entre elles. En effet, certains protocoles sont très sensibles au nombre de connexions avec lesquelles ils partagent la bande passante. Cette famille de topologie, qu'elle soit composée de deux ou de *n* sources, est très pratique pour étudier le comportement des algorithmes de contrôle de congestion.

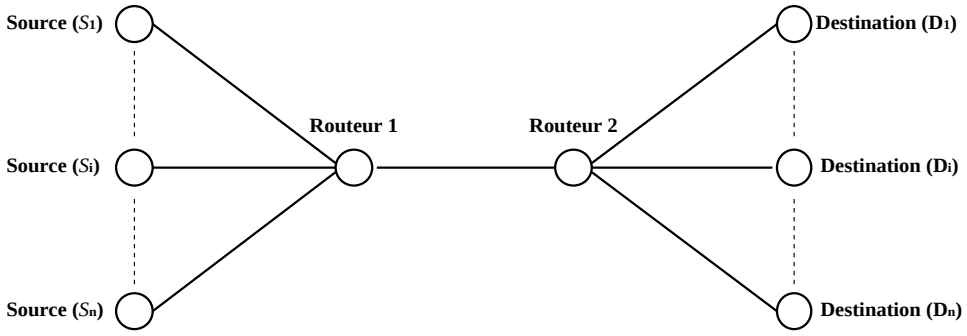

FIG. 5.3 – *Topologie en étoile.*

– Les topologies admettant des chemins à congestions multiples représentent la seconde grande famille de topologies utilisées en évaluation des performances du contrôle de congestion. L'une des topologies [33, 49, 27] de cette famille donne la possibilité d'observer le comportement d'un protocole en fonction du nombre de congestions auxquelles il est confronté. Cette topologie est composée de quatre routeurs et de cinq paires source/destination (*cf.* figure 5.4). Dans cette topologie, on distingue trois groupes de connexions. Le premier est composé d'une paire source/destination (connexion E) qui traverse les quatre routeurs. Les liens sortant des routeurs ayant tous le même débit, un paquet traversant quatre routeurs rencontrera au plus trois congestions. En effet, à la sortie du quatrième routeur les paquets de la connexion E seront forcément émis à un débit inférieur ou égal à la bande passante du lien sortant de ce routeur, et donc aucune congestion ne pourra s'y former. Le second groupe est composé de deux paires source/destination (connexions C et D) qui traversent trois routeurs (*i.e.*, deux congestions potentielles). Le dernier groupe est composé de deux paires source/destination (connexions A et B) qui traversent deux routeurs (*i.e.*, une congestion potentielle). Cette configuration est utilisée pour mettre à l'épreuve l'équité des protocoles et étudier leurs réactions en fonction du nombre de congestions qu'ils rencontrent (une, deux ou trois suivant le groupe de connexions).

Pour l'évaluation des performances de Primo, les deux familles de topologies seront utilisées. Pour la première famille, la topologie utilisée (*cf.* figure 5.3) sera composée de *n* paires source/destination (*n* variant de deux à vingt) et de deux routeurs. Cette topologie a été retenue car elle permet une étude aisée de l'influence des différents paramètres du réseau sur

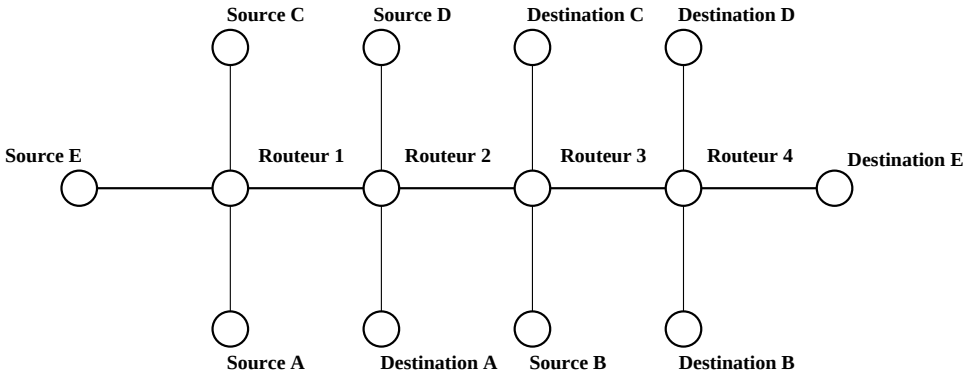

FIG. 5.4 – *Topologie en peigne.*

le mécanisme de contrôle de congestion. Pour la seconde famille (chemin à congestions multiples), la topologie utilisée sera celle décrite précédemment (*cf.* figure 5.4).

5.3.2 Paramètres du réseau

Une fois la topologie retenue, l'influence des différents paramètres du réseau doit être étudiée.

Dans un grand nombre d'études sur l'évaluation des performances du contrôle de congestion, les différents paramètres du réseau sont fixés. Or, l'utilisation d'un réseau avec un unique paramètrage risque fortement de biaiser les résultats obtenus. En effet, rien ne garantit que les résultats obtenus pour une configuration donnée se confirment avec une simulation utilisant d'autres valeurs. Le protocole Primo sera donc comparé aux autres protocoles, pour chaque paramètre du réseau sur une plage de valeurs étendue et pertinente. Les paramètres du réseau influents sur le contrôle de congestion peuvent être classés en trois grandes familles : le rapport de bandes passantes des liens, les files d'attente, et les temps de propagation.

Rapport de bandes passantes des liens. Le rapport entre la somme des bandes passantes des liens entrants (liens situés entre les sources et le routeur) et celle du lien de congestion (lien situé entre les deux routeurs) est un paramètre important dans le cadre de l'évaluation du contrôle de congestion. En effet, plus la bande passante du lien congestionné est faible par rapport à la somme des bandes passantes des liens entrant, plus le risque de congestion sera important. La variation de ce paramètre donne donc la possibilité d'évaluer les capacités d'un protocole à éviter les congestions dans un environnement favorisant plus ou moins leur apparition. Dans [49, 31, 75, 66], la bande passante du lien congestionné est fixe et représente entre 5% et 15% de la somme des bandes passantes des liens entrant. Dans les études proposant de faire varier le rapport de bandes passantes, les plages de variations peuvent être très différentes. Par exemple, dans [27], le rapport de bandes passantes varie de 0,5% à 40%, alors que dans [8], le rapport de bandes passantes varie de 0,001% à 1%.

Certaines plages de variations telles que celles utilisées dans [8] ne testent que les cas à fort risque de congestions, ce qui revient pratiquement au même que de tester le protocole dans un seul cas. Les réseaux n'étant pas saturés en permanence, il est également intéressant d'étudier les cas à faible potentiel de congestion.

Notons que la plage de variation du rapport de bandes passantes choisie pour effectuer les simulations dépend du type de trafic à transporter. Si le trafic utilisé est de type *On / Off* [8] (trafic pour lequel les sources n'émettent pas de manière continue), il n'est pas aberrant de considérer des cas où le rapport de bandes passantes est inférieur à 5%. En revanche, si le trafic utilisé est de type FTP (où toutes les sources émettent simultanément et de manière continue), il n'est pas réaliste d'utiliser un rapport de bandes passantes inférieur à 5%. En effet, il est fréquent que des réseaux soient dimensionnés de sorte que le rapport de bandes passantes soit inférieur à 5%, mais dans ce cas, l'hypothèse est faite que toutes les sources ne seront pas simultanéments actives.

Dans notre cas, le trafic utilisé étant de type FTP (*cf.* paragraphe 5.3.3), la plage de variation du rapport de bandes passantes choisie se situe entre 5 et 80%. Dans la littérature on trouve assez peu de tests effectués dans un environnement à faible potentiel de congestion. Les réseaux n'étant pas perpétuellement congestionnés, il est pertinent d'envisager les situations à faible (*i.e.*, rapport de bandes passantes de 80%) comme celles à fort (*i.e.*, rapport de bandes passantes de 5%) potentiel de congestion.

Files d'attente. Les files d'attente ayant une taille limitée, elles peuvent être amenées, en cas de congestion, à perdre (ou détruire) des paquets. Dans ce cas, deux paramètres doivent être pris en compte : la politique de destruction des paquets et la taille maximale de la file d'attente. Pour ce qui est de la politique de destruction, les algorithmes RED et *Drop Tail* sont généralement utilisés. Ces deux politiques sont actuellement implémentées sur les routeurs ; il est donc nécessaire d'évaluer leur influence sur tous nouveaux protocoles. Quelle que soit la politique de gestion de la file d'attente, sa taille maximale est un paramètre primordial pour le contrôle de congestion. En effet, si la file est de petite taille, elle risque d'être fréquemment saturée. Inversement, si elle est trop grande, cela risque d'induire des temps de transmission très importants, qui nuisent aux flux temps réel. De plus, lors de congestion, si la file d'attente a une grande taille, la source mettra longtemps à réagir, car les informations lui parviendront avec retard important (*i.e.*, les paquets passeront un temps non négligeable dans la file). Cette lenteur de réaction de la source impliquera un grand nombre de pertes et donc une mauvaise utilisation du réseau. Dans la littérature, lorsque les topologies utilisées sont composées de deux sources, la taille maximale de la file d'attente varie entre 5 et 50 paquets [30, 66, 49, 75]. Cette plage de valeurs sera utilisée pour notre évaluation.

Lorsque la politique RED (*cf.* paragraphe 2.5.2) est utilisée, il faut également définir deux seuils, le premier à partir duquel les paquets commenceront à être détruits avec une probabilité croissant avec la taille moyenne de la file, le second pour lequel tous les paquets arrivant seront détruits. Dans le cadre de notre évaluation, le premier seuil sera fixé à la moitié de la taille maximale de la file d'attente. Le second sera fixé à la taille maximale de la file (la probabilité maximale de destruction des paquets sera donc de 1). Dans la littérature, il n'est jamais fait mention des valeurs utilisées pour ces deux seuils, c'est pourquoi nous avons choisi de les fixer arbitrairement. Pour cela, nous avons considéré que lorsque la taille moyenne de la file atteignait la moitié de sa taille maximale (valeur du premier seuil), le risque qu'une congestion se forme était réel. Pour le second seuil, nous avons choisi de détruire systématiquement les paquets arrivant dans la file lorsque celle-ci est complètement saturée : il a donc été fixé à la taille maximale de la file.

Temps de propagation. Les temps de propagation des liens peuvent avoir une forte influence sur le comportement d'un protocole de transport. Dans le cas où la topologie est composée de plusieurs nœuds sources, deux cas peuvent être distingués : les temps de propagation des connexions peuvent être homogènes ou hétérogènes.

- Dans le cas homogène, le temps de propagation de chaque connexion (*i.e.*, RTT de la connexion lorsque le réseau n'est pas congestionné) sera le même. Seule la durée de ce temps de propagation peut avoir un impact sur les réactions du protocole. Plus le temps de propagation de la connexion sera long, plus les sources mettront longtemps à réagir à l'apparition d'une congestion. Dans la littérature, les temps de propagation fréquemment utilisés pour les comparaisons de protocoles varient sur une plage assez étendue, entre 12 ms et 300 ms [30, 31, 75, 66, 47]. Dans notre cas, le temps de propagation des connexions variera entre 14 ms et 404 ms. Cette plage de valeurs permettra de tester le protocole d'une part dans un environnement où il aura la possibilité de réagir rapidement et d'autre part dans une situation où les informations lui parviendront avec un retard important.
- Dans le cas de temps de propagation hétérogènes [75], l'une des connexions aura un RTT plus court que l'autre. Cette situation peut avoir pour effet de rendre une connexion plus agressive que l'autre. Cette configuration est utilisée pour tester l'équité du protocole lorsque les temps de propagation de deux connexions sont différents (ce qui est assez fréquent dans la réalité). Pour évaluer l'équité du protocole testé, l'une des deux connexions aura un temps de propagation fixé à 24 ms, alors que celui de l'autre connexion variera de 24 ms à 122 ms. Cette plage de valeurs a été définie aux vues des résultats obtenus durant des simulations préliminaires. En effet, cette plage de variation est largement suffisante pour mettre en évidence les protocoles ayant un comportement inéquitable dans une telle situation.

5.3.3 Scénarios

De par son comportement préventif, le protocole Primo ne peut être mis en concurrence avec un protocole correctif sous peine d'obtenir des performances médiocres. En effet, lorsque des protocoles correctifs et préventifs cohabitent sur un même réseau, les connexions « préventives » ont du mal à obtenir une part minimum de bande passante. Ce phénomène est dû au fait que lorsqu'une congestion commence à se former sur le réseau (*i.e.*, la taille d'une file d'attente commence à croître), la connexion utilisant le protocole préventif va réduire son débit d'émission alors que l'autre connexion continuera d'augmenter le sien jusqu'à observer la perte d'un de ses paquets. Dans ces conditions, les protocoles préventifs n'ont aucune chance de fonctionner correctement. Notons qu'il n'est pas non plus possible de mettre en concurrence différents protocoles préventifs, car ils n'ont pas forcément le même degré de prévention. En effet, si l'un des deux protocoles a un seuil d'évitement de congestion (seuil à partir duquel il commence à réduire son débit) beaucoup plus bas que celui de l'autre protocole, une situation d'inéquité semblable à celle que nous venons de décrire sera constatée.

L'évaluation du protocole Primo (de type préventif) a donc été faite en supposant qu'il serait utilisé sur un réseau privé ou sur une partie de réseau où il ne cohabiterait qu'avec lui-même. Nous avons donc décidé que pour une simulation donnée, toutes les connexions utiliseraient le même protocole de transport.

Simulations avec une seule congestion. L'influence des paramètres du réseau présentés précédemment a été testée sur une topologie composée de deux paires source/destination et de deux routeurs (*cf.* figure 5.3, page 108). Afin de limiter le nombre de simulations (qui est supérieur à 400), dans un ensemble de simulations où l'influence d'un paramètre du réseau (dont la valeur varie) est étudiée, les autres paramètres sont fixés (à des valeurs raisonnables) et identiques pour toutes les simulations de cet ensemble. Le tableau 5.1 récapitule les valeurs des principaux paramètres du réseau en fonction de l'ensemble de simulations. Dans l'ensemble de simulations portant sur les RTTs hétérogènes, les temps de propagation des liens allant des sources au routeur 1 sont donnés par paires (1/1, 1/2, *etc.*). Le premier chiffre de chaque paire indique le temps de propagation du lien allant de la source 1 au routeur 1 et le second celui du lien allant de la source 2 au routeur 1.

Paramètre étudié	Type de file	Taille de file en paquets	TP-LS en ms	TP-LC en ms	D-LC en Mbits/s
Taille de la file RED	RED	5 10 15 20 30 40 50	1	10	3
Taille de la file Drop-Tail	Drop-Tail	5 10 15 20 30 40 50	1	10	3
RTT homogène	Drop-Tail	30	1	5 10 20 30 50 100 200	3
RTT hétérogène	Drop-Tail	30	1/1 1/2 1/5 1/10 1/15 1/20 1/50	10	3
Rapport de bandes passantes	Drop-Tail	30	1	10	1 2 3 4 6 10 16

TAB. 5.1 – *Récapitulatif des valeurs utilisées lors des simulations. TP-LS : Temps de Propagation des Liens allant des Sources au routeur 1, TP-LC : Temps de Propagation du Lien Congestionné, D-LC : Débit du Lien Congestionné.*

Les paramètres du réseau ne figurant pas dans ce tableau sont communs aux cinq ensembles de simulations :

- le débit des liens allant des sources au routeur 1 est de 10 Mbits/s ;
- le débit des liens allant du routeur 2 aux destinations est toujours égal à celui du lien congestionné ;
- le temps de propagation des liens allant du routeur 2 aux destinations est de 1 ms.

Un dernier ensemble de simulations étudiant l'influence du nombre de sources a été testé. Dans la littérature, ce paramètre est généralement fixé [30, 31, 75, 66, 47, 8] à des valeurs très diverses variant d'une à plusieurs dizaines de sources. Nous avons choisi de faire varier le nombre de source de 2 à 20 afin d'évaluer la capacité de cohabitation du protocole testé lorsqu'il est mis en concurrence avec d'autres flots. Pour cet ensemble, une topologie composée de *n* paires source/destination a été utilisée (*cf.* figure 5.3 page 108). Cette topologie a donc été successivement composée de 2, 3, 5, 7, 10, 15, 20 paires source/destination.

Les autres paramètres du réseau sont restés fixes :

- le temps de propagation des liens allant des sources au routeur 1 est de 1 ms ;

- le temps de propagation du lien congestionné est de 10 ms ;
- le temps de propagation des liens allant du routeur 2 aux destinations est de 1 ms ;
- la file est de type *Drop-Tail* et a une taille maximale de 80 paquets ;
- le débit des liens allant des sources au routeur 1 est de 10 Mbits/s ;
- le débit du lien congestionné représente 15% de la somme du débit des liens entrant sur le routeur 1 ;
- le débit des liens allant du routeur 2 aux destinations est toujours égal à celui du lien congestionné.

Tous les ensembles de simulations ne comportant qu'une seule congestion ont également été testés dans un environnement où des pertes d'acquittements surviennent. La perte d'acquittements peut entraîner des réactions diverses de la part des protocoles. Il est donc important de les évaluer dans cette situation. Pour ce faire, les six ensembles de simulations précédents ont été testés en générant un trafic dans le sens du retour (destinations $\rightarrow$ sources). Ce trafic crée des congestions dans le sens du retour et donc des pertes d'acquittements. Une paire de nœuds source/destination a donc été ajoutée aux topologies utilisées. Le trafic dans le sens du retour est produit par une connexion FTP utilisant le protocole TCP Reno. L'utilisation de TCP Reno garantit l'apparition périodique de congestions, et donc des pertes d'acquittements.

Simulations avec plusieurs congestions. Dans cet ensemble de simulations, la topologie utilisée est composée de quatre routeurs et de cinq paires source/destination (*cf.* figure 5.4 page 108). Cet ensemble de simulations est utilisé pour évaluer l'équité des protocoles dans un environnement multi-congestionné. En effet, suivant la situation du nœud source, les paquets d'une connexion traverseront une, deux, ou trois congestions :

- les paquets des connexions a et b traversent une seule congestion ;
- les paquets des connexions c et d traversent deux congestions ;
- les paquets de la connexion e traversent trois congestions.

Cette situation donne la possibilité d'observer quelle part de bande passante chaque connexion réussit à obtenir. Dans cette configuration, il est également intéressant d'étudier comment une connexion parvient-elle à s'imposer lorsque les autres sont déjà établies depuis un certain temps. Inversement, il est intéressant d'observer comment l'arrivée de nouvelles connexions perturbe les connexions déjà établies. Pour cela, différent scénarios de démarrages des groupes de connexions ont été mis en œuvre.

Protocoles et trafics utilisés. Le protocole Primo a été évalué en le comparant à d'autres protocoles de transport. Les protocoles utilisés pour les comparaisons sont les suivants :

- TCP Reno : cette version de TCP est la plus répandue sur Internet.
- TCP Sack : cette option de TCP est la plus performante parmis celles apportant une amélioration dans la gestion des acquittements.
- TCP Vegas : cette version de TCP est la seule à adopter un comportement préventif.
- TFRC : cette alternative à TCP est la plus aboutie (elle est en court de standardisation).

Notons que l'ensemble de ces protocoles est implémenté sous *ns* ce qui nous autorise à comparer objectivement (*i.e.*, sans erreurs dues à l'implémentation) les résultats de notre nouveau protocole avec ceux obtenus par des protocoles déjà validés sous *ns*.

Le trafic utilisé pour tous les ensembles de simulations correspond au transfert d'un fichier de taille infini *via* FTP. L'utilisation d'un tel trafic simplifie l'exploitation des résultats et évite de les entacher d'erreurs dues à une mauvaise interprétation de comportements liés à un trafic plus complexe.

Les situations dans lesquelles sont testés les différents protocoles dans cette méthode d'évaluation nécessitent la réalisation de 445 simulations décomposées comme suit :

- 6 ensembles de simulations liés aux paramètres du réseau : rapport de bandes passantes (1), file d'attente (2), temps de propagation (2), nombre de sources (1) ;
- 7 simulations par ensembles ;
- 2 groupes de simulation comprenant tous les ensembles de simulation : avec et sans pertes d'acquittements ;
- 5 simulations dans un environnement multi-congestionné ;
- 5 protocoles testés pour chaque simulation.

$$\text{nombre total de simulations} = ((6 \times 7) \times 2 + 5) \times 5 = 445$$

5.4 Exploitations des résultats

Dans cette section nous expliquons quels sont les critères d'évaluation retenus, le calcul des indicateurs correspondant ainsi que la représentation adoptée pour les résultats.

5.4.1 Critères d'efficacité et mesures

Critères d'efficacité. Les critères d'efficacité définissent, d'un point de vue utilisateur ou opérateur, les performances d'un protocole. Les différents critères retenus sont les suivants :

- **Débit d'émission** [48, 2, 19, 10, 27, 31]. La valeur moyenne du débit d'émission doit être aussi grande que possible. Cet aspect du débit d'émission est très important d'un point de vue utilisateur. En effet, plus le débit moyen d'émission est important et plus les transferts de données se feront rapidement. Du point de vue de l'opérateur, la constance du débit d'émission est un critère important. En effet, plus le débit d'émission sera constant (tout en respectant les contraintes liées aux congestions), plus le réseau sera stable.
- **Débit de réception** [66]. Comme pour le débit d'émission, la valeur moyenne du débit de réception doit être la plus grande possible. La constance du débit de réception est non seulement importante d'un point de vue opérateur (*cf.* stabilité du réseau) mais elle l'est aussi d'un point de vue utilisateur dans le cas d'une transmission de flots temps réel (ex : visio-conférence).
- **Taux d'occupation** [33]. D'un point de vue opérateur, ce critère est primordial. Il représente le taux d'utilisation des ressources disponibles sur le réseau, il doit donc être le plus proche possible de 100%. Si le taux d'occupation du réseau est faible, c'est que le réseau est sous utilisé (ex : temps mort durant une transmission FTP).
- **Taux de pertes** [48, 19, 33]. D'un point de vue opérateur, ce critère est également très important. Il représente la quantité de données perdues. Ce critère peut également être caractérisé par la quantité de données retransmises [2] ou en inversement la quantité de

données acquittées [10, 75]. Un fort taux de pertes indique une mauvaise utilisation du réseau.

- **Équité** [10, 75, 33, 47]. Ce critère montre la capacité qu'a un protocole à partager équitablement la bande passante entre plusieurs connexions empruntant un même lien. Idéalement, toutes les connexions ayant en commun un lien congestionné auront la même quantité de bande passante allouée.
- **Taux d'espace libre** [30]. D'un point de vue utilisateur, ce critère est important dans le cas d'une transmission de flots temps réel (ce critère peut être caractérisé par la durée du RTT [10, 2]). Un faible taux d'espace libre de la file d'attente induit un temps de transmission important (temps de propagation + temps passé dans une file d'attente pleine). Or, un temps de transmission élevé nuit à la qualité de réception des flux temps réel. Prenons l'exemple d'une visio-conférence, si le temps de transmission est élevé, les images et le son arriveront avec un certain retard et la communication entre les participants ne sera pas fluide.
- **Stabilité de la file d'attente**. Dans le cadre d'une transmission d'un flot temps réel, ce critère compte beaucoup. En effet, si la taille de la file d'attente n'est pas stable, il risque d'y avoir une gigue importante du délai d'acheminement. Dans la cas d'une visio-conférence les images et le son arriveront plus où moins rapidement (en fonction de la taille de la file) et la communication entre les participants risque d'être saccadée. Notons que ce critère peut également être représenté par la variance du RTT [2].

Mesures. Afin d'évaluer les critères de performance précédents, il est nécessaire d'effectuer certaines mesures durant les simulations. Ces mesures se décomposent en deux groupes :

- **Mesures effectuées sur les nœuds du réseaux**. Ce sont principalement les mesures de débit d'émission et de réception des nœuds sources et destinations. C'est également dans ce groupe que la quantité totale de données reçue par les récepteurs et le nombre de paquets perdus sont mesurés. Ces mesures nécessitent un *post* traitement afin de les rendre exploitables et d'en extraire une information quantitative autorisant une comparaison numérique des différents protocoles testés.
- **Mesures effectuées sur les files d'attente**. Il s'agit de la récupération périodique de la taille de la file d'attente. Cet ensemble de valeur permet de tracer une courbe et donc d'observer la dynamique de congestion du réseau. Cette mesure donne également la possibilité de déterminer le taux moyen d'espace libre dans la file d'attente ainsi que sa stabilité.

5.4.2 Indicateurs

Afin de simplifier la lecture des résultats, à partir des critères d'efficacité précédemment définis, des indicateurs autorisant une comparaison numérique ont été créés. Ces indicateurs sont issus du *post* traitement des mesures effectuées sur les nœuds du réseau. Les sept indicateurs obtenus sont les suivants :

- **Constance du débit d'émission**. Pour calculer la constance du débit d'émission, il faut commencer par calculer le débit d'émission moyen :

$$\text{Débit d'émission moyen} = \frac{\text{quantité de données émises}}{\text{durée de simulation}} \tag{5.1}$$

Le débit d'émission instantané est récupéré sous la forme d'un vecteur indiquant les valeurs successives qu'a pris le débit d'émission durant la simulation. La constance du débit d'émission est obtenue en calculant l'écart type du débit d'émission instantané. Pour améliorer la représentation graphique, la valeur obtenue est ensuite normalisée par le débit d'émission moyen puis mise sous forme de pourcentage de sorte que plus le débit d'émission sera constant, plus l'indicateur de constance sera proche de 100% :

$$\text{Constance du débit d'émission} = 100 - \left(\frac{\sqrt{\frac{\sum_1^N (\text{débit instantané} - \text{débit moyen})^2}{N}}}{\text{débit moyen}} \times 100 \right) \quad (5.2)$$

– **Constance du débit de réception**. Cet indicateur se calcule de la même manière que le précédent.

$$\text{Constance du débit de réception} = 100 - \left(\frac{\sqrt{\frac{\sum_1^N (\text{débit instantané} - \text{débit moyen})^2}{N}}}{\text{débit moyen}} \times 100 \right) \quad (5.3)$$

– **Équité moyenne**. L'équité moyenne est calculée en comparant le débit de réception moyen d'une connexion à la part de bande passante qui devrait idéalement lui être allouée. Par exemple, s'il n'y a que deux flots se partageant un lien, le débit de réception idéal sera égal à la moitié de la bande passante du lien partagé. Afin d'améliorer la représentation graphique, l'équité moyenne est donnée en pourcentage.

$$\text{Équité moyenne} = \frac{\text{débit de réception moyen} \times 100}{\text{débit optimal}} \quad (5.4)$$

$$\text{Débit optimal} = \frac{\text{bande passante du lien partagé}}{\text{nombre de flots}} \quad (5.5)$$

– **Taux d'occupation**. Cet indicateur est obtenu en comparant la bande passante du lien congestionné à la somme des débits de réception moyens des connexions (plus le réseau est utilisé au maximum de ses capacités, plus le taux d'occupation est proche de 100%).

$$\text{Taux d'occupation} = \frac{\sum \text{débit de réception moyen}}{\text{bande passante du lien congestionné}} \times 100 \quad (5.6)$$

– **Taux de bonnes transmissions**. Afin que la représentation de l'ensemble des indicateurs soit positive (plus leur valeur tend vers 100% meilleure est la note), nous avons choisi d'utiliser le taux de bonnes transmissions plutôt que le taux de pertes. Cet indicateur est obtenu en comparant le nombre de paquets perdus au nombre de paquets émis. Pour augmenter la lisibilité des résultats, la valeur obtenue est multipliée par 10. En effet, le nombre de paquets perdu étant très faible par rapport au nombre de paquet émis, il est difficile de comparer les protocoles testés en utilisant un simple pourcentage car, dans la plupart des cas, ce taux serait compris entre 98% et 100%.

$$\text{Taux de bonnes transmissions} = 100 - \left(\frac{\text{nombre de paquets perdus}}{\text{nombre de paquets émis}} \times 100 \times 10 \right) \quad (5.7)$$

– **Taux d'espace libre dans la file d'attente**. Cet indicateur est obtenu en calculant la taille moyenne de la file d'attente (la taille instantanée de la file d'attente est récupérée sous la forme d'un vecteur indiquant la taille de la file d'attente au cours du temps). La taille moyenne de la file d'attente est ensuite comparée à sa taille maximale (plus la file sera vide, plus ce taux sera proche de 100%)

$$\text{Taille moyenne de la file d'attente} = \frac{\sum_1^N (\text{Taille instantanée de la file d'attente})}{N} \quad (5.8)$$

$$\text{Taux d'espace libre dans la file d'attente} = 100 - \left(\frac{\text{Taille moyenne de la file}}{\text{Taille maximale de la file}} \times 100 \right) \quad (5.9)$$

– **Constance de la taille de la file d'attente**. La constance de la taille de la file d'attente est calculée de la même manière que celle du débit d'émission (ou de réception). La constance de la taille de la file d'attente est donc obtenue en calculant l'écart type de la taille instantanée. Pour faciliter la lecture graphique des résultats, la valeur obtenue est alors normalisée par la taille moyen de la file d'attente puis mise sous forme de pourcentage, de sorte que plus la taille de la file d'attente est constante, plus l'indicateur de constance est proche de 100% :

$$\text{Constance de la file d'attente} = 100 - \left(\frac{\sqrt{\frac{\sum_1^N (\text{Taille instantanée} - \text{Taille moyenne})^2}{N}}}{\text{Taille moyenne}} \times 100 \right) \quad (5.10)$$

5.4.3 Présentation des résultats

Les sept indicateurs présentés dans le paragraphe 5.4.2 permettent d'évaluer l'ensemble des critères d'efficacité énoncés dans le paragraphe 5.4.1. La plupart de ces critères sont directement valués grâce aux indicateurs. En revanche, les débits moyens (d'émission et de réception) ne sont pas explicitement repris par ces indicateurs, mais ils peuvent en être déduits. En effet, si le taux d'occupation et l'équité sont bons, le débit d'émission moyen sera par conséquent bon, et si le taux de bonnes transmissions est également bon, le débit de réception moyen le sera aussi. Ces sept indicateurs permettent donc une comparaison rapide et facile des différents protocoles testés.

Afin d'en faciliter la lecture, les résultats seront présentés sous forme de graphiques « radar » (*cf.* figure 5.5), composés de sept axes (un par indicateur). Ces sept axes sont tous gradués de 0 à 100, ainsi un protocole qui est performant pour tous les critères testés sera représenté

par un polygone reliant les extrémités des sept axes (*cf.* figure 5.5). Inversement, un protocole non performant sera représenté par un polygone dont les sommets seront des points placés à proximité de la base des axes.

La valeur donnée pour chaque indicateur de performances (*cf.* paragraphe 5.4.2) représente la note moyenne obtenue pour l'ensemble des simulations (composé de sept simulations, *cf.* tableau 5.1), et cela afin de synthétiser les résultats.

Il arrive que l'utilisation de la moyenne biaise les résultats (*i.e.*, la moyenne d'une grande et d'une petite valeur donne une valeur médiane qui ne reflète pas la réalité). Lorsque c'est le cas, les commentaires accompagnant les graphiques le précisent. De plus, la totalité des résultats obtenus aux simulations se trouve dans l'annexe D.

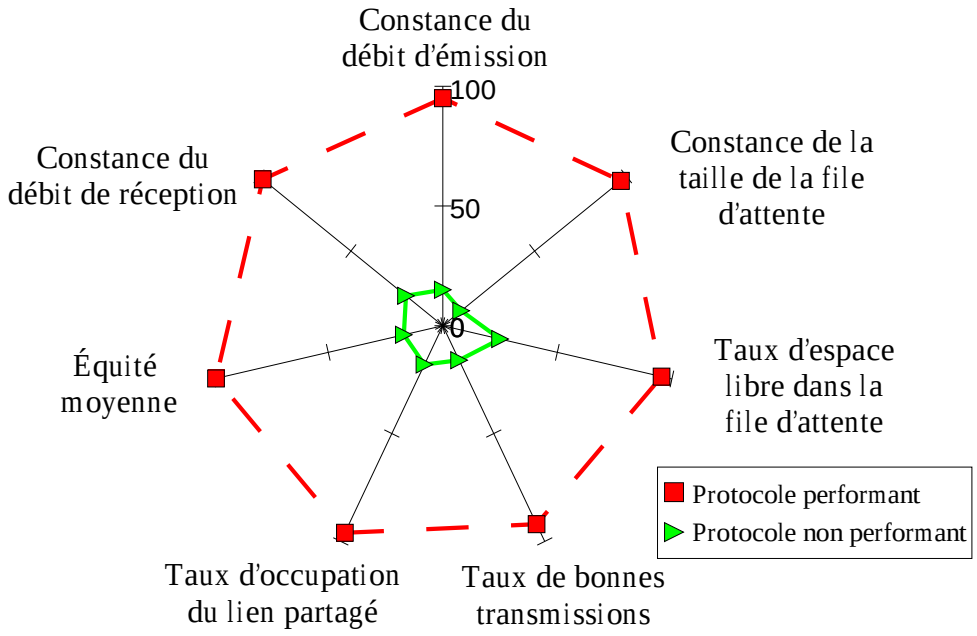

FIG. 5.5 – *Exemple de représentation avec un graphique « radar ».*

5.5 Conclusion

Dans ce chapitre, une méthode d'évaluation des performances du contrôle de congestion a été décrite. Des différents modes d'évaluation possibles (tests en environnement réel, simulations, émulations), la simulation apparaît comme étant le mode le plus adapté à notre situation. La souplesse qu'apporte ce mode d'évaluation donne la possibilité de mesurer les performances des protocoles testés dans une grande variété de situations. La plupart des études portant sur l'évaluation des performances des protocoles de transport utilisent le simulateur *ns* qui fait office de référence. L'utilisation de ce simulateur nous autorisera à comparer les résultats obtenus par Primo à ceux obtenus par d'autres protocoles de transport (Reno, Sack, Vegas, TFRC) déjà développés et validés sous *ns*.

Différents contextes de simulations ont été définis. Pour commencer, deux types de topologies ont ensuite été sélectionnés. Le premier type de topologie utilisé est un réseau composé de n paires source/destination (avec n variant de 2 à 20) et de deux routeurs intermédiaires reliés

entre eux par un lien congestionné (*cf.* figure 5.3). Ce type de topologie est utilisé pour mesurer l'influence qu'ont les différents paramètres du réseau sur le contrôle de congestion. La seconde topologie utilisée (*cf.* figure 5.4) permet de créer plusieurs congestions (de une à trois congestions) sur un même chemin. Ce type de topologie donne la possibilité d'évaluer la capacité d'un protocole à partager équitablement la bande passante dans des situations propices à l'inéquité.

Afin de définir l'impact de certains paramètres du réseau (sur le contrôle de congestion), différents ensembles de simulations ont été définis : rapport de bandes passantes, temps de propagation, files d'attente, nombre de sources (*cf.* section 5.3). Dans chaque ensemble de simulations, le paramètre du réseau dont l'impact est testé prend différentes valeurs sur une plage pré-définie alors que les autres paramètres sont fixés (*cf.* tableau 5.1 page 112). Le trafic utilisé pour ces ensembles de simulations correspond au transfert d'un fichier de taille infinie *via* FTP. Ce type de trafic permet une interprétation précise des résultats et évite les erreurs dues à une mauvaise compréhension d'un trafic plus complexe.

Les protocoles préventifs ne peuvent être mis en concurrence avec des protocoles correctifs sous peine d'obtenir des performances médiocres. C'est pourquoi, l'évaluation du protocoles Primo (de type préventif) s'est faite sous l'hypothèse qu'il serait le seul protocole implémenté sur le réseau. En effet, de par son comportement préventif, le protocole Primo est destiné à être utilisé sur un réseau privé ou sur une partie de réseau où il ne cohabiterait qu'avec lui-même. Les performances de Primo seront comparées dans le chapitre 6 à celle de TCP Reno, de TCP Sack, de TCP Vegas et de TFRC.

Afin de caractériser les performances d'un protocole, des critères d'efficacité du point de vue de l'utilisateur et de l'opérateur ont été définis. Des indicateurs numériques ont été créés afin de donner une valeur à ces critères et donc de les comparer objectivement. Les indicateurs suivant ont été définis :

- constance du débit d'émission ;
- constance du débit de réception ;
- équité moyenne ;
- taux d'occupation ;
- taux de bonne transmission ;
- taux d'espace libre dans la file d'attente ;
- constance de la taille de la file d'attente.

Une représentation graphique des valeurs des divers indicateurs facilite leur lecture (*cf.* 5.5).

La méthode d'évaluation que nous avons définie est multi-critères, elle permet ainsi d'évaluer un mécanisme de contrôle de congestion de manière complète. En effet, contrairement aux évaluations généralement rencontrées qui n'évaluent que quelques critères, cette méthode propose d'évaluer les performances d'un protocole pour l'ensemble des critères d'efficacité que nous avons recensé dans la littérature [48, 2, 19, 10, 27, 66, 31, 75, 33, 47]. Cette méthode d'évaluation propose également de tester les mécanismes de contrôle de congestion dans des situations diverses. L'évaluation des protocoles se fait donc de manière objective et non dans un environnement qui pourrait en favoriser certains.

Lors de la définition de notre méthode, nous nous sommes placés dans le cas particulier d'un protocole préventif qui n'est donc pas destiné à être mis en concurrence avec d'autres protocoles de transport sur Internet. Dans le cadre d'une étude portant sur un protocole visant à être déployé sur Internet, il serait nécessaire de compléter cette méthode. En effet, le déploiement sur

Internet implique la compatibilité avec TCP New Reno (l'une des versions les plus répandues sur Internet [90]), il faudrait donc ajouter un ensemble de simulations où le nouveau protocole serait confronté à TCP New Reno. Ainsi la méthode développée dans ce chapitre pourrait faire office de banc d'essais pour l'évaluation des nouveaux protocoles qu'ils soient préventifs ou correctifs.

Chapitre 6

Évaluation des performances

Dans ce chapitre, nous présentons les résultats des simulations permettant de comparer les performances des protocoles de transport suivants : TCP Reno, TCP Sack, TCP Vegas, TFRC et Primo.

À l'exception de la section 6.7, les résultats présentés sont illustrés par des graphiques « radar » (*cf.* figure 5.5, page 118).

Les sections 6.1 à 6.6 proposent une comparaison des performances en fonction d'un paramètre du réseau (bande passante, type et taille de la file d'attente, temps de propagation, nombre de sources). Dans chacune de ces sections, les résultats sont décomposés en deux ensembles : ceux obtenus pour des simulations sans pertes d'acquittements et ceux obtenus pour des simulations avec pertes d'acquittements.

Dans la section 6.7, les résultats obtenus dans un environnement multi-congestionné sont présentés. Ils sont illustrés par les courbes de débit de réception de chaque connexion. L'analyse de ces courbes permet d'évaluer visuellement l'équité et la stabilité des protocoles testés.

6.1 Influence de la bande passante

Dans cette section, nous évaluons l'influence du rapport de bandes passantes entre les liens entrants (liens situés entre les sources et le routeur) et le lien de congestion (lien situé entre les deux routeurs). Dans cet ensemble de simulations, le rapport de bandes passantes varie de 5% à 80% alors que les autres paramètres du réseau sont fixes (*cf.* tableau 5.1, page 112).

6.1.1 Simulations sans pertes d'acquittements

Observations. Comme cela apparaît sur la figure 6.1, hormis pour les indicateurs concernant la file d'attente, les résultats obtenus par Primo, TFRC et TCP Vegas sont assez proches. Cependant, le débit d'émission de TCP Vegas semble légèrement moins constant que celui de Primo et de TFRC. Cette différence est due aux deux premières simulations, dans lesquelles le rapport de bandes passantes est respectivement égal à 5% et 10% (*cf.* annexe D). En ce qui concerne les indicateurs liés à la file d'attente, Primo et TCP Vegas ont des comportements assez différents des autres protocoles. En effet, la taux d'espace libre dans la file d'attente pour Primo et TCP Vegas est assez élevé (respectivement 91% et 85% alors qu'il est inférieur à 50% pour TCP

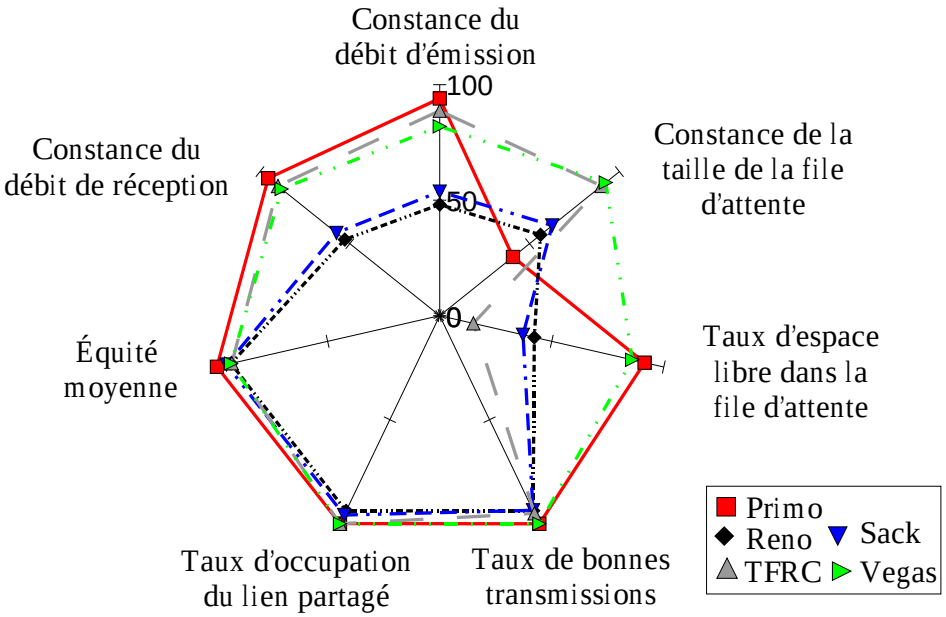

FIG. 6.1 – *Influence de la bande passante sans pertes d'acquittements.*

Reno et Sack). En revanche, les résultats obtenus par Primo et TCP Vegas pour la constance de la taille de la file d'attente (respectivement 40% et 92%) sont assez différents.

Les protocoles TCP Reno et Sack obtiennent également des résultats assez proches. Pour ces deux protocoles, les débits d'émission et de réception sont assez peu constants (entre 48% et 58%). De plus, le taux d'espace libre ainsi que la constance de la file d'attente sont très faibles. Notons que TFRC obtient lui aussi de mauvais résultats (15%) pour le taux d'espace libre dans la file d'attente. En revanche, ce protocole parvient à stabiliser la taille de la file d'attente (la constance de la taille de la file est de 90%).

Analyse. Nous avons pu voir que TCP Vegas avait du mal à conserver un débit d'émission constant lorsque le rapport de bandes passantes est faible. On peut supposer que ce phénomène est lié à son mécanisme anti-rafales qui est moins efficace lorsque la fenêtre d'émission (*i.e.*, le débit d'émission) est de petite taille.

Les bons résultats obtenus par Primo et TCP Vegas concernant le taux d'espace libre dans la file d'attente s'expliquent par leur comportement préventif. En effet, les protocoles préventifs cherchent à éviter les pertes de données en réduisant leur débit d'émission dès que la file d'attente dépasse un certain seuil. Plus ce seuil sera bas, plus le taux moyen d'espace libre dans la file d'attente sera important.

Remarque 2 *Les mauvais résultats obtenus par Primo pour la constance de la taille de la file d'attente n'impliquent pas de mauvaises performances. En effet, lorsque la file d'attente a une taille moyenne assez faible (deux à trois paquets), la moindre variation de sa taille instantanée est fortement répercutée sur l'indicateur de constance (cf. équation 5.10, page 117). Mais ces faibles variations n'ont aucune influence sur la qualité des transmissions de données. En effet, ces variations paraissent fortes si l'on observe leur valeur en pourcentage (i.e., valeur relative*

à la taille de la file d'attente), mais la gigue réelle (i.e., valeur dans l'absolu) est faible [1].

L'inconstance du débit d'émission de TCP Reno et Sack est due au fait qu'aucun de ces protocoles n'implémente de système évitant l'émission en rafale (*i.e.*, la totalité de la fenêtre de congestion est envoyée en un temps très restreint). Le passage dans les files d'attente des routeurs ne suffisant pas à atténuer l'effet de rafale, le débit de réception est également inconstant.

Le faible taux d'espace libre dans la file d'attente pour TCP Reno, Sack et TFRC s'explique par leur comportement correctif. En effet, pour réguler leur débit, ils ont besoin de saturer la file d'attente (d'où un faible taux d'espace libre) et d'observer des pertes. L'inconstance de la taille de la file d'attente pour TCP Reno et Sack est liée aux réductions drastiques du débit d'émission (*i.e.*, réduction de moitié) à chaque détection d'une perte. Chacune de ces réduction entraîne une forte diminution de la taille de la file d'attente. Ces diminutions importantes, engendrent une inconstance de la taille de la file d'attente. Contrairement à TCP Reno et Sack, TFRC parvient à stabiliser la taille de la file d'attente grâce à son système de régulation du débit d'émission qui évite les brusques variations.

Conclusion. Quel que soit le rapport de bande passante, les résultats obtenus par Primo, TCP Vegas et TFRC sont satisfaisants pour l'ensemble des indicateurs et assez proches. Cependant, TCP Vegas et Primo seront plus performants que TFRC pour un transport de données temps réel car ces protocoles parviennent à conserver un espace libre important dans la file d'attente, ce qui minimise le délai d'acheminement. Cette différence est due au comportement des protocoles : préventif pour TCP Vegas et Primo et correctif pour TFRC. Notons également qu'aucun protocole ne se distingue des autres pour ce qui est de l'équité, du taux de bonnes transmissions et du taux d'occupation. On peut donc en conclure que, quel que soit le comportement du protocole (préventif ou correctif) et sa manière de gérer le débit (fenêtre ou inter-paquets), le rapport de bandes passantes n'a pas d'influence pour ces 3 indicateurs.

Résultat 1 *Quel que soit le rapport de bande passante, les protocoles préventifs parviennent à maintenir un espace libre important dans la file d'attente.*

Résultat 2 *Les protocoles TCP Reno et Sack ne parviennent pas à maintenir les débits d'émission et de réception constants.*

6.1.2 Simulations avec pertes d'acquittements

Observations. Les pertes d'acquittements entraînent une assez forte détérioration des performances de TCP Vegas (*cf.* figure 6.2). En effet, les résultats obtenus pour la constance des débits, l'équité et le taux d'occupation du lien partagé sont assez faibles (entre 55% et 65%). Cette baisse des performances est largement due aux simulations pour lesquelles le rapport de bandes passantes est relativement faible (inférieur à 20% ce qui représente quasiment la moitié des cas testés). Comme TCP Vegas, Reno est fortement touché par les pertes d'acquittements. En effet, pour ce dernier, les résultats obtenus aux différents indicateurs chutent fortement (réduction de moitié pour la constance des débits, et d'un quart pour l'équité et le taux d'occupation).

Les résultats obtenus par TCP Sack et de TFRC sont, dans l'ensemble, légèrement moins bons que lorsqu'il n'y a pas de pertes d'acquittements. Mais, on peut tout de même observer

1. Les variations de la taille de la file d'attente sont en réalité très faibles.

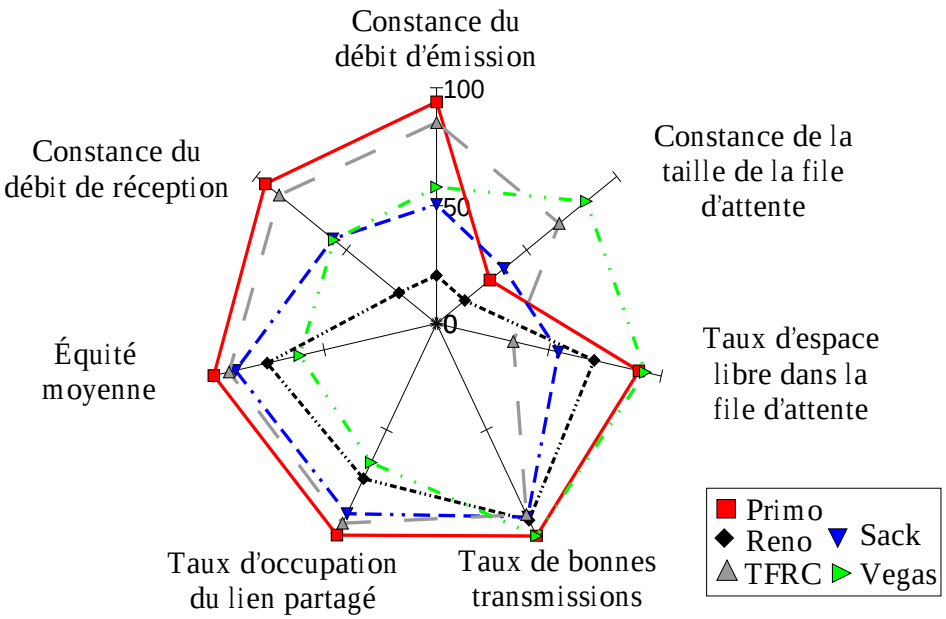

FIG. 6.2 – *Influence de la bande passante en cas de pertes d'acquittements.*

que TFRC reste performant. Pour finir, notons que les résultats de Primo ne sont pas affectés par les pertes d'acquittements, ils sont sensiblement les mêmes que ceux obtenus sans pertes d'acquittements.

Analyse. On peut supposer que la détérioration des performances de TCP Vegas est due à l'augmentation de la durée du RTT engendrée par l'apparition périodique d'une congestion dans le sens du retour. Or, si TCP Vegas voit augmenter la durée du RTT, il estimera, à tort, que la bande passante disponible a diminué. Cette « sous-estimation » entraînera une réduction inutile de la fenêtre de congestion (*i.e.*, du débit d'émission) qui conduira à la sous-utilisation des capacités du réseau. De plus, comme nous l'avons vu précédemment, lorsque la fenêtre de congestion est de petite taille (*i.e.*, le débit d'émission est faible), les performances du mécanisme anti-rafales de TCP Vegas sont diminuées. Les apparitions et disparitions successives des congestions dans le sens retour engendrent des variations de la durée du RTT et donc de la taille de la fenêtre de congestion (*cf.* fonctionnement de TCP Vegas, section 2.3). Ces deux phénomènes (fenêtre de petite taille et variante) contribuent à l'inconstance du débit d'émission. Les files d'attente des routeurs ne suffisant pas à atténuer les variations du débit d'émission, cette inconstance se répercute sur le débit de réception.

Pour TCP Reno, les pertes d'acquittements entraînent des expirations du chien de garde de retransmission et donc de fortes réductions du débit (*i.e.*, la fenêtre de congestion est réduite à MSS octets). Dans TCP Reno, la réduction de la fenêtre de congestion après l'expiration du chien de garde de retransmission est suivie d'une phase de *démarrage lent*. L'enchaînement d'une forte réduction puis d'une augmentation exponentielle de la taille de la fenêtre de congestion engendre d'importantes variations du débit d'émission. Ces variations ont pour effet d'accentuer l'inconstance du débit d'émission et donc de celui de réception. Les réductions inutiles (liées aux pertes d'acquittements) du débit d'émission engendrent une sous-utilisation du réseau (taux d'occupation du lien inférieur à 75%). De plus, ces réductions provoquent des situations

d'inéquité (environ 75%). En effet, lorsque l'une des connexions perd des acquittements, elle réduit son débit d'émission. L'autre connexion en profite alors pour augmenter le sien et donc prendre une plus large part de bande passante. Grâce à l'amélioration apportée à la gestion des acquittements, TCP Sack est beaucoup moins affecté que TCP Reno par les pertes d'acquittements.

Les pertes d'acquittements ralentissent le glissement de la fenêtre de congestion de TCP Reno et Vegas, ce qui contribue à la réduction du débit d'émission et donc à la sous-utilisation du réseau. De plus, après la perte d'une série d'acquittements, l'arrivée d'un nouvel acquittement entraînera un glissement important de la fenêtre de congestion et donc une forte augmentation du débit. Les variations de la vitesse de glissement de la fenêtre de congestion contribuent donc également à l'inconstance des débits.

L'insensibilité de TFRC et Primo aux pertes d'acquittements est liée à trois caractéristiques de leur mécanisme de contrôle de congestion :

- leur système d'évaluation de l'état de congestion du réseau n'est pas basé sur la détection de pertes de segments *via* la réception d'acquittements ;
- la régulation de leur débit d'émission est assurée par un délai inter-paquets, ce qui élimine les problèmes liés à la vitesse de glissement de la fenêtre ;
- leur mécanisme de détection de congestion est implémenté du côté du récepteur, ce qui leur permet de ne pas tenir compte des congestions se formant dans le sens du retour.

Conclusion. Lorsque le rapport de bande passante varie, qu'il y ait ou non des pertes d'acquittements, les résultats obtenus par TFRC et Primo sont assez proches, même si Primo est légèrement plus performant. En revanche, on peut observer que les performances de TCP Vegas sont fortement altérées lorsque des pertes d'acquittements surviennent. On peut supposer que cette diminution des performances est due à l'estimation du RTT dont la valeur augmente à cause des congestions se trouvant dans le sens retour. Cette augmentation du RTT entraîne des réductions de débit inutiles et donc une sous-utilisation des capacités du réseau. On peut également constater que TCP Reno est très sensible aux pertes d'acquittements : elles le poussent à réduire inutilement le débit d'émission et accentuent son inconstance. Enfin, il apparaît clairement que l'amélioration de la gestion des acquittements apportée par TCP Sack lui permet d'être peu sensible aux pertes d'acquittements.

Résultat 3 *Les performances de TCP Sack sont peu influencées par les pertes d'acquittements.*

Résultat 4 *Les protocoles utilisant une fenêtre pour réguler leur débit (notament TCP Reno et Vegas) paraissent plus vulnérables aux pertes d'acquittements que ceux utilisant un délai inter-paquets.*

Résultat 5 *Les protocoles implémentant leur mécanisme de détection des congestions du côté récepteur (Primo et TFRC) sont moins vulnérables aux pertes d'acquittements.*

6.2 Influence de la taille de file d'attente (*Drop Tail*)

Dans cette section, nous évaluons l'impact sur le contrôle de congestion de la taille de la file d'attente. La file d'attente du nœud située à l'entrée du lien de congestion implémente une

politique *Drop Tail* et sa taille varie de 5 à 50 paquets. Les autres paramètres du réseau sont fixes (*cf.* tableau 5.1 page 112).

6.2.1 Simulations sans pertes d'acquittements

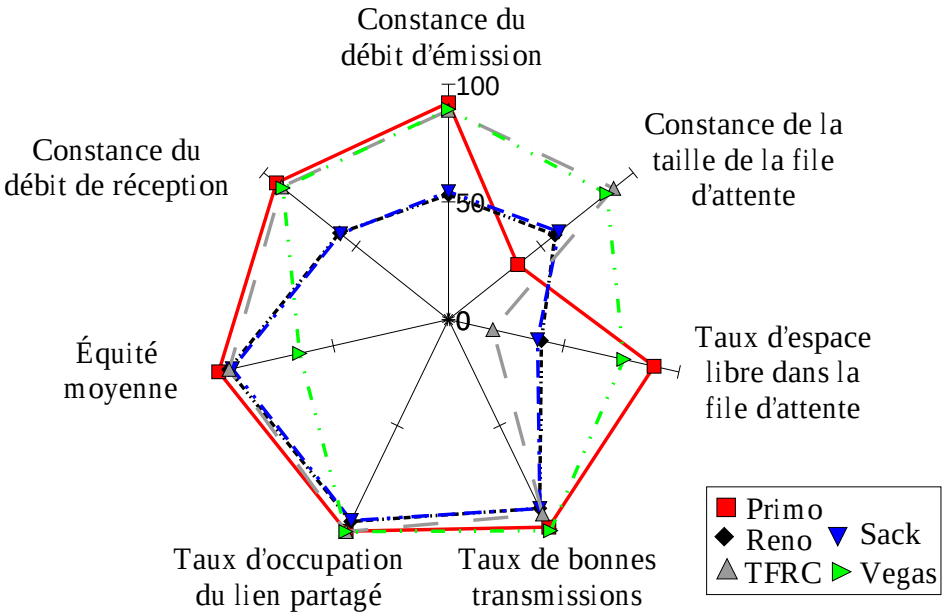

FIG. 6.3 – *Influence de la taille de file d'attente (Drop Tail) sans pertes d'acquittements.*

Observations. Dans cet ensemble de simulations, les performances des protocoles sont quasiment les mêmes que celles obtenues dans l'ensemble de simulations mesurant l'influence de la bande passante sans pertes d'acquittements (*cf.* paragraphe 6.1.1).

Quelques différences sont tout de même observables. En effet, le résultat obtenu par TCP Vegas pour l'équité est bien moins bon que celui obtenu dans l'ensemble de simulations 6.1.1 (64,5 % contre 93,15%). Ce résultat est en partie dû au cas où la file d'attente a une taille maximale de 5 paquets (*cf.* annexe D), qui fait chuter fortement la moyenne. Dans ce cas, l'équité de TCP Vegas est seulement de 2,1%, ce qui signifie qu'une seule des deux connexions parvient à émettre des données. Notons que le résultat obtenu par TCP Vegas pour le taux d'espace libre dans la file d'attente est également moins bon que dans l'ensemble précédent. Ce résultat est dû aux simulations pour lesquelles la file d'attente a une taille maximale de 5 et 10 paquets (*cf.* annexe D). Dans ces simulations, le taux d'espace libre dans la file d'attente est inférieur à 65%.

L'inconstance des débits d'émission et de réception de TCP Reno et Sack se retrouve dans cet ensemble de simulations.

Analyse. Cet ensemble de simulations montre que TCP Vegas a besoin d'une taille de file d'attente minimale supérieure à 5 paquets pour fonctionner correctement. Le système de contrôle de congestion de TCP Vegas cherche à maintenir le débit de la source à une valeur légèrement supérieure à la bande passante disponible. Cette « sur-estimation » entraîne un remplissage de la

file d'attente. Or, si ce remplissage est supérieur ou égal à la taille maximale de la file d'attente, l'une des deux connexions verra ses paquets se faire détruire. Cette destruction entraînera une réduction de son débit d'émission. La file étant remplie par les paquets de la connexion n'ayant pas réduit son débit d'émission, l'autre connexion ne pourra jamais ré-augmenter le sien sous peine de perdre de nouveau des paquets. Ces conditions conduisent donc TCP Vegas à sacrifier l'un des flots pour parvenir à maintenir le débit de l'autre. Cependant, notons que dans la réalité, il est peu probable de rencontrer des routeurs ayant une file d'attente dont la taille maximale serait de 5 paquets.

La baisse des performances de TCP Vegas pour le taux d'espace libre dans la file d'attente est également liée au maintien du débit d'émission à une valeur supérieure à la bande passante disponible. En effet, comme nous venons de le voir, si la file d'attente est petite, l'une des deux connexions la remplira en permanence. Le taux d'espace libre de la file sera par conséquent faible.

Les problèmes rencontrés par TCP Vegas auraient également pu être constatés pour Primo si ce dernier avait été paramétré de manière à être légèrement plus agressif. En effet, en rendant plus agressif Primo, la taille moyenne de la file d'attente aurait été plus importante, et les mêmes observations auraient alors été faites.

L'inconstance des débits d'émission et de réception de TCP Sack et Reno, est comme précédemment liée aux rafales d'émission qui ne sont pas évitées par ces protocoles.

Conclusion. Les résultats obtenus par TFRC et Primo sont assez proches et ces deux protocoles sont les plus efficaces pour cet ensemble de simulations où la taille de la file d'attente variait de 5 à 50 paquets. Cependant, on peut noter que Primo, grâce à son comportement préventif, obtient de meilleurs résultats que TFRC pour le taux d'espace libre dans la file d'attente ce qui confirme le résultat 1.

Notons que si l'on fait abstraction de la simulation dans laquelle la file d'attente a une taille de 5 paquets (configuration peu probable dans un environnement réel), les résultats de TCP Vegas sont également assez proches de ceux de Primo et de TFRC.

Comme précédemment, TCP Reno et Sack se différencient des autres protocoles par l'inconstance de leurs débits d'émission et de réception (*cf.* résultat 2).

6.2.2 Simulations avec pertes d'acquittements

Observations. Dans cet ensemble de simulations les pertes d'acquittements ont une très forte influence sur la constance des débits d'émission et de réception de TCP Reno et Vegas. En effet, la constance de leurs débits est inférieure à 10%. Notons également que le taux d'occupation du lien partagé ainsi que l'équité sont également très faibles (inférieurs à 25%) pour TCP Vegas.

Contrairement à l'ensemble de simulations précédent (*cf.* paragraphe 6.1.2), ici, les performances de TCP Sack sont affectées par les pertes d'acquittements. Les débits d'émission et de réception de TCP Sack sont très inconstants (la constance des débits est proche de 25%). L'équité et le taux d'occupation du lien sont également moins bons (les résultats sont inférieurs de 10% à ceux obtenus lorsqu'il n'y a aucune perte d'acquittement).

Enfin, les résultats obtenus par TFRC et Primo sont quasiment les mêmes avec ou sans pertes d'acquittements.

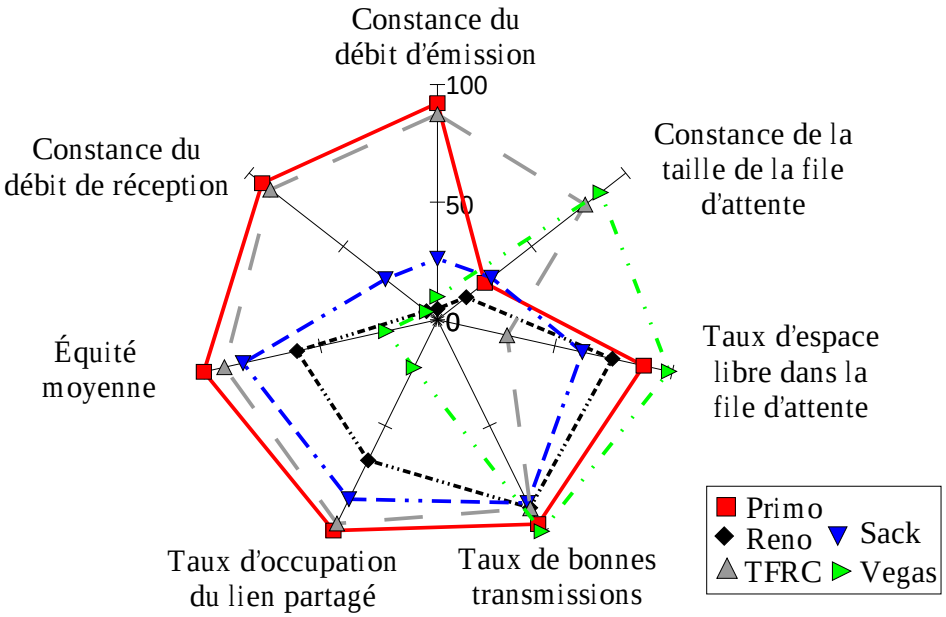

FIG. 6.4 – *Influence de la taille de file d'attente (Drop Tail) en cas de pertes d'acquittements.*

Analyse. Les observations effectuées dans cet ensemble de simulations confirment la sensibilité aux pertes d'acquittements de TCP Vegas (*cf.* paragraphe 6.1.2). Les améliorations apportées par TCP Sack à la gestion des acquittements ne suffisent pas à le rendre insensible à leur perte. En effet, comme pour TCP Reno, les pertes d'acquittements entraînent des expirations du chien de garde de retransmission de TCP Sack et donc de fortes réductions de débit. Cependant, la gestion plus fine des acquittements de TCP Sack ainsi que la redondance des informations qu'ils contiennent, lui permettent d'observer moins fréquemment qu'avec Reno des expirations inutiles du chien de garde de retransmission. Les performances de TCP Sack restent donc meilleures que celles de TCP Reno dans un tel environnement.

Cet ensemble de simulations confirme également que les protocoles dont le mécanisme de détection de congestion est situé du côté du récepteur sont peu ou pas sensibles aux pertes d'acquittements. Ainsi, TFRC et Primo parviennent à conserver leurs bonnes performances malgré les pertes d'acquittements.

Conclusion. Lorsque la taille maximale de la file d'attente (*Drop Tail*) varie entre 5 et 50 paquets et que des pertes d'acquittements surviennent, l'inconstance des débits d'émission et de réception des protocoles TCP Reno, Sack et Vegas les rend inutilisables pour une transmission temps réel. Les protocoles utilisant une fenêtre pour réguler leur débit paraissent plus vulnérables aux pertes d'acquittements que les protocoles dont la régulation du débit est basée sur un délai inter-paquets (*cf.* résultat 4). Comme nous l'avons vu précédemment, la perte d'acquittements a tendance à ralentir le glissement de la fenêtre (pour TCP Reno et Vegas) et complique la gestion de sa taille, ce qui se traduit par une chute et une inconstance du débit d'émission. Comme dans les simulations précédentes, les protocoles Primo et TFRC sont les plus performants.

6.3 Influence de la taille de file d'attente (RED)

Dans cette section, comme dans la section précédente, l'influence de la taille de la file d'attente est évaluée, mais cette fois avec une file implémentant une politique RED. La taille de la file varie de 5 à 50 paquets et les autres paramètres du réseau sont fixes (*cf.* tableau 5.1 page 112).

6.3.1 Simulations sans pertes d'acquittements

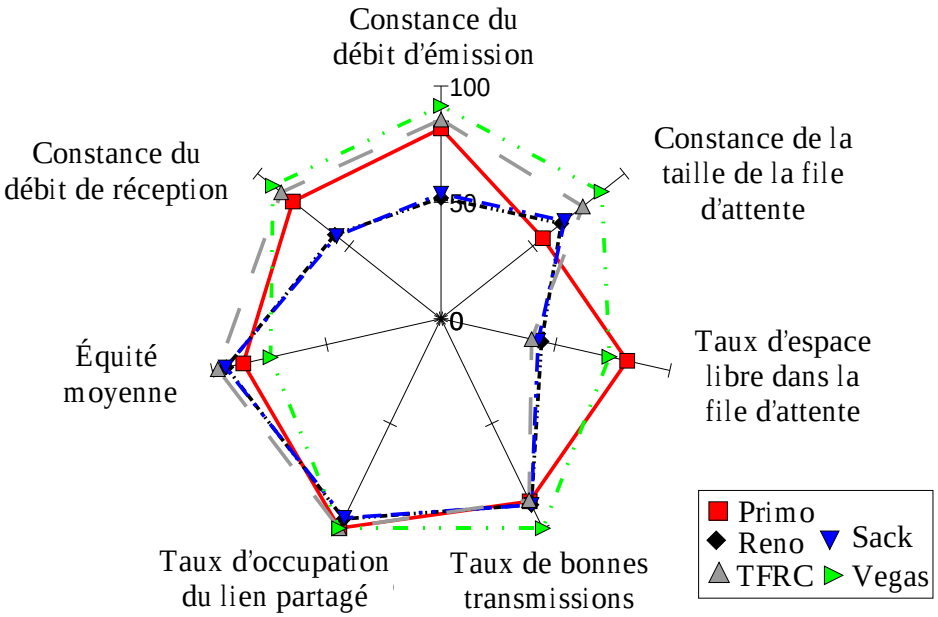

FIG. 6.5 – *Influence de la taille de file d'attente (RED) sans pertes d'acquittements.*

Observations. Les résultats sont présentés sur le graphique « radar » de la figure 6.5. Dans cet ensemble de simulations, la politique de gestion de file d'attente RED pose des problèmes à Primo lorsque la taille maximale de la file est de 5 paquets (*cf.* annexe D). En effet, les résultats obtenus pour ce cas sont très mauvais : la constance des débits d'émission et de réception, l'équité, le taux de bonnes transmissions et le taux d'espace libre dans la file d'attente sont tous inférieurs à 20%. Notons que lorsque la taille de la file d'attente est supérieure ou égale à 10 paquets, les résultats obtenus par Primo sont meilleurs que ceux obtenus par les autres protocoles (*cf.* figure 6.6). Le diagramme de la figure D intégrant tous les résultats (de 5 à 50 paquets) sous la forme d'une moyenne ; le problème mentionné se retrouve atténué.

L'équité de TCP Vegas n'est également pas très bonne (environ 75%). Les autres protocoles testés ont sensiblement les mêmes performances quelle que soit la politique de gestion de file utilisée (RED ou *Drop Tail*).

Analyse. Les mauvaises performances de Primo lorsque la file d'attente a une taille maximale de 5 paquets sont dues à la destruction aléatoire opérée par la politique RED (*cf.* paragraphe 2.5.2). En effet, dès que la file dépasse la taille moyenne de 2 paquets, les nouveaux

paquets arrivant dans la file ont une chance d'être détruits. Cette destruction prématurée et aléatoire nuit fortement au calcul du débit d'émission. En effet, la perte de paquets finit par entraîner l'expiration du chien de garde de retransmission de l'une des connexions ce qui la conduit à réduire de moitié son débit d'émission. Cette situation se répétant plusieurs fois, les performances du protocole sont fortement diminuées. Notons cependant, comme nous l'avons déjà souligné précédemment, qu'il est peu probable de rencontrer un routeur ayant une file d'attente d'une taille maximale de 5 paquets.

Remarque 3 *Le même phénomène n'est heureusement pas extrapolable à une situation où la file d'attente serait de 50 paquets et où il y aurait 20 sources (cas de figure beaucoup plus probable). En effet, avec Primo, quel que soit le nombre de connexions, la quantité de données présente dans la file d'attente est la même. Or, si la taille maximale (et le seuil de destruction préventive) de la file d'attente augmente alors que la quantité de données qui y est présente reste la même, il n'y aura plus de pertes de paquets ni d'expirations du chien de garde. Les bonnes performances de Primo seront donc préservées.*

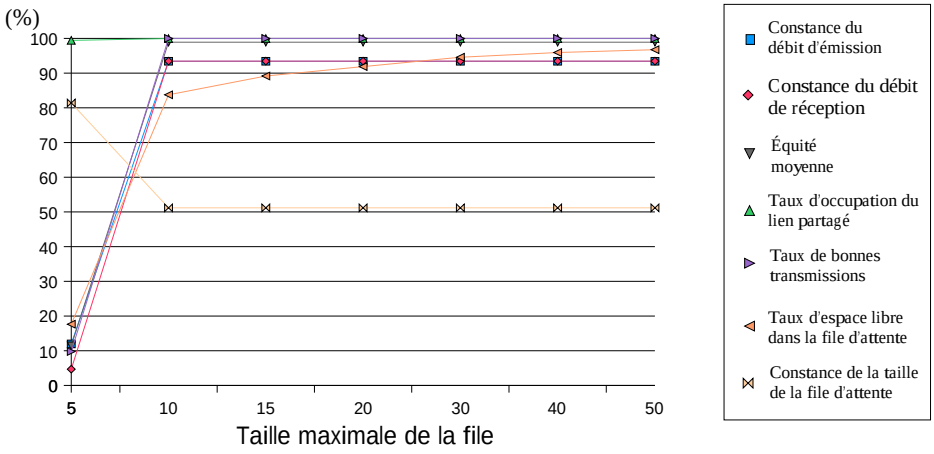

FIG. 6.6 – *Évolution des indicateurs de Primo en fonction de la taille maximale d'une file d'attente de type RED.*

Avec une politique *Drop Tail*, TCP Vegas ne parvenait pas à maintenir l'équité entre les flots lorsque la file d'attente avait une taille maximale de 5 paquets. Ici, dans la même situation, l'équité entre les flots de TCP Vegas est de 71%, ce qui laisse supposer que la politique RED, en détruisant plus fréquemment les paquets du flot ayant le débit d'émission le plus important, permet de rétablir l'équité.

Conclusion. Dans cet ensemble de simulations, lorsque la file d'attente a une taille maximale supérieure à 5 paquets, la politique RED ne semble pas avoir d'influence sur le contrôle de congestion des différents protocoles. On peut cependant supposer que, dans un environnement propice à l'inéquité, la politique RED aura une influence bénéfique sur le contrôle de congestion (*cf.* cas de TCP Vegas avec une file d'attente de 5 paquets). En effet, si deux connexions se partagent inéquitablement la bande passante disponible, la connexion ayant la plus grande part de bande passante aura plus de chance de voir ses paquets se faire détruire préventivement

par la politique RED. La destruction des paquets entraînera une réduction de débit et donc un ré-équilibrage du partage de la bande passante. Hormis pour la simulation dans laquelle la file d'attente a une taille maximale de 5 paquets (situation peu probable), les performances de Primo sont très bonnes.

6.3.2 Simulations avec pertes d'acquittements

Observations. Les résultats observés pour Primo sans pertes d'acquittements se retrouvent ici (*cf.* figure 6.7).

Comme pour les simulations où l'influence de la politique *Drop Tail* est testée, les performances de TCP Reno se détériorent lorsque des acquittements sont perdus. En revanche, celles de TCP Vegas n'en pâtissent pas. En effet, pour TCP Vegas, seule la stabilité de la file d'attente se détériore. Mais cela n'a pas d'importance car le taux d'espace libre reste important (supérieur à 90%). L'utilisation d'une politique RED pour les files d'attente semble donc avoir un effet positif sur les performances de TCP Vegas.

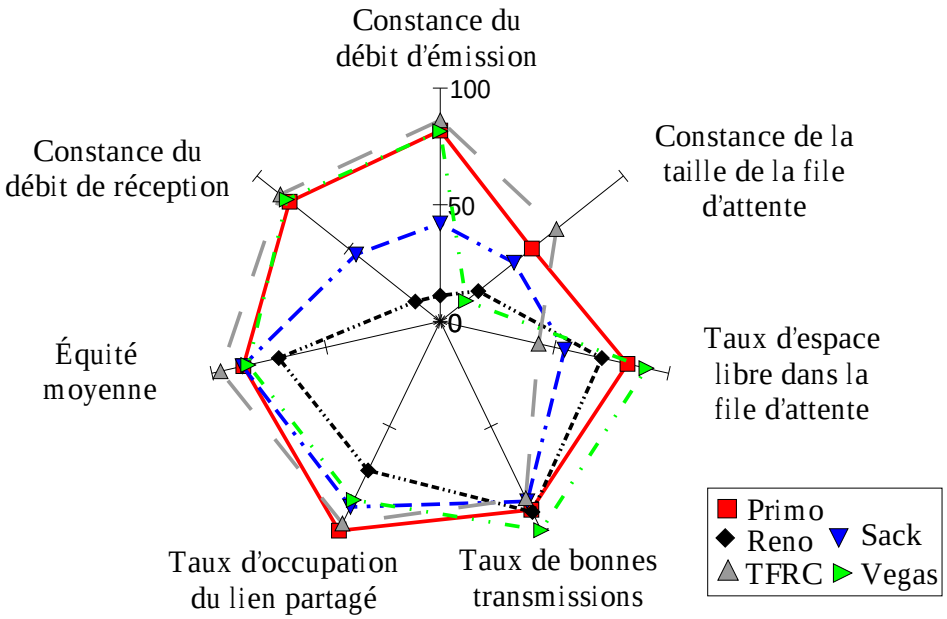

FIG. 6.7 – *Influence de la taille de file d'attente (RED) en cas de pertes d'acquittements.*

Analyse. Le protocole Primo n'étant pas influencé par les pertes d'acquittements, les résultats obtenus précédemment sont sensiblement les mêmes que ceux obtenus dans cet ensemble de simulations. Ainsi, Primo obtient de bons résultats lorsque la taille maximale de la file d'attente est supérieure à 5 paquets.

Il est surprenant de constater que, dans ces conditions d'évaluation, lorsque des pertes d'acquittements surviennent, les résultats de TCP Vegas s'améliorent. En effet, il semble qu'ici l'augmentation de la durée du RTT, engendrée par les congestions se formant dans le sens du retour, permet à TCP Vegas d'améliorer l'équité entre les flots. Dans ces simulations, nous avons pu constater que TCP Vegas modifiait plus fréquemment la taille de sa fenêtre de congestion

lorsque des acquittements se perdent. On peut supposer que ces ajustements successifs de la fenêtre de congestion (*i.e.*, du débit d'émission) permettent d'améliorer l'équité.

Conclusion. Hormis l'amélioration des performances de TCP Vegas, les résultats obtenus dans cet ensemble de simulations sont sans surprise. Comme pour les autres simulations, les performances de TCP Reno se dégradent fortement lorsque des pertes d'acquittements surviennent, alors que les performances des autres protocoles (hormis TCP Vegas) restent sensiblement les mêmes (*cf.* résultats 5 et 3).

Il est intéressant de noter que, quelle que soit la politique utilisée (RED ou *Drop Tail*), à partir d'une certaine taille maximale de la file d'attente, les résultats obtenus par les protocoles préventifs ne changent plus. En effet, lorsque la taille maximale de la file d'attente dépasse 10 paquets, les résultats obtenus par TCP Vegas et Primo aux divers indicateurs (hormis ceux directement liés à la taille maximale de la file) reste constants (*cf.* annexe D et figure 6.6). Cette caractéristique tient au fait que ces protocoles sont de type préventif. Ainsi, ils n'ont pas besoin de saturer la file d'attente du lien partagé pour en estimer la bande passante, comme le font les protocoles correctifs.

Résultat 6 *À partir d'un certain seuil, l'augmentation de la taille maximale de la file d'attente n'a plus d'influence sur les performances des protocoles préventifs (Primo et TCP Vegas).*

6.4 Influence des temps de propagation homogènes

Dans cette section, l'influence de la durée du délai d'aller retour (RTT) minimum est évaluée. Le RTT minimum (*i.e.*, somme des temps de propagation) des deux connexions varie de 14 ms à 404 ms, les autres paramètres du réseau sont fixes (*cf.* tableau 5.1 page 112).

6.4.1 Simulations sans pertes d'acquittements

Observations. Dans cet ensemble de simulations (*cf.* figure 6.8), l'augmentation de la durée du RTT entraîne une dégradation de la constance des débits d'émission et de réception de TCP Vegas .

Pour TCP Reno et Sack, lorsque le RTT dépasse 200 ms, l'équité commence à se détériorer. Notons que, de manière générale, pour ces deux protocoles, plus le RTT est long, plus les performances se dégradent. La constance du débit d'émission en est l'exemple le plus flagrant : pour un RTT de 14 ms, cet indicateur est proche de 50% alors que pour un RTT de 404 ms, il est inférieur à 2%.

Les performances des protocoles TFRC et Primo restent très bonnes quels que soient les temps de propagation (*i.e.*, la durée du RTT) utilisés.

Analyse. Pour TCP Reno et Sack, on peut remarquer que plus le temps de propagation est important, plus la constance des débits d'émission et de réception se dégrade. Ce phénomène est lié à l'utilisation d'une fenêtre sans mécanisme anti-rafales. En effet, plus le RTT est long, plus la phase d'attente des acquittements permettant le glissement de la fenêtre sera longue. Or, durant cette phase d'attente, aucun paquet n'est émis et le débit d'émission chute à 0 kbit/s.

L'alternance des phases d'émission et de « silence » conduit à une forte instabilité des débits d'émission et donc de réception.

La dégradation de la constance des débits d'émission et de réception de TCP Vegas est due à son mécanisme anti-rafales. Il semble que plus le temps de propagation est important, moins ce mécanisme est efficace. En effet, l'augmentation du temps de propagation entraîne une augmentation du délai « inter-segments[2] » de TCP Vegas, et donc des périodes de « silence » entre chaque émission de deux segments (*cf.* paragraphe 2.3.2). Ce phénomène, bien qu'il soit moins marqué, est assimilable à celui observé pour TCP Reno et Sack.

Les protocoles n'utilisant pas de fenêtre (*i.e.*, TFRC et Primo) ne semblent pas être perturbés par l'augmentation du temps de propagation. Le fait de ne pas avoir à attendre le glissement de la fenêtre permet à ces protocoles de conserver un délai inter-paquets constant et ainsi de lisser leur débit d'émission.

Conclusion. Lorsque les temps de propagation entre une source et un récepteur sont importants, il semble plus judicieux d'utiliser un protocole dont la gestion du débit d'émission est assurée grâce à un délai inter-paquets. Cependant, l'utilisation de tels protocoles peut engendrer des pertes massives (si leur chien de garde de retransmission est mal calibré) en cas de grave congestion. En effet, les protocoles n'utilisant pas de fenêtre ne peuvent pas limiter la quantité de données en transit. Il faut donc s'assurer qu'ils détecteront rapidement les congestions afin de ne pas saturer le réseau lorsqu'il est déjà congestionné.

Résultat 7 *Les protocoles utilisant un délai inter-paquets pour réguler le débit (Primo et TFRC) parviennent à le maintenir constant, même lorsque les temps de propagation sont très importants.*

Résultat 8 *Le mécanisme anti-rafales de TCP Vegas paraît devenir inefficace à mesure que les temps de propagation augmentent.*

6.4.2 Simulations avec pertes d'acquittements

Observations. Comme dans tous les autres ensembles de simulations, les performances de TCP Vegas et Reno sont fortement affectées par les pertes d'acquittements (*cf.* figure 6.9). Comme dans l'ensemble de simulations du paragraphe 6.2.2, TCP Vegas est le protocole le plus fortement touché par ces pertes. En effet, quels que soient les délais de propagation utilisés, l'une des deux connexions ne parvient pratiquement plus à émettre de données. Par conséquent, l'équité devient très faible (inférieur à 30%). L'incapacité d'émettre de l'une des sources entraîne une sous utilisation des capacités du réseau (*i.e.*, taux d'occupation du lien congestionné proche de 30%).

Comme dans les autres ensembles de simulations, TCP Sack n'est que légèrement perturbé par les pertes d'acquittements : ses performances (notamment pour l'équité et le taux d'occupation du lien partagé) sont un peu moins bonnes que lorsque tous les acquittements lui parviennent.

Enfin, les performances de TFRC et de Primo sont peu ou pas modifiées par les pertes d'acquittements, ce qui tend à confirmer que ces protocoles y sont insensibles.

2. Délai durant lequel TCP Vegas émettera (si la fenêtre d'émission lui permet) 2 MSS octets de données.

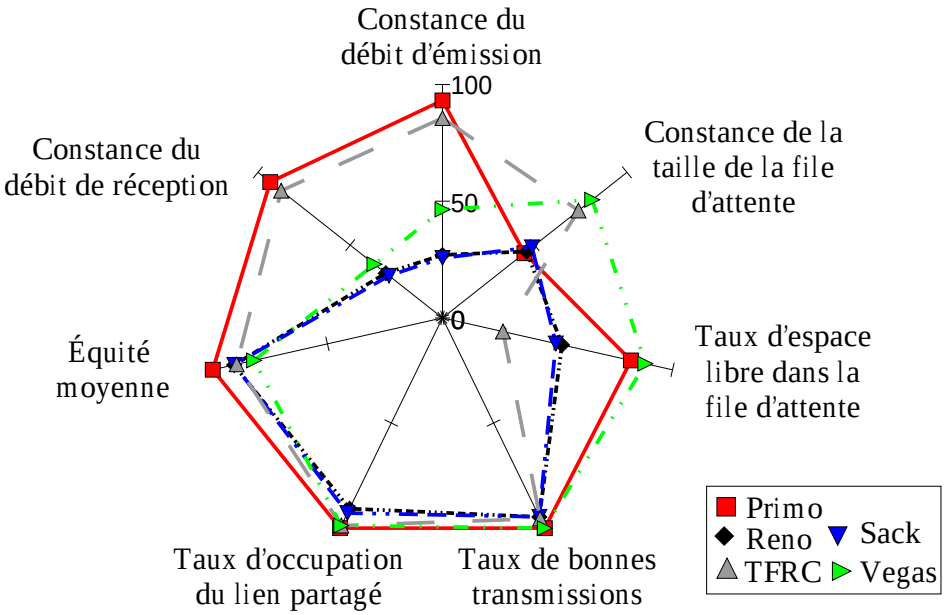

FIG. 6.8 – *Influence des temps de propagation homogènes sans pertes d'acquittements.*

Analyse. La dégradation des performances de TCP Vegas confirme sa sensibilité aux pertes d'acquittements. L'augmentation de la durée du RTT engendrée par les congestions dans le sens du retour entraîne une réduction importante du débit de l'une des sources. Ceci se traduit par une sous-utilisation des capacités du réseau et l'inéquité du partage de la bande passante.

Comme dans tous les autres ensembles de simulations, les performances de TCP Reno se dégradent fortement lorsque des pertes d'acquittements surviennent. En effet, les expirations inutiles et répétées de son chien de garde de retransmission engendrent de fortes réductions de la taille de la fenêtre de congestion et donc du débit d'émission.

Pour Primo, l'inconstance de la taille de la file d'attente, comme dans les autres ensembles de simulations, est sans conséquence pour la qualité de la transmission. Mais, comme nous l'avons vu dans le paragraphe 6.1.1, la taille moyenne de la file d'attente étant très faible (*i.e.*, espace libre important), la gigue réelle du délai d'acheminement (*i.e.*, de la taille de la file d'attente) n'aura pas d'effet sur la transmission des données.

Conclusion. Ces simulations confirment les résultats 4 et 5 obtenus précédemment : les protocoles TCP Reno et Vegas sont très sensibles aux pertes d'acquittements alors que TFRC et Primo ne paraissent pas être affectés. Le protocole TCP Sack, grâce à sa gestion plus fine des acquittements, n'est que légèrement touché par leur perte (*cf.* résultat 3), ce qui lui permet de maintenir ses performances.

6.5 Influence des temps de propagation hétérogènes

Dans cette section, nous évaluons l'influence sur le contrôle de congestion de l'hétérogénéité des délais de propagation. Le RTT minimum (*i.e.*, somme des temps de propagation) de la première connexion est fixé à 24 ms et celui de la seconde varie de 24 ms à 122 ms. Les autres

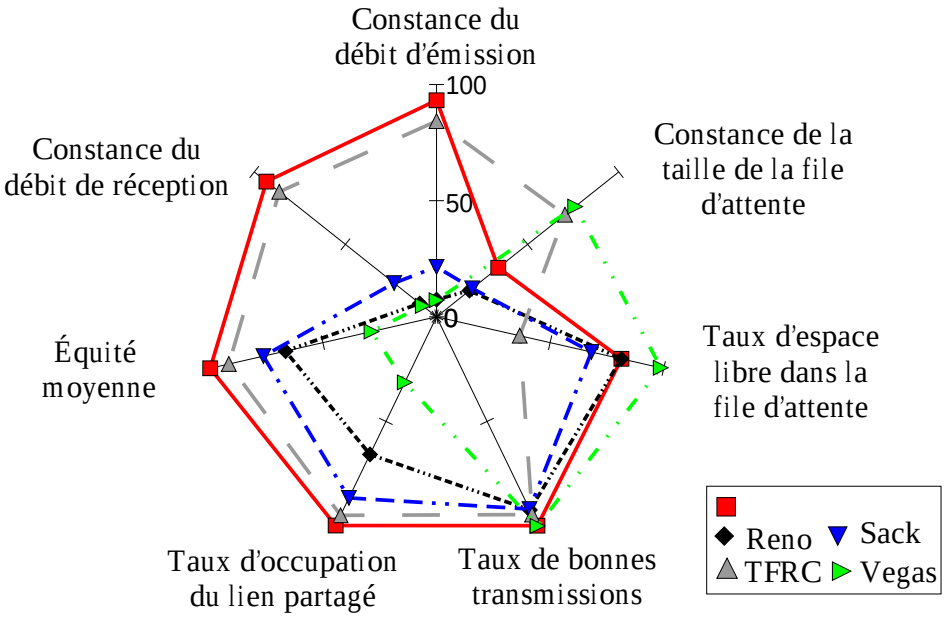

FIG. 6.9 – *Influence des temps de propagation homogènes en cas de pertes d'acquittements.*

paramètres du réseau sont fixes (*cf.* tableau 5.1 page 112).

6.5.1 Simulations sans pertes d'acquittements

Observations. Contrairement à ce que nous avons observé dans les simulations précédentes, TCP Reno et Sack ont du mal à maintenir l'équité entre les flots (*cf.* figure 6.10). De même, l'équité entre les flots utilisant TCP Vegas n'est pas très bonne (proche de 75% et pouvant descendre jusqu'à 34% dans certains cas).

Les protocoles TFRC et Primo ont des performances assez proches des précédentes. La différence des temps de propagation entre les deux connexions ne semble pas les influencer.

Analyse. L'inéquité des protocoles TCP Reno et Sack est due à leur technique de contrôle de congestion, qui est liée à la fréquence de réception des acquittements (*i.e.*, la réception d'un acquittement entraîne l'augmentation de la taille de la fenêtre de congestion). L'attente entre l'émission d'un paquet et la réception de son acquittement n'étant pas la même pour les deux connexions, une forte inéquité entre les flots est inévitable. En effet, la connexion ayant le RTT le plus court recevra plus rapidement ses acquittements et pourra ainsi augmenter la taille de sa fenêtre de congestion plus fréquemment. Cette augmentation rapide de la taille de la fenêtre de congestion de l'une des deux connexions aura pour effet de lui faciliter la récupération de bande passante et donc favorisera l'inéquité.

Bien que la régulation de la taille de la fenêtre de congestion de TCP Vegas ne soit pas liée à la fréquence de réception des acquittements, l'équité entre les flots se dégrade lorsqu'ils n'ont pas les mêmes temps de propagation. En effet, la connexion ayant le RTT le plus court recevra ses acquittements plus rapidement que l'autre connexion. Or, même si les deux connexions ont une fenêtre de congestion de même dimension, la fenêtre de la connexion recevant plus rapidement ses acquittements glissera plus fréquemment que l'autre. Cela permettra à la connexion

dont la fenêtre glisse le plus fréquemment d'émettre des données plus souvent et d'avoir ainsi un débit plus important. Notons que ce phénomène est également observable pour TCP Reno et Sack.

Conclusion. Cet ensemble de simulations met en évidence que l'utilisation d'une fenêtre a un effet négatif sur l'équité lorsque les connexions ont des temps de propagation différents.

Il semble donc plus judicieux d'utiliser des protocoles dont la régulation de débit est basé sur un délai inter-paquets lorsque différents flots partageant un même lien ont des temps de propagation hétérogènes. Cette situation étant fréquente, nous retrouvons là l'une des justifications des recherches sur les protocoles sans fenêtre [37]. Notons cependant que l'utilisation d'un délai inter-paquets ne garantit pas que l'équité sera respectée lorsque les délais de propagation sont hétérogènes. En effet, l'équité dans une telle situation dépend également de la manière dont est gérée l'augmentation (et la réduction) du débit. Ainsi, il est indiqué dans [101] que le protocole RAP, qui utilise un délai inter-paquets, a un comportement similaire à celui de TCP Reno (donc inéquitable) lorsque les délais de propagation sont hétérogènes.

Résultat 9 *Lorsque les délais de propagation sont hétérogènes, l'utilisation d'un délai inter-paquets pour réguler le débit d'émission favorise l'équité.*

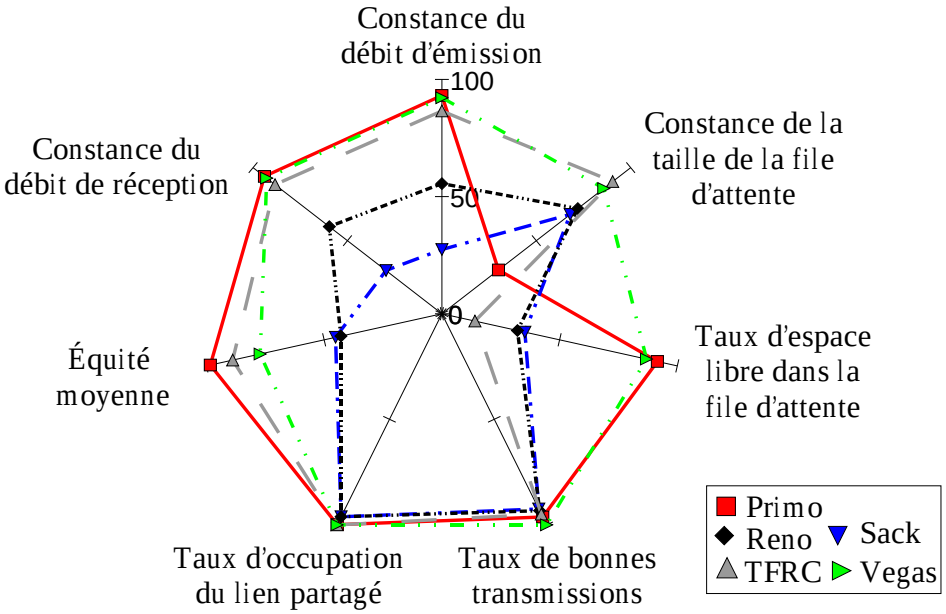

FIG. 6.10 – *Influence des temps de propagation hétérogène sans pertes d'acquittements.*

6.5.2 Simulations avec pertes d'acquittements

Observations. Pour TCP Reno et Sack, dans cet ensemble de simulations, l'équité est meilleure lorsque des pertes d'acquittements surviennent (*cf.* figure 6.11). En revanche, les pertes d'acquittements ne profitent pas à TCP Vegas dont les performances chutent fortement.

Les performances de TFRC et Primo, comme dans tous les autres ensembles de simulations, sont peu ou pas influencées par les pertes d'acquittements.

Analyse. L'amélioration de l'équité des protocoles TCP Reno et Sack peut s'expliquer par le fait que la connexion ayant le RTT le plus court émet plus de données. Elle reçoit donc plus d'acquittements que l'autre connexion, ce qui augmente ainsi ses chances d'en perdre. Or, pour ces deux protocoles, les pertes d'acquittements entraînent des réductions de débit ; la connexion ayant la plus large part de bande passante réduira plus fréquemment son débit que l'autre. Cette réduction fréquente du débit permet de ré-équilibrer le partage de la bande passante entre les deux connexions.

Conclusion. Bien que les performances en terme d'équité s'améliorent pour TCP Reno et Sack, les protocoles les plus performants restent Primo et TFRC. La sensibilité de TCP Vegas aux pertes d'acquittements est une fois de plus mise en évidence.

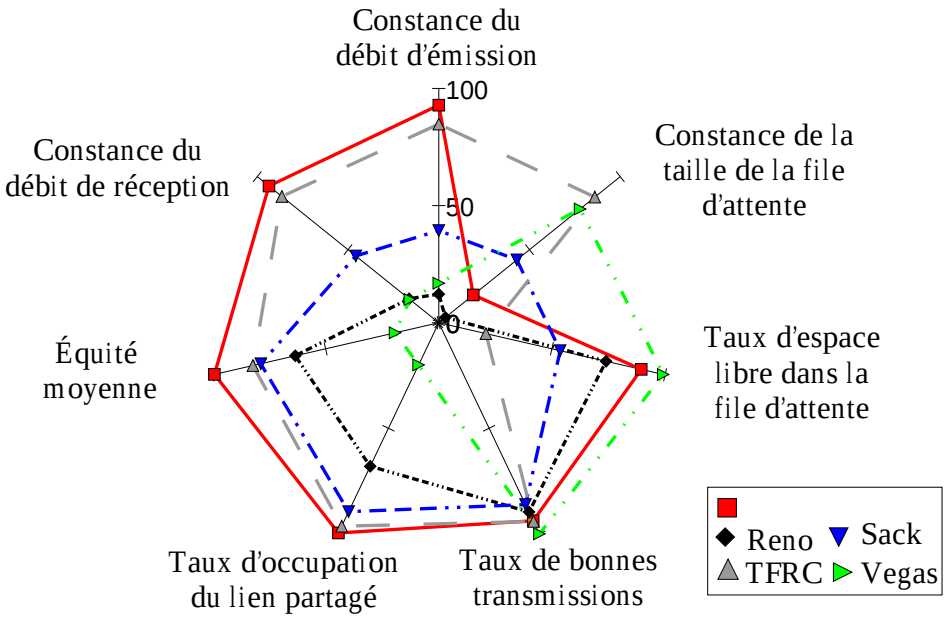

FIG. 6.11 – *Influence des temps de propagation hétérogène en cas de pertes d'acquittements.*

6.6 Influence du nombre de sources

Dans cette section, nous évaluons l'influence du nombre de connexions sur le contrôle de congestion. La topologie est composée de n nœuds sources reliés à n nœuds destinations par l'intermédiaire de deux routeurs, eux-même reliés entre eux par le lien de congestion (*cf.* figure 5.3, page 108). Le nombre de sources et de récepteurs (n) varie de 2 à 20, alors que les autres paramètres du réseau restent fixes (*cf.* page 112).

6.6.1 Simulations sans pertes d'acquittements

Observations. Dans cet ensemble de simulations (*cf.* figure 6.12), une fois de plus TCP Vegas éprouve des difficultés à maintenir l'équité entre les flots (l'équité est proche de 75%). On peut noter que le taux d'espace libre dans la file d'attente souffre également de l'augmentation du

nombre de sources. Plus il y a de sources, plus la file se remplit (*cf.* annexe D). En revanche, les nuisances engendrées par l'augmentation de la taille de la file d'attente sont atténuées par sa constance (*i.e.*, la constance de la taille de la file d'attente est supérieure à 95%).

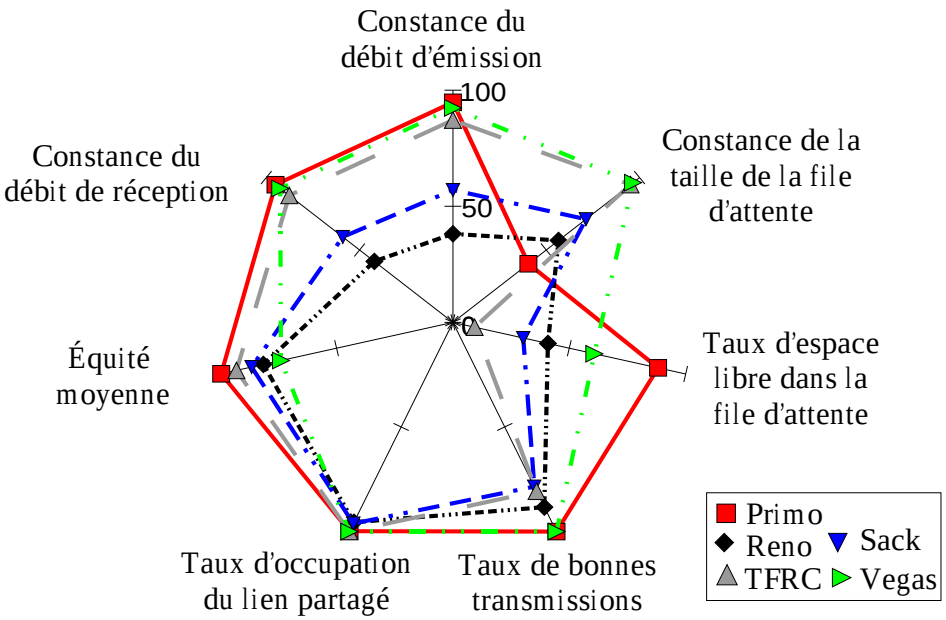

FIG. 6.12 – *Influence du nombre de sources sans pertes d'acquittements.*

On peut remarquer que, pour les protocoles correctifs, le nombre de sources influence légèrement les taux de bonnes transmissions, notamment pour TFRC, dont l'indicateur chute à 80%. En effet, plus le nombre de sources augmente, plus les pertes de données sont fréquentes pour TFRC.

Dans cet ensemble de simulations, Primo est le seul protocole dont les performances ne sont pas affectées par l'augmentation du nombre de sources.

Analyse. Pour TCP Vegas, on peut supposer que la diminution du taux d'espace libre dans la file d'attente à mesure que le nombre de sources augmente est due à la manière dont la taille de la fenêtre de congestion est gérée. En effet, comme nous l'avons vu dans le paragraphe 2.3.2, TCP Vegas augmente de quelques octets la taille de sa fenêtre de congestion afin de maintenir un débit légèrement supérieur à la bande passante disponible. Or, lorsque le nombre de connexions croît, l'augmentation arbitraire de la taille de la fenêtre de congestion peut se répercuter sur l'espace disponible dans la file d'attente.

Pour les protocoles correctifs, la diminution du taux de bonnes transmissions à mesure que le nombre de sources augmente semble logique. Les protocoles correctifs ont besoin de saturer périodiquement les files d'attente des routeurs avant de réduire leur débit d'émission. Durant la période où la file d'attente est saturée (*i.e.*, en période de congestion), les paquets y arrivant sont détruits. Or, plus le nombre de sources est élevé, plus le nombre de paquets arrivant au routeur lors des congestions le sera. Ceci explique l'augmentation du taux de pertes (*i.e.*, la diminution du taux de bonnes transmissions).

Conclusion. La détérioration des performances lorsque le nombre de sources augmente laisse supposer que certains phénomènes, non observés dans ces différents ensembles de simulations, risquent d'apparaître lors du passage à l'échelle.

Pour TCP Vegas l'augmentation du nombre de sources engendrent une diminution de l'espace libre dans la file d'attente. Primo reste donc le seul protocole à maintenir un espace libre important dans la file d'attente (proche de 90%). Il est également le seul protocole à ne pas être influencé par l'augmentation du nombre de sources.

6.6.2 Simulations avec pertes d'acquittements

Observations. Une fois de plus, lors de l'apparition de pertes d'acquittements, l'équité de TCP Vegas (proche de 60%) est pénalisée (*cf.* figure 6.13). En revanche, on peut noter que la constance des débits d'émission et de réception de TCP Reno habituellement très perturbée par les pertes d'acquittements l'est moins lorsqu'il y a beaucoup de sources (*cf.* annexe D).

Comme pour tous les autres ensembles de simulations, les protocoles TCP Sack, TFRC et Primo sont peu ou pas touchés par les pertes d'acquittements.

Analyse. Les résultats obtenus dans ce dernier ensemble de simulations confirment les précédents. Bien que l'effet des pertes d'acquittements sur TCP Reno et Vegas soit atténué dans cet ensemble de simulations, ces deux protocoles y restent sensibles. L'insensibilité de TCP Sack, TFRC et Primo aux pertes d'acquittement est confirmée (*cf.* résultats 5 et 3).

Conclusion. Dans cet ensemble de simulations, comme dans tous les autres, TFRC et Primo sont les protocoles les plus performants. L'augmentation du nombre de sources semble atténuer l'effet des pertes d'acquittements sur les protocoles qui y sont sensibles. En effet, Les résultats obtenus par TCP Reno et Vegas sont quasiment les mêmes qu'il y ait ou non des pertes d'acquittements.

6.7 Équité dans un environnement multi-congestionné

Dans cet ensemble de simulations, la topologie utilisée est composée de cinq paires source/destination et de quatre routeurs (*cf.* figure 5.4, page 109). Dans les cinq paires source/destination, on peut distinguer trois groupes de connexions. Les connexions A et B forment le premier groupe, leurs paquets ne traversent que deux routeurs (*i.e.*, une congestion) avant d'arriver à destination. Le second groupe est composé des connexions C et D dont les paquets traversent trois routeurs (*i.e.*, deux congestions) avant d'atteindre leur destination. Enfin, le dernier groupe est formé par la connexion E, dont les paquets passent par les quatre routeurs (*i.e.*, trois congestions) avant d'arriver à destination.

Les résultats obtenus pour différents scénarios de démarrages des connexions sont commentés dans le paragraphe 6.7.1. Ces résultats sont ensuite analysés dans le paragraphe 6.7.2. Nous tirons des conclusions au paragraphe **??**

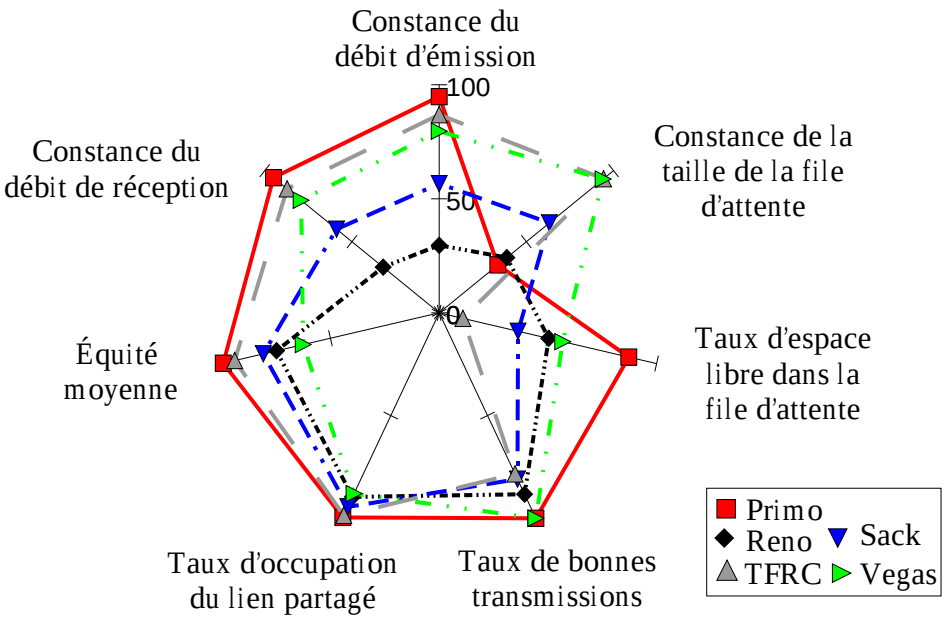

FIG. 6.13 – *Influence du nombre de sources en cas de pertes d'acquittements.*

6.7.1 Observations

Premier scénario de démarrage. Pour commencer, une première série de simulations où toutes les sources commencent à émettre simultanément a été effectuée (*cf.* figure 6.14). Les courbes représentant le débit de réception de chaque connexion nous montrent l'instabilité des sources utilisant les protocoles TCP Reno ou TCP Sack. Malgré cette instabilité, l'inéquité de ces deux protocoles est cependant observable. En effet, au bout d'une centaine de secondes de simulation, les connexions A et D prennent le dessus et commencent à monopoliser la bande passante. Cette domination est très nette quand le protocole TCP Sack est utilisé.

Les résultats obtenus par TFRC sont assez bons bien que la part de bande passante allouée à la connexion E soit faible. Hormis pour la connexion E, l'équité moyenne est assez bonne : les connexions A, B, C et D obtiennent une part équitable de bande passante. On peut également noter que TFRC ne parvient pas à stabiliser le débit de réception des différentes connexions : après 200 s de simulation, des variations de débit de l'ordre de 30% sont encore observables. Cette incapacité à conserver un débit constant pourrait nuire à la stabilité du réseau et donc à la qualité de réception des données.

Pour la stabilité du débit de réception, le protocole TCP Vegas est le plus efficace. En effet, le débit de réception de chaque connexion est quasiment constant sur toute la durée de la simulation. Malheureusement, le partage de la bande passante n'est pas très équitable : la connexion A dispose d'une part de bande passante presque trois fois supérieure à celle dont dispose la connexion E et aucune des connexions ne possède la même quantité de bande passante.

En utilisant Primo, le partage de la bande passante entre les connexions A, B, C et D est relativement équitable. En revanche, la connexion E dispose de moitié moins de bande passante que les autres connexions. Notons que la constance du débit de réception obtenue avec Primo est quasiment aussi bonne que celle obtenue avec TCP Vegas.

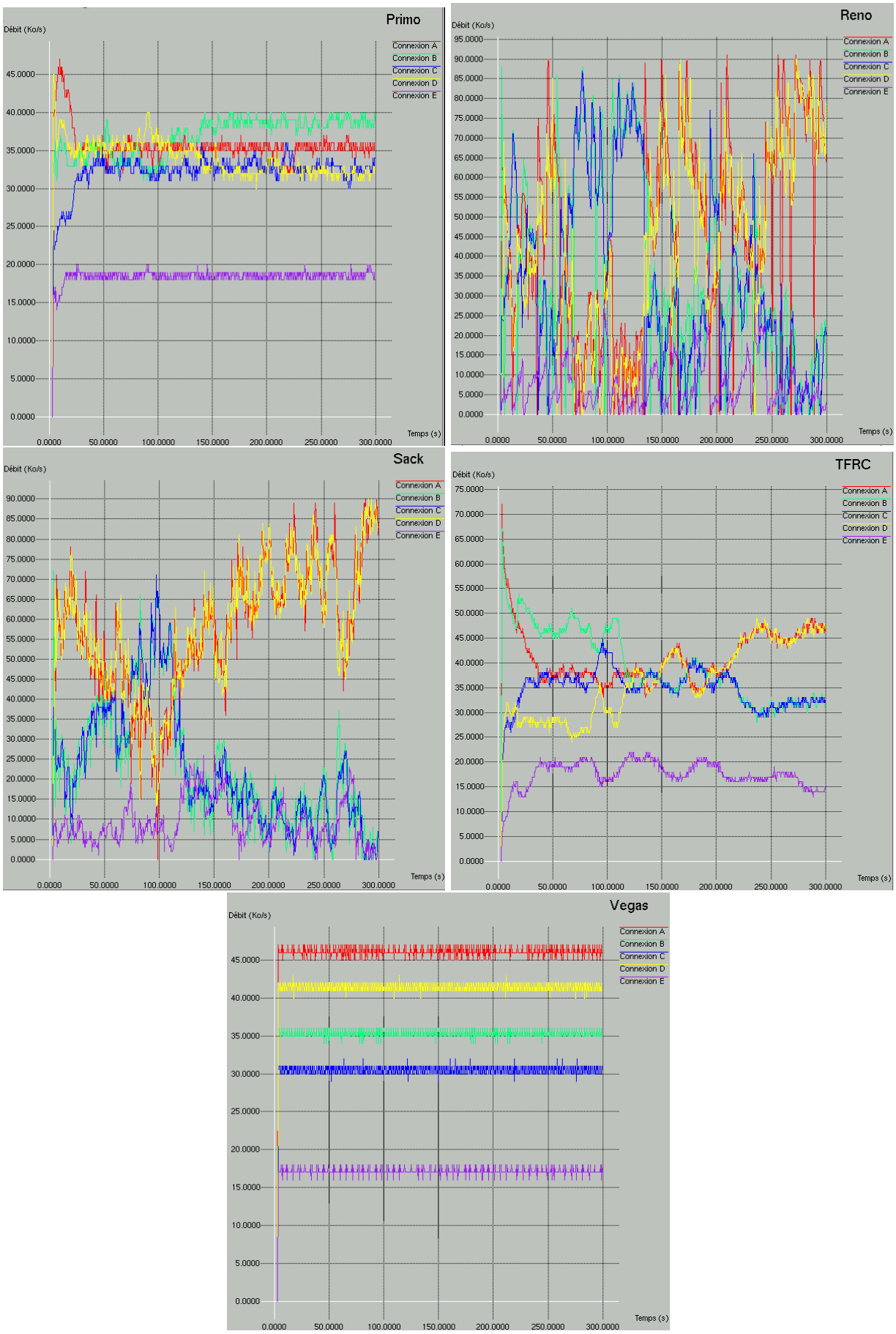

FIG. 6.14 – *Débit de réception dans un environnement multi-congestioné : démarrage simultané de toute les sources.*

Second scénario de démarrage. Dans la seconde série de simulations (*cf.* figure 6.15), le groupe de connexions ne traversant qu'une congestion (connexions A et B) commence à émettre seul sur le réseau. Puis, dix secondes plus tard, le second groupe (connexions C et D) commence à émettre. Enfin, après vingt secondes de simulation, la connexion E commence à émettre.

Comme dans le premier scénario, les protocoles TCP Reno et Sack ne parviennent pas à stabiliser leur débit de réception. Le partage des ressources est également très mauvais : pour TCP Sack, les connexions A et D prennent rapidement le dessus et utilisent une grande part de la bande passante disponible. Pour TCP Reno, les connexions A, B, C et D monopolisent à tour de rôle la bande passante, alors que la connexion E n'en dispose que d'une très faible part.

Le protocole TFRC éprouve également des difficultés à partager équitablement la bande passante entre les différentes connexions. En effet, on peut voir que les connexions A et D prennent rapidement une part importante de bande passante. Les connexions B, C et E ne parviennent à obtenir qu'une part de bande passante environ deux fois inférieure à celle des connexions A et D. Notons que, quelle que soit la connexion, la stabilité du débit de réception est assez bonne. Ce protocole serait donc utilisable dans le cadre d'une transmission temps réel.

Les résultats obtenus par TCP Vegas et Primo sont excellents. La stabilité du débit de réception est quasiment parfaite pour les deux protocoles. Les quelques variations observables sont dues à la recherche de bande passante disponible. Pour TCP Vegas, l'équité est très bonne, bien que la connexion E ait une part de bande passante légèrement inférieure à celle des autres connexions. Pour Primo, l'équité est parfaite et toutes les connexions ont la même quantité de bande passante allouée.

Troisième scénario de démarrage. Dans la troisième série de simulations (*cf.* figure 6.16), le groupe de connexions ne traversant que deux congestions (connexions C et D) commence à émettre seul sur le réseau. Puis, dix secondes plus tard, le groupe de connexions ne traversant qu'une congestion (connexions A et B) commence à émettre. Enfin, après vingt secondes de simulation, la dernière connexion (E) commence à émettre.

Les résultats obtenus pour cette série de simulations sont très proches de ceux obtenus dans le second scénario. On peut tout de même noter que l'inéquité des protocoles TCP Reno et Sack est encore plus marquée dans cette série. Pour TCP Sack, les connexions A et D prennent rapidement et largement le dessus en monopolisant la bande passante du lien congestionné. Pour TCP Reno, on peut observer que, par moments, les connexion A et D sont quasiment seules à émettre sur le réseau (par exemple entre t = 100 s et t = 130 s), les autres connexions ne disposant que d'une part très faible de la bande passante.

L'inéquité entre les flots est également accentuée pour TFRC. En effet, comme précédemment, la quantité de bande passante allouée aux connexions A et B est quasiment deux fois supérieure à celle allouée aux autres connexions. On peut noter que, pour TCP Vegas, les connexions ne traversant qu'une congestion disposent de plus de bande passante que les autres, mais cette différence n'est pas trop marquée. Enfin, les résultats obtenus par Primo restent aussi bons que dans la série précédente ; la stabilité du débit de réception et l'équité sont quasiment parfaites.

Quatrième scénario de démarrage. Dans la quatrième série de simulations (*cf.* figure 6.17), la connexion traversant trois congestions (connexion E) commence à émettre seule sur le réseau.

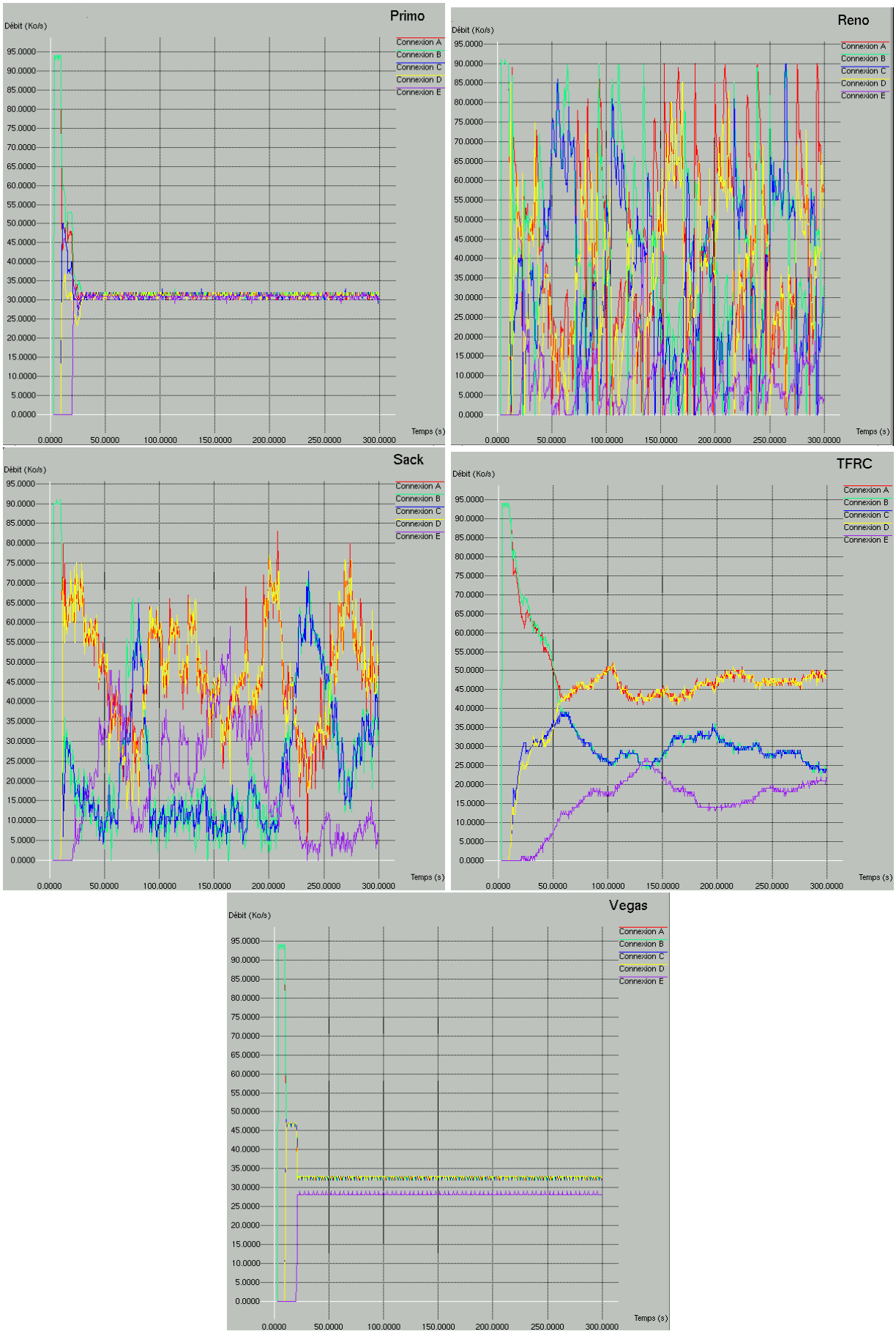

FIG. 6.15 – *Débit de réception dans un environnement multi-congestioné : démarrage des connexions traversant une congestion, puis de celles en traversant deux et enfin de celle en traversant trois.*

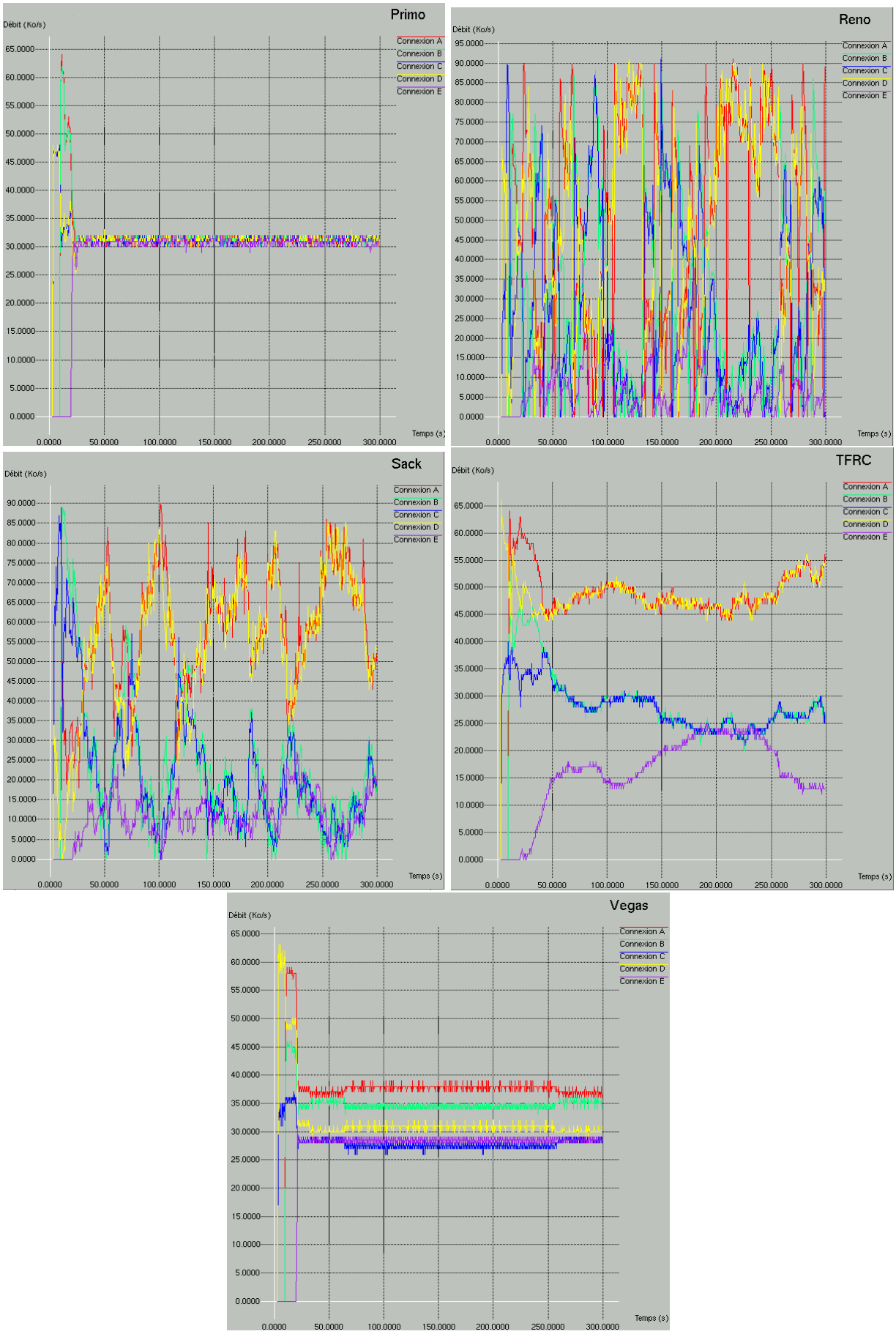

FIG. 6.16 – *Débit de réception dans un environnement multi-congestioné : démarrage des connexions traversant deux congestions, puis de celles en traversant une et enfin de celle en traversant trois.*

Puis, cinq secondes plus tard, le groupe de connexions ne traversant qu'une congestion (connexions A et B) commence à émettre. Enfin, après dix secondes de simulation, le groupe de connexions traversant deux congestions (connexions C et D) commence à émettre.

Dans cette série de simulations, l'inéquité entre les flots utilisant TCP Sack est très marquée. Dès le début, et quasiment tout au long de la simulation, la bande passante est monopolisée par les connexions A et D. Il en va de même pour TCP Reno, avec lequel la domination des connexions A et D est très rapide et dure toute la simulation.

Les résultats obtenus par TFRC montrent une inéquité assez forte entre les connexions A, D et E. En effet, la connexion E a une part de bande passante allouée jusqu'à trois fois inférieure à celle des connexions A et D.

Les résultats obtenus par TCP Vegas et Primo sont quasiment les mêmes que ceux obtenus lorsque toutes les sources commencent à émettre simultanément. Pour TCP Vegas, la stabilité du débit d'émission est parfaite mais le partage de la bande passante n'est pas totalement équitable. Pour Primo, le partage de la bande passante est relativement équitable entre les connexions A, B, C et D jusqu'à t = 200 s, puis la connexion A consomme légèrement plus de bande passante que les autres connexions. Notons que, tout au long de la simulation, la part de bande passante allouée à la connexion E est deux à trois fois inférieure à celle allouée aux autres connexions.

Cinquième scénario de démarrage. Dans cette série de simulations (*cf.* figure 6.18), la connexion traversant trois congestions (connexion E) commence à émettre seule sur le réseau. Puis, cinq secondes plus tard, le groupe de connexions traversant deux congestions (connexions C et D) commence à émettre. Enfin, après dix secondes de simulation, le groupe de connexions ne traversant qu'une congestion (connexions A et B) commence à émettre.

Dans cette série de simulations, les résultats obtenus par TCP Sack et TFRC sont assez mauvais en terme d'équité. En effet, au bout de 25 s pour TCP Sack (et 50 s pour TFRC), les connexions A et D commencent à monopoliser la bande passante disponible. Pour TCP Sack, on peut observer qu'à certains moments (par exemple entre t = 200 s et t = 250 s) la bande passante allouée à la connexion E est plus de dix fois inférieure à celles des connexions A et D. De même, pour TFRC, au bout de 250 s de simulation, la bande passante octroyée aux connexions B, C et E est plus de sept fois inférieure à celle des connexions A et D.

Les résultats obtenus par TCP Reno sont similaires à ceux du quatrième scénario : la bande passante est monopolisée à tour de rôle par les connexions A, B, C, D et la connexion E n'en obtient qu'une part infime.

Les résultats obtenus par TCP Vegas sont très proches des ceux obtenus dans le quatrième scénario. Pour Primo, la bande passante est équitablement partagée entre les connexions A, B, C et D. La part de bande passante obtenue par la connexion E est deux fois plus faible que celle obtenue par les autres connexions.

6.7.2 Analyse

Lissage et constance du débit. Quel que soit le scénario de démarrage utilisé, la constance et le lissage du débit de réception, pour un protocole donné, sont les mêmes. En effet, on peut regrouper les protocoles en trois classes :

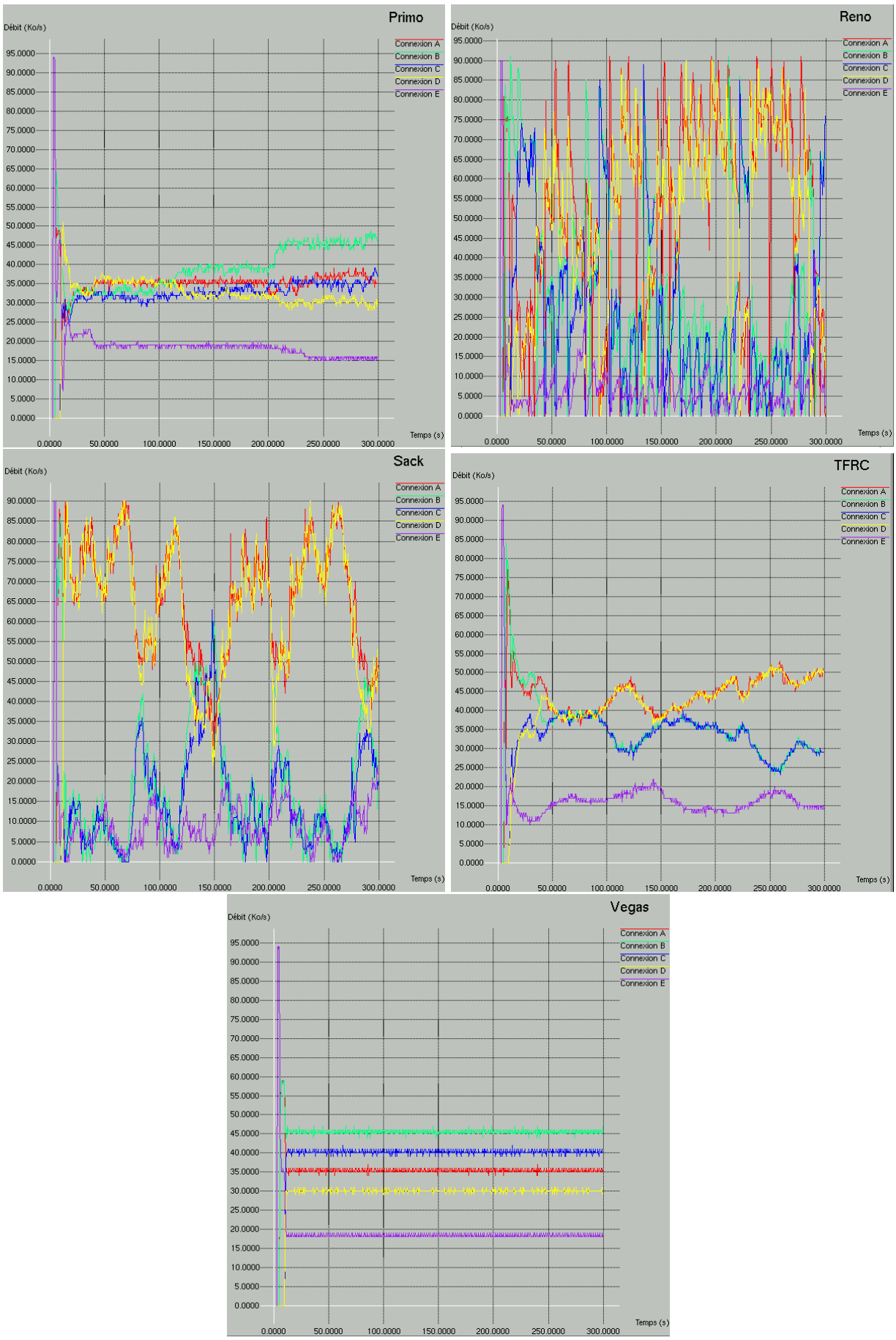

FIG. 6.17 – *Débit de réception dans un environnement multi-congestioné : démarrage de le connexion traversant trois congestions, puis de celles en traversant une, et enfin de celles en traversant deux.*

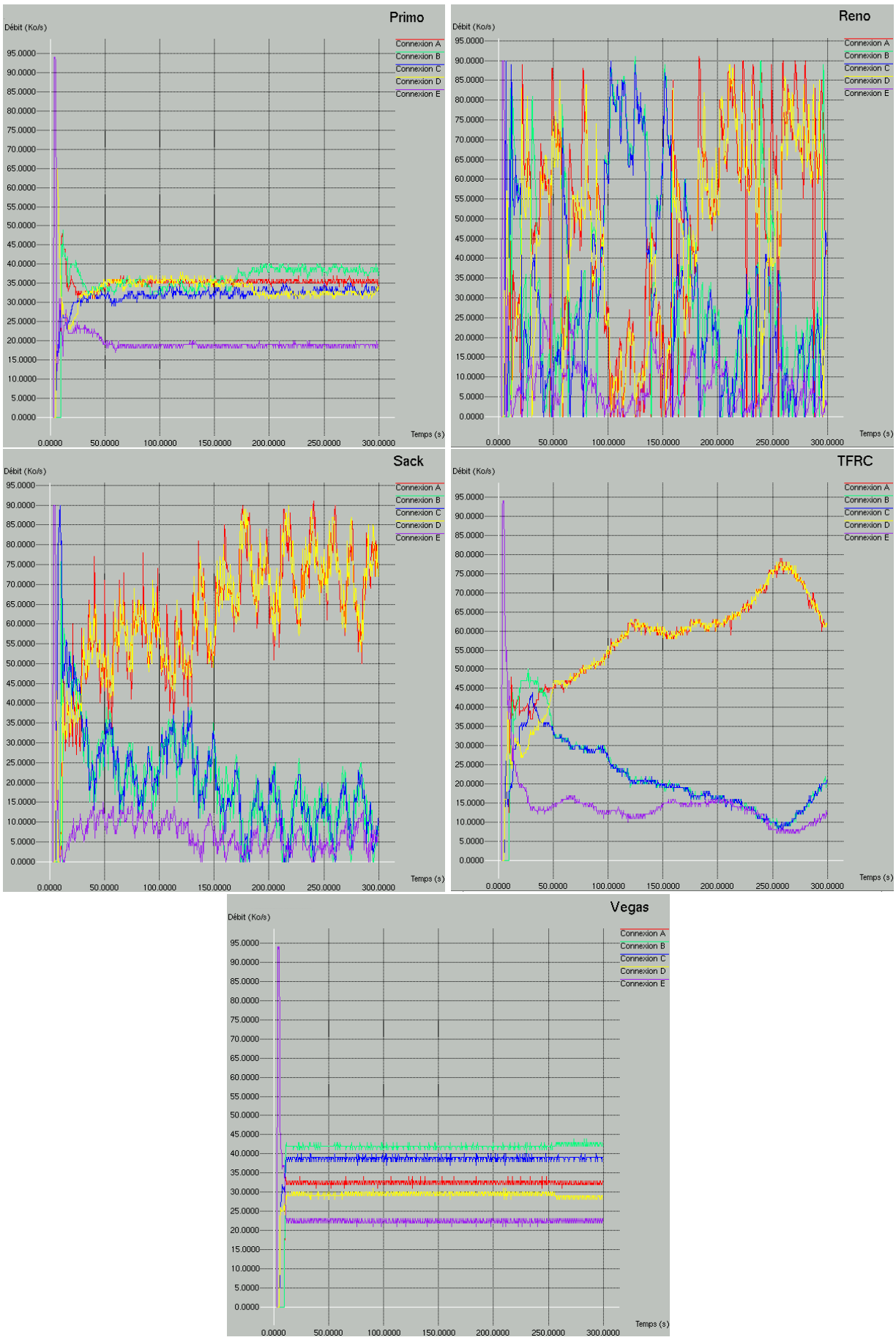

FIG. 6.18 – *Débit de réception dans un environnement multi-congestioné : démarrage de la connexion traversant trois congestions, puis de celles en traversant deux, et enfin de celles en traversant une.*

– les protocoles ne parvenant ni à lisser, ni à stabiliser le débit de réception : TCP Reno et Sack ;
– les protocoles ne parvenant pas à stabiliser le débit de réception, mais réussissant à le lisser : TFRC ;
– les protocoles parvenant à stabiliser et à lisser leur débit de réception : TCP Vegas et Primo.

Les variations de débit de TCP Reno et Sack pouvant aller jusqu'à 100% (cas où le débit de réception passe de son maximum à 0), ces protocoles ne paraissent pas adaptés au transport de flots de données temps réel.

Bien que TFRC ne parvient pas à stabiliser complètement le débit de réception, il réussit à bien le lisser, grâce à son mécanisme de régulation de débit basé sur un délai inter-paquets. Les variations de débit de ce protocole étant assez douces, il pourrait être utilisé dans le cadre d'une transmission multimédia. En effet, lorsque le débit varie progressivement, l'application fournissant les données à transmettre a la possibilité de s'y adapter (*e.g.*, compression plus importante dans le cas d'une transmission vidéo).

Enfin, les protocoles TCP Vegas et Primo offrent une stabilité et un lissage du débit quasiment parfaits. Notons que, pour la stabilisation du débit, TCP Vegas est légèrement plus performant que Primo. En effet, nous avons observé qu'après une phase d'initialisation, TCP Vegas conserve le même débit de réception tout au long de la simulation, quel que soit le scénario de démarrage utilisé.

Ces premières constatations laissent apparaître que les protocoles préventifs sont les plus efficaces pour stabiliser et lisser le débit de réception. Notons cependant que, malgré son comportement correctif, TFRC offre un très bon lissage du débit. Enfin, l'observation des courbes de débit de réception de TCP Reno et Sack nous permet de conclure que ces protocoles paraissent inutilisables pour la transmission d'un flot multimédia.

Équité. Idéalement, quel que soit leur groupe d'appartenance, toutes les connexions devraient obtenir la même quantité de bande passante (*i.e.*, avoir le même débit de réception). Cependant, nous avons observé que, généralement, la connexion traversant le plus de congestions est défavorisée par rapport aux autres.

Comme pour le lissage et la constance du débit de réception, on peut distinguer 3 groupes de protocoles :

– les protocoles avec lesquels on constate une très forte inéquité, quel que soit le scénario de démarrage des connexions : TCP Reno et Sack ;
– les protocoles avec lesquels l'inéquité entre les connexions est variable, sans jamais être très bonne, en fonction du scénario de démarrage : TFRC ;
– les protocoles pour lesquels, suivant le scénario de démarrage l'équité obtenue varie de parfaite à raisonnable : TCP Vegas et Primo.

Comme nous l'avons vu dans l'ensemble de simulations 6.5, TCP Reno et Sack ne parviennent pas à maintenir l'équité lorsque les délais de propagation des connexions en concurrence sont hétérogènes. Cette constatation se vérifie dans cet ensemble de simulations. On peut supposer que, dans le cas d'un environnement multi-congestionné, l'inéquité de ces protocoles est accentuée par les pertes de segments dues aux multiples congestions. En effet, plus le nombre

de congestions traversées par les paquets est important, plus ces derniers ont de chance d'être détruits dans les routeurs. Or, les connexions dont les paquets traversent le plus de congestions étant celles dont le RTT est le plus long (*cf.* figure 5.4 page 109), elles ont peu de chance d'obtenir une part de bande passante équitable.

Le protocole TFRC parvient dans certains cas à réduire l'inéquité entre les connexions. On peut supposer que ces meilleures performances par rapport à TCP Reno et Sack sont en partie dues à sa capacité à maintenir l'équité lorsque les connexions concurrentes possèdent des temps de propagation différents (*cf.* section 6.5). En revanche, avec ce protocole, quel que soit le scénario de démarrage de sources, les connexions ne parviennent jamais à obtenir exactement la même part de bande passante.

Dans plusieurs scénarios de démarrage, TCP Vegas et Primo parviennent à partager de manière totalement équitable la bande passante entre les différentes connexions. Cette équité est obtenue grâce au comportement préventif de ces protocoles. En effet, un tel comportement permet de réduire l'agressivité des connexions traversant le moins de routeurs, et donc ré-équilibre le partage de la bande passante. Dans ces simulations, nous avons pu remarquer que Primo était généralement plus équitable que TCP Vegas. Cette situation est principalement liée à deux caractéristiques de Primo :

- Primo a un « degré de prévention » plus élevé que TCP Vegas. Comme nous l'avons vu dans tous les ensembles de simulations précédents, Primo a un taux d'espace libre dans la file d'attente qui est supérieur à TCP Vegas. Ceci traduit le fait que Primo commence à réduire son débit d'émission pour un niveau de congestion plus faible que celui utilisé par TCP Vegas. Il semble donc qu'en autorisant un niveau de congestion plus faible (à partir duquel le débit sera réduit), Primo améliore l'équité entre les flots.
- Primo utilise un délai inter-paquets pour réguler son débit d'émission. En effet, comme nous l'avons vu dans l'ensemble de simulations portant sur les délais de propagation hétérogènes (*cf.* section 6.5), les protocoles utilisant un délai inter-paquets pour réguler leur débit parviennent plus facilement à maintenir l'équité.

Dans ces simulations, nous avons observé que les connexions A et D disposaient fréquemment d'une part de bande passante supérieure à celle des autres connexions. L'obtention d'une part importante de bande passante par la connexion D est difficilement explicable. En effet, on pourrait s'attendre à ce que ce soient les connexions dont les paquets ne traversent qu'une congestion (connexions A et B) qui possèdent la plus grande part bande passante. En analysant ces résultats, nous ne voyons pas d'explication liée aux caractéristiques des protocoles. Par contre, nous pouvons en esquisser une liée à la topologie utilisée. En effet, le nombre de flots entrant sur les routeurs n'est pas toujours le même. Les routeurs auxquels sont connectés les sources A, B, C et E n'ont que 3 flots en entrée (*cf.* figure 5.4 page 109) alors que le routeur auquel est connecté la source D en a 4. Cette différence pourrait peut-être, dans certains cas, favoriser la connexion D. Or, si l'on admet que cette situation favorise la connexion D, il est logique que la connexion B (qui est directement en concurrence avec la D) ait un débit d'émission inférieur à ce qu'il devrait être. Il est cependant difficile de vérifier cette hypothèse.

Dans la méthode d'évaluation présentée au chapitre 5, nous avons proposé d'utiliser différents scénarios de démarrage des sources afin d'observer de quelle manière une connexion parvient à s'imposer lorsque d'autres connexions sont déjà établies. Les résultats de ces simulations ne nous ont permis de tirer aucune conclusion quant à l'influence de l'ordre de démarrage des sources. En effet, il semble que l'instant auquel démarrent les connexions soit plus impor-

FIG. 6.19 – *Dans la figure de gauche, les groupes de connexions E, C/D et A/B démarrent à 5 s d'intervalle, dans la figure de droite ces groupes démarrent dans le même ordre avec un intervalle de 5,1 s. Ces figures ont été obtenues en utilisant le protocole TFRC.*

tant que leur ordre de démarrage. Pour vérifier cette hypothèse nous avons effectué des simulations dans lesquelles l'ordre de démarrage des sources était toujours le même mais en utilisant différentes valeurs pour l'intervalle de temps séparant les démarrages. Les résultats que nous avons obtenus ont confirmé notre supposition. Dans la figure 6.19, en conservant le même ordre de démarrage mais en utilisant des intervalles d'attente différents, nous obtenons pour TFRC des résultats très différents. On peut donc en conclure que l'ordre d'arrivée des connexions sur le réseau à moins d'influence que l'instant auquel elles commencent à émettre. Les routeurs peuvent être saturés ou non ; suivant l'instant auquel le premier paquet d'une connexion y arrivera, il sera détruit ou non. Or, si ce paquet est détruit, le débit d'émission de la connexion sera réduit. Inversement, si ce paquet parvient au récepteur, le débit de connexion sera augmenté. Ces deux cas engendrent deux situations totalement différentes qui auront une forte influence sur les résultats de la simulation. Notons que cette observation se vérifie principalement lorsque les protocoles testés sont de type correctif, car ils ont tendance à périodiquement saturer la file d'attente. En effet, les protocoles préventifs ne saturant pas les files d'attente, il y a peu de chance que des paquets soient perdus quel que soit l'instant auquel ils arrivent au routeur.

Comme nous venons de le voir, ces simulations sont difficiles à interpréter et leurs résultats sont très variables (un léger décalage de l'instant de démarrage d'une source a un impact important sur les résultats de la simulation). Cependant, elles nous autorisent tout de même à distinguer les protocoles ayant tendance à être inéquitables. En effet, nous avons pu constater que, pour les protocoles correctifs, il semble difficile de maintenir l'équité dans un environnement multi-congestionné. Notons que ces constatations ne nous permettent pas de conclure à la supériorité des protocoles préventifs, mais elles nous autorisent à supposer qu'ils ont plus de facilités à partager équitablement la bande passante dans un environnement multi-congestionné.

6.7.3 Conclusion

Les simulations effectuées dans un environnement multi-congestionné nous ont permis de distinguer, en terme de lissage et de constance du débit de réception, trois classes de protocoles :

- les protocoles ne parvenant ni à lisser, ni a stabiliser leur débit : TCP Reno et Sack ;
- les protocoles parvenant uniquement à lisser leur débit : TFRC ;
- les protocoles parvenant à lisser et à stabiliser leur débit : TCP Vegas et Primo.

Ces résultats confirment ceux obtenus dans les ensembles de simulations visant à évaluer l'influence des différents paramètres du réseau (*cf.* sections 6.1 à 6.6).

Nous avons également distingué trois classes de protocoles pour ce qui est de l'équité :

- les protocoles pour lesquels l'inéquité est très forte, quel que soit le scénario de démarrage des connexions : TCP Reno et Sack ;
- les protocoles pour lesquels l'inéquité varie, sans jamais être très bonne, en fonction du scénario de démarrage : TFRC ;
- les protocoles pour lesquels l'équité varie de parfaite à raisonnable, suivant le scénario de démarrage : TCP Vegas et Primo.

Ces simulations nous laissent supposer que les protocoles préventifs ont plus de faciliter à maintenir l'équité dans un environnement multi-congestionné. Cependant, la complexité des résultats nous permet uniquement d'émettre des hypothèses qui seront difficiles à vérifier.

6.8 Conclusion

Après avoir mis au point dans le chapitre 4 un nouveau protocole de transport incluant un mécanisme de contrôle de congestion, nous avons voulu évaluer ses performances. Cette évaluation s'est faite grâce à la méthode présentée au chapitre 5. L'étude des résultats nous a permis de dégager certaines conclusions sur le comportement des protocoles testés ainsi que sur leur capacité à être utilisés pour la transmission d'un flot temps réel.

Bilan. Les protocoles TCP Reno et Vegas sont très sensibles aux pertes d'acquittements. En effet, dans tous les ensembles de simulations, lorsque des pertes d'acquittements surviennent, leurs performances se dégradent fortement. Les autres protocoles testés ne semblent pas être sensibles aux pertes d'acquittements. En effet, l'option implémentée par TCP Sack lui permet de s'affranchir des réductions de débit inutiles engendrées par les pertes d'acquittements et donc de maintenir ses performances. Le contrôle de congestion de TFRC et de Primo n'étant pas basé sur la détection de pertes *via* la réception d'acquittements, leurs performances ne sont pas diminuées lorsque des congestions apparaissent dans le sens retour.

Résultat 10 *Les protocoles implémentant leur mécanisme de détection de congestion du côté du récepteur (Primo et TFRC) sont peu sensibles aux pertes d'acquittements.*

Les simulations présentées dans ce chapitre ont mis en évidence que TCP Vegas ne parvient pas à maintenir l'équité entre les flots lorsque la taille maximale de la file d'attente est très petite (de l'ordre de 5 ko, *cf.* section 6.2). Notons qu'il est peu probable de rencontrer une telle situation dans un environnement réel. Le même constat a été fait pour Primo lorsque la file d'attente implémente une politique RED. Lorsque les files d'attente sont de taille raisonnable

(supérieure à 10 ko), TCP Vegas et Primo sont les seules protocoles qui parviennent à ne pas les saturer. Le caractère préventif du contrôle de congestion implémenté par ces protocoles leur permet de laisser un espace libre important dans les files d'attente.

Les protocoles ayant un comportement correctif (TCP Reno, Sack et TFRC) ont tendance à saturer les files d'attente. Le protocole TFRC est celui qui laisse le moins d'espace libre dans les files d'attente.

Résultat 11 *Lorsque les files d'attente ont une taille maximale raisonnable (i.e., au moins supérieure à 10 ko), les protocoles préventifs (Primo et TCP Vegas) ne les remplissent jamais complètement.*

Comme nous l'avons vu dans le chapitre 5, la constance des débits d'émission et de réception sont des critères importants d'un point de vue utilisateur (*i.e.*, fluidité de la réception d'un flux temps réel) comme d'un point de vue opérateur (*i.e.*, stabilité du réseau). Dans ce domaine, les protocoles TCP Reno et Sack obtiennent des résultats assez mauvais. En effet, aucun de ces deux protocoles n'est équipé d'un mécanisme anti-rafales. Ils ont donc tendance à émettre la totalité de leur fenêtre d'émission en un temps très restreint, puis à attendre la réception des acquittements correspondants. Ce comportement conduit à une alternance de phases d'émission à un débit très élevé et de phases de « silence » (phase sans émission).

Le protocole TCP Vegas, qui est équipé d'un système anti-rafales, parvient à maintenir un débit d'émission et de réception constant tant que les temps de propagations ne sont pas trop importants, et que la taille de sa fenêtre de congestion n'est pas trop faible. En revanche, lorsque le RTT augmente (*cf.* section 6.4) ou que la taille de sa fenêtre de congestion diminue (*cf.* section 6.1), le mécanisme anti-rafales devient inefficace, ce qui rend instable les débits d'émission et de réception. De même, les pertes d'acquittements semblent perturber le mécanisme anti-rafales de ce protocole.

Seuls TFRC et Primo, dont le contrôle de congestion n'est pas basé sur une fenêtre, parviennent à conserver des débits d'émission et de réception constants quelle que soit la situation.

Résultat 12 *L'utilisation d'un délai inter-paquets pour réguler le débit d'émission favorise sa constance.*

Ces simulations ont également montré que les protocoles utilisant une fenêtre pour gérer le débit d'émission ne sont pas équitables lorsque des flots ayant des temps de propagation hétérogènes se partagent un même lien. En effet, nous avons constaté qu'avec TCP Reno, Sack ou Vegas, la connexion ayant le RTT le plus court a tendance à utiliser une part de bande passante plus importante que celle utilisée par l'autre connexion (*cf.* section 6.5). Ce phénomène est dû à deux caractéristiques liées à la gestion des fenêtres de congestion et d'émission :

- la gestion de la taille de la fenêtre de congestion dans TCP Reno et Sack est telle que la connexion ayant le RTT le plus court verra sa fenêtre croître plus rapidement que celle de l'autre connexion ;
- la gestion du glissement de la fenêtre d'émission (quel que soit le protocole) est telle que la connexion ayant le RTT le plus court verra sa fenêtre glisser plus fréquemment que l'autre connexion.

Résultat 13 *L'utilisation d'un délai inter-paquets pour réguler le débit d'émission favorise l'équité.*

Nous avons également constaté que, lorsque le nombre de sources augmente, les performances de certains protocoles se dégradent (*cf.* section 6.6). Pour TCP Vegas, le taux d'espace libre dans la file d'attente diminue à mesure que le nombre de sources augmente. Pour TFRC, le taux de bonnes transmissions diminue lorsque le nombre de source augmente. Ces indications laissent supposer que ces deux protocoles risquent de changer de comportement (*i.e.*, augmentation de la taille de la file d'attente, du taux de pertes, *etc.*) lors de leur implantation sur un réseau composé d'un grand nombre de machines. En revanche, les performance de Primo ne sont pas affectées par l'augmentation du nombre de sources, ce qui est de bonne augure pour un passage à l'échelle.

Résultat 14 *Le protocole Primo supporte mieux que les autres protocoles l'augmentation du nombre de sources.*

Pour finir, un dernier ensemble de simulations nous a permis d'évaluer l'équité des protocoles dans un environnement multi-congestionné. La complexité d'interprétation de ces simulations ne nous a pas permis de conclure à la supériorité d'un protocole par rapport aux autres. En revanche, ces simulation nous indiquent que les protocoles de type préventif paraissent avoir plus de facilité à maintenir l'équité dans un environnement multi-congestionné.

Résultat 15 *Les protocoles préventifs (Primo et TCP Vegas) ont plus de facilité à maintenir l'équité dans un environnement multi-congestionné.*

Le transport de flots temps réel requiert un délai d'acheminement et une gigue de ce délai les plus faibles possibles. Dans nos simulations, seule la variation de la taille de la file d'attente peut avoir une influence sur le délai d'acheminement et sa gigue. Lorsqu'il n'y a que deux sources, en évitant de saturer la file d'attente, TCP Vegas et Primo minimisent le délai d'acheminement. Nous avons également noté que dans cette situation, ces deux protocoles parviennent à maintenir une gigue assez faible (*cf.* remarque 2, page 123). En revanche, nous avons observé que, lorsque le nombre de sources augmente, Primo est le seul protocole à conserver un espace important dans la file d'attente (*cf.* section 6.6). Enfin, dans un environnement multi-congestionné, Primo et TCP Vegas, grâce à la constance de leur débit, semblent être les plus aptes à stabiliser la taille des différentes files d'attente et donc la gigue. L'ensemble de ces observations, nous permet, pour Primo, d'extrapoler raisonnablement les résultats concernant la constance et le taux de remplissage des files d'attente (*i.e.*, le délai d'acheminement et sa gigue), pour des situations à n sources et m congestions. Ainsi, Primo est le seul protocole à minimiser le délai d'acheminement ainsi que sa gigue, dans la plupart des situations.

Résultat 16 *Le protocole Primo minimise le délai d'acheminement ainsi que sa gigue dans beaucoup de situations.*

Perspective d'utilisation. Les simulations présentées dans ce chapitre nous ont permis de tirer quelques conclusions à propos des protocoles testés :

– **TCP Reno**. Le principal défaut de ce protocole pour une transmission multimédia est l'inconstance de son débit d'émission et donc de réception (par exemple : images saccadées dans une transmission vidéo). De plus, son comportement correctif le pousse à saturer les files d'attente, ce qui engendre une augmentation du RTT qui peut nuire à la qualité d'une transmission temps réel (par exemple : temps de latence important durant une

visio-conférence). Ce protocole devient inéquitable lorsque les connexions empruntant un même lien ont des RTT hétérogènes. Enfin, sa sensibilité aux pertes d'acquittements, qui semblent inévitables dans un environnement réel, diminue ses performances.

- **TCP Sack**. Les défauts de ce protocole sont les mêmes que ceux de TCP Reno hormis pour la vulnérabilité aux pertes d'acquittements. En effet, l'amélioration qu'il apporte à la gestion des acquittements (*i.e.*, redondance d'information) lui permet de ne quasiment pas être influencé par leur pertes ;
- **TCP Vegas**. Le principal avantage de ce protocole pour une transmission de données temps réel est son comportement préventif. Cela lui permet de maintenir un espace libre important dans la file d'attente et ainsi de minimiser le délai d'acheminement des données. De plus, de manière générale, les débits d'émission et de réception de ce protocole sont constants, ce qui est une qualité importante pour le transport de données multimédia. En revanche, ce protocole est très sensible aux pertes d'acquittements qui ont tendance à fortement dégrader l'ensemble de ses performances (constance de débit, taux d'occupation du lien congestionné, *etc.*) ;
- **TFRC**. Dans ce protocole, le mécanisme de détection de congestions est situé du côté du récepteur, ce qui lui permet de s'affranchir de l'influence des pertes d'acquittements. De plus, l'utilisation d'un délai inter-paquets pour réguler son débit d'émission favorise son lissage. Son comportement correctif le pousse à saturer les files d'attente, ce qui a tendance à augmenter le délai d'acheminement des données (qui nuit aux transmissions temps réel) ;
- **Primo**. Comme TCP Vegas, qui a un comportement préventif, ce protocole est bien adapté aux transmissions de données temps réels. De plus, il utilise un délai inter-paquets, ce qui garantit la constance de son débit d'émission. Enfin, comme TFRC, le mécanisme mesurant l'état de congestion est basé du côté récepteur, ce qui lui permet d'être insensible aux pertes de données.

À l'issue de ces comparaisons, seuls les protocoles TFRC et Primo semblent être utilisables pour le transport de flux temps réel. En effet, les protocoles TCP Reno et Sack sont inutilisables dans le cadre d'une transmission vidéo, car ils sont incapables de stabiliser les débits d'émission et de réception. Quant à TCP Vegas, sa sensibilité aux pertes d'acquittements le rend inefficace dans un grand nombre de situations.

Dans ces simulations, le protocole Primo est légèrement plus performant que TFRC. Cependant, on peut noter que dans certains cas, Primo devient largement plus performant que TFRC, notamment lorsque le nombre de sources augmente. Enfin, on remarquera que Primo est le seul protocole à préserver un espace libre important dans la file d'attente, quelle que soit la situation dans laquelle il se trouve. Cette caractéristique rend Primo plus adapté que TFRC pour le transport de données supportant mal les délais d'acheminement importants.

Chapitre 7

Conclusion

7.1 Bilan des travaux

7.1.1 Positionnement du problème

Depuis 1988, suite à une grave série de congestions, les recherches menées sur les mécanismes de contrôle de congestion n'ont jamais cessées. De nos jours, avec l'accroissement du nombre de lignes à haut débit, les applications à flux de données temps réel (vidéos, audio, *etc.*) se sont rapidement répandues sur Internet. Ce type d'applications, supportant mal les fortes variations de débit imposées par TCP, ont donné lieu à de nombreuses études visant à développer un contrôle de congestion qui leur soit adapté.

Avant de commencer à développer notre propre système de contrôle de congestion, nous étudié les solutions déjà proposées. Cette étude bibliographique nous a permis de dégager les pistes d'amélioration possibles.

Quelle piste d'améliorations ?

Dans ce manuscrit de thèse, nous avons commencé au chapitre 2 par étudier le fonctionnement de TCP Reno. Puis une synthèse des nombreuses évolutions et alternatives proposées à ce protocole a été présentée. Cette étude des différents mécanismes de contrôle de congestion actuellement implémentés dans des protocoles de transport *unicast*, nous a permis de dégager quelques pistes pour le développement de nouveaux protocoles. Comme nous l'avons vu, les différents types d'améliorations apportées peuvent être classés en trois grandes familles :

- **Amélioration de la gestion des acquittements.** Cette piste a déjà été largement étudiée, et semble désormais peu prometteuse. En effet, les précisions apportées par TCP Sack, New Reno ou Fack ne paraissent plus significativement améliorables.
- **Amélioration de la gestion du débit d'émission.** Cet axe est celui qui laisse le plus d'opportunités d'améliorations. Le débit d'émission peut être géré par une fenêtre d'émission ou grâce à un délai inter-paquets. L'utilisation d'une fenêtre de congestion a l'avantage de limiter automatiquement la quantité de données en transit sur le réseau. En revanche, la gestion du débit d'émission sans fenêtre d'émission permet généralement d'accroître le lissage du débit d'émission.
- **Informations provenant directement du réseau.** L'indication explicite de congestion (*i.e.*, le marquage des paquets par les routeurs) est une piste qui semble prometteuse (*cf.*

travaux menés sur ECN [43, 97, 8, 34]). Cependant, les techniques liées aux protocoles implémentés sur les équipements intermédiaires (*e.g.*, les routeurs) posent des problèmes au niveau du déploiement. En effet, le déploiement de ces techniques ne concerne pas uniquement les deux extrémités d'une connexion, mais tous les équipements qui participent à la communication. De plus, l'utilisation de ces techniques peut engendrer des problèmes de compatibilité avec les politiques de sécurité réseau. Il est, par exemple, fréquent que des pare-feux rejettent les paquets contenant un marquage ECN en suspectant une attaque.

De ces trois pistes, nous avons choisi d'exploiter celle de l'amélioration de la gestion du débit d'émission. Une fois la piste d'amélioration, choisie, il a fallut déterminer le type de comportement que nous souhaitions développer : préventif ou correctif.

Comportement préventif ou correctif ?

L'augmentation du nombre d'utilisateurs disposant d'une ligne à haut débit a entraîné une forte augmentation du nombre d'applications à flux de données temps réel (*e.g.*, vidéos, audio, *etc.*). Ces applications utilisent UDP car elles supportent assez mal les fortes variations de débit imposées par TCP. Or UDP n'implémente aucun mécanisme de contrôle de congestion, ce qui, à terme, risque d'engendrer de graves congestions. Pour résoudre ce problème, de nouveaux protocoles proposent d'effectuer un contrôle de congestion sans pour autant imposer de fortes variations de débit. De plus, ces protocoles ont pour objectif de partager équitablement la bande passante disponible avec TCP : ils sont *TCP-friendly*.

Le partage équitable de la bande passante avec TCP est un facteur limitant pour améliorer le contrôle de congestion. En effet, la cohabitation avec TCP Reno et New Reno (versions représentant 90% du trafic Internet [90, 32]) impose aux nouveaux protocoles d'avoir un comportement correctif. Il semble pourtant évident qu'un protocole correctif, qui crée des congestions, est moins performant qu'un protocole préventif, qui les évite. Il a d'ailleurs été montré à plusieurs reprises que les protocoles préventifs, tel que TCP Vegas, étaient plus efficaces que les protocoles correctifs [2, 75]. Bien que le contrôle de congestion préventif pose des problèmes de déploiement (impossibilité de les mettre en concurrence avec des protocoles correctifs), nous avons tout de même choisi de développer un mécanisme de ce type.

Méthode utilisée

La modélisation continue des phénomènes de congestions sur les réseaux à commutation de paquets apporte une grande simplification du problème. Les problèmes des réseaux à commutation de paquets (comme le contrôle de congestions) peuvent être réduits à des problèmes de contrôle des systèmes à entrées sorties, qui sont largement traités en automatique [26]. En effet, une connexion peut être modélisée par un système à une entrée et deux sorties avec un bouclage implicite à délai incertain. Les sorties du système (le débit de réception et le délai d'acheminement) sont transmises au correcteur (la source) afin qu'il puisse ajuster l'entrée du système (le débit d'émission). Le développement d'un nouveau mécanisme de contrôle de congestion basé sur un correcteur automatique est une piste qui jusqu'à présent est restée peu explorée. La mise au point d'un tel correcteur est plus simple à mettre en œuvre dans un environnement continu où tous les paramètres sont maîtrisés.

Le développement de protocoles préventifs, basé sur des principes issus de la théorie du contrôle des systèmes, semble être l'une des dernières pistes prometteuses peu explorées. C'est

pourquoi nous avons choisi de l'exploiter.

7.1.2 Validité de l'approximation continue

Le développement d'un correcteur dans un environnement continu implique sa discrétisation lors de son implantation dans une protocole de transport. Nous avons donc cherché à déterminer quel était le domaine de validitié de la modélisation en continue.

Pour ce faire, nous avons étudié la cohérence entre les modèles discret et continu d'un réseau à commutation de paquets (*cf.* chapitre 3). Nous avons comparé le comportement d'un protocole de transport discret et celui de sa modélisation continue. La comparaison du comportement des deux modèles s'est faite grâce à des simulations. Ces simulations ont permis de déterminer les paramètres du réseau ou du protocole qui avaient une influence sur la cohérence des modèles discret et continu. Elles ont également permis de dégager certaines limites pour la validité de l'approximation continue :

- **Rapport de bande passante.** Lorsque le rapport de bande passante entrante/sortante est très grand (*i.e.*, il y a une grosse différence entre les débits entrant et sortant du nœud de congestion), et que le protocole de transport impose de fortes variations de débit, l'approximation continue ne reflète pas le comportement discret. En revanche, lorsque le protocole de transport régule le débit de manière lissée, quel que soit le rapport de bande passante, l'approximation continue est pertinente.
- **Délais de propagation.** Les délais de propagation des liens ont très peu d'influence sur la cohérence entre les modèles continu et discret. Notons cependant que pour des valeurs très importantes des temps de propagation (lorsque le délai de propagation aller-retour dépasse 1,5 s) l'approximation continue n'est plus cohérente avec le modèle discret.
- **Variations de débit.** L'importance des variations de débit est un paramètre très influent. En effet, lorsque les variations de débits sont douces, les modèles continu et discret sont cohérents. Par contre, lorsque le protocole impose de fortes variations, les modèles continu et discret ont des comportements assez différents.
- **Tolérance et taille des paquets.** La réactivité du protocole ainsi que la taille des paquets (tant qu'elle reste réaliste, c'est-à-dire, inférieure à 10 ko) ne semblent pas agir sur la cohérence entre les modèles continu et discret.

Les résultats montrent que, dans une grande variété de situations, la modélisation continue d'un réseau à commutation de paquets est pertinente. Ces résultats ont été obtenus avec un protocole simple de type AIMD, mais nous pensons qu'ils peuvent être extrapolés à d'autres protocoles de transport de type AIMD. Dans notre étude, on retiendra donc que, pour des valeurs raisonnables des délais de propagation et de la taille des paquets, si le protocole modélisé cherche à lisser le débit d'émission (*i.e.*, pas de brusque variations), le protocole (discret) et son approximation continue auront le même comportement.

La validité de l'approximation continue d'un réseau ouvre de nouvelles perspectives pour le contrôle de congestion. En effet, des résultats prometteurs [83, 118, 22] obtenus dans le domaine du continu pourraient être transposés en discret. De plus, la validité de l'approximation continue autorisera des simulations de plus en plus complexes grâce à de nouveaux modèles. Ces nouvelles modélisations continues devraient permettre d'effectuer des simulations à large échelle [67], qui sont actuellement irréalisables avec des simulateurs à événements discrets.

7.1.3 Développement de Primo

Nouveaux mécanisme de contrôle de congestion

Sous l'hypothèse que les résultats obtenu en continu étaient transposables en discret, nous avons développé un correcteur automatique dédié au contrôle de congestion.

Les objectifs fixés pour le développement de ce nouveau mécanisme de congestion étaient les suivants :

- **Rapidité.** La bande passante disponible doit être utilisée à son maximum dès le début de la connexion.
- **Prévention.** Le protocole doit être capable de réduire le débit de la source avant que les congestions n'apparaissent. Cette caractéristique permettra de réduire la quantité de données perdues lors d'une connexion.
- **Constance du débit.** Le débit doit être le plus constant possible, tout en respectant les contraintes liées au contrôle de congestion et à l'équité.
- **Réaction aux perturbations.** Le protocole doit être capable d'adapter proportionnellement son débit à la charge du réseau.

Après plusieurs ajustements, le correcteur obtenu est de type Proportionnel Intégral (PI) modifié. L'utilisation de ces deux composantes couplée à une phase d'initialisation efficace, permet au correcteur de satisfaire les spécifications précédentes.

Discrétisation

La discrétisation du correcteur développé en continu s'est faite sous *ns*. Ce simulateur fait office de référence dans le domaine du contrôle de congestion (*i.e.*, la plupart des études l'utilisent).

Cette étape du développement a donné lieu à l'ajout de quelques fonctionnalités ainsi qu'à des ajustements (*cf.* section 4.6). En effet, la discrétisation du correcteur nous a amené à ajouter des mécanismes détectant les incohérences entre la valeur du FTT et celle du débit de réception dues à des phénomènes de *clustering* dans les files d'attente (qui n'existent pas en continu). Il a également été nécessaire d'ajouter un chien de garde de retransmission afin que le protocole puisse détecter les situations de blocage et, ainsi qu'il ne sature pas le réseau.

Le correcteur discrétisé a été implanté dans un protocole de transport nommé Primo (Proportionnel intégral modifié).

7.1.4 Méthode d'évaluation

N'ayant recensé aucune méthodologie de référence permettant une comparaison multi-critères des protocoles de transport, nous avons développé une méthode d'évaluation. Cette méthode d'évaluation est basée sur des simulations effectuées à l'aide de *ns*. L'utilisation de ce simulateur nous autorise à comparer les résultats obtenus par Primo à ceux obtenus par d'autres protocoles (TCP Reno, Sack ,Vegas et TFRC) déjà développés et validés sous *ns*.

Cette méthode définit les contextes de simulations dans lesquels seront testés les protocoles. Deux types de topologies ont été retenus :

- La première topologie utilisée est composée de n paires source/destination (n variant de 2 à 20), dont les sources et les destinations sont reliées *via* deux routeurs intermédiaires,

eux-mêmes reliés entre eux par un lien congestionné (*cf.* figure 5.3 page 108). Ce type de topologie permet d'évaluer l'influence des différents paramètres du réseau : rapport de bande passante des liens, taille et type des files d'attente, temps de propagation et nombre de sources. Afin d'évaluer les protocoles dans diverses situations, les paramètres testés du réseau varient sur des plages de valeurs définies à l'aide de la littérature [48, 2, 19, 10, 27, 66, 31, 75, 33, 47] et des constatations effectuées lors de simulations préliminaires.

– La seconde topologie utilisée possède plusieurs liens congestionnés. Elle est composée de cinq paires source/destination et quatre routeurs intermédiaires (*cf.* figure 5.4 page 109). Suivant les routeurs auxquels sont reliées la source et la destination d'une connexion, le chemin qu'emprunteront ses paquets comprendra de une à trois congestions. Cette topologie permet d'évaluer les capacités d'un protocole à partager équitablement la bande passante alors que la situation est propice à l'inéquité.

La diversité des situations simulées (445 simulations) permet d'évaluer les protocoles de manière objective et non dans un environnement qui pourrait en favoriser certains. La méthode d'évaluation que nous avons définie est multi-critères (*cf.* section 5.4). En effet, elle permet d'évaluer un mécanisme de contrôle de congestion pour l'ensemble des critères d'efficacité que nous avons recensé dans la littérature [48, 2, 19, 10, 27, 66, 31, 75, 33, 47].

Cette méthode a été définie dans le but d'évaluer un protocole préventif qui n'a pas pour objectif d'être déployé sur Internet tel qu'il est. Dans le cadre d'une évaluation portant sur un protocole ayant pour objectif d'être déployé sur Internet, il serait nécessaire de compléter cette méthodologie d'évaluation. En effet, le déploiement sur Internet implique la compatibilité avec TCP New Reno (l'une des versions les plus répandues sur Internet [90]). Il faudra donc ajouter un ensemble de simulations dans lesquelles le nouveau protocole sera mis en concurrence avec TCP New Reno. Ainsi la méthode développée dans cette étude pourrait faire office de banc d'essais pour l'évaluation des nouveaux protocoles qu'ils soient préventifs ou correctifs.

7.1.5 Résultats d'évaluation

Les résultats de l'évaluation des performances nous ont permis d'analyser le comportement des protocoles testés. Cette analyse nous autorise à tirer des conclusions quant à leur capacité à être utilisés pour le transport de données temps réel (vidéo, audio, *etc.*).

Les protocoles TCP Reno et Sack ne semblent pas être adaptés au transport de données multimédia à cause de l'inconstance de leur débit. De plus, leur comportement correctif a tendance à engorger les files d'attente, ce qui a pour effet d'augmenter le délai d'acheminement des données : cela nuit à la qualité d'une transmission temps réel. Notons également que les performances de TCP Reno sont fortement diminuées lorsque des pertes d'acquittements surviennent.

Le protocole TCP Vegas présente, pour le transport de données temps réel, l'avantage d'avoir un comportement préventif. Ce comportement lui permet de maintenir un espace important dans les files d'attente et ainsi de minimiser les délais d'acheminement. On peut également remarquer que ce protocole parvient à lisser le débit de réception, ce qui l'autorise à transporter des données multimédia qui sont sensibles à sa variation. En revanche, les performances de TCP Vegas se dégradent très fortement lorsque des pertes d'acquittements sont constatées.

Le mécanisme de détection des congestions de TFRC est situé du côté du récepteur. Ce type d'implantation lui permet de ne pas être perturbé par les pertes d'acquittements. Le débit

d'émission de TFRC est parfaitement lissé grâce à son système de régulation, qui utilise un délai inter-paquets. Le seul inconvénient de ce protocole, dans le cadre d'une transmission temps réel, est qu'il a tendance à saturer les files d'attente, ceci est dû à son comportement correctif.

Le protocole Primo, en alliant les avantages de TCP Vegas et de TFRC est très efficace pour une transmission temps réel. En effet, son caractère préventif, son contrôle de débit basé sur un délai inter-paquets, ainsi que son système de détection de congestion situé du côté du récepteur, font de ce protocole, un très bon candidat au transport de données temps réel.

7.2 Perspectives

7.2.1 Implantation

Dans cette étude, il a été montré que le mécanisme de contrôle de congestion implémenté par Primo est performant lorsqu'il n'est en concurrence qu'avec lui-même. Cette situation est rendue possible uniquement si Primo se trouve implémenté dans un environnement dédié : réseau privé dédié à ce protocole (cadre dans lequel nous nous sommes placés pour notre étude), réseau *ad'hoc* ou lien entre téléphone cellulaire et réseau Internet.

Réseau *ad'hoc*. Le mécanisme de contrôle de congestion de Primo tel qu'il a été conçu ne peut, actuellement, fonctionner que si le chemin allant de l'émetteur au récepteur ne change pas. Or, généralement les réseaux *ad'hoc* sont utilisés pour leur capacité d'adaptation à la mobilité des machines composant le réseau. Le mécanisme de contrôle de congestion proposé dans Primo devra donc subir quelques modifications pour palier le problème des re-routages. Bien que, dans les réseaux *ad'hoc*, les routes sont amenées à changer fréquemment, on peut supposer que la route allant d'un émetteur à un récepteur aura un temps de propagation stable durant un certain intervalle de temps. Si cet intervalle de temps est suffisamment long (de l'ordre de la minute), un mécanisme d'évaluation périodique de la bande passante et du FTT pourrait être ajouté à Primo. Ainsi, périodiquement, Primo mesurerait un nouveau FTT de référence ainsi que la nouvelle part de bande passante disponible et les utiliserait pour ajuster son débit d'émission durant la période suivante. Notons que cette technique n'est utilisable que si la période des mesures est suffisamment longue. Dans le cas contraire, la dégradation des performances engendrée par l'estimation trop fréquente du FTT de référence (émission d'une rafale de quatre paquets) nuirait fortement au contrôle de congestion de Primo et le rendrait inefficace. Remarquons que, dans la cas d'un réseau où les routes changent très fréquemment, il semble impossible de définir une mécanisme de contrôle de congestion efficace, quel qu'il soit. Notons que les réseaux *ad'hoc* peuvent également être fixes ou à faibles mouvements (*e.g.*, réseaux de capteurs). Une telle configuration est assimilable à un réseau privé dédié à Primo, et dans ce cas, il peut être utilisé tel qu'il est actuellement.

Lien entre téléphone cellulaire et Internet. Les problèmes de mobilité qui peuvent être rencontrés en *ad'hoc*, seront réduits ici au mouvement d'une seule machine. En effet, contrairement à un réseau *ad'hoc*, dans la cas d'une liaison entre téléphone cellulaire et Internet, seul le téléphone peut se déplacer. Or, comme nous le savons, à chaque changement de cellule, des informations sont échangées entre le téléphone et les bornes réceptrices (c'est le *handover*) [65]. On peut aisément imaginer que Primo pourrait profiter de ces informations pour savoir à quel

moment il doit mettre à jour le FTT de référence et ainsi s'adapter à la nouvelle route. Notons que, comme pour les réseaux *ad'hoc*, si le changement de cellule est trop fréquent, les estimations répétées du FTT de référence risquent de nuire à l'efficacité du contrôle de congestion.

Internet. Le déploiement sur Internet de Primo semble compromis pour deux raisons :

- Primo est un protocole préventif. La quasi totalité du trafic Internet (90%) est gérée par diverses versions correctives de TCP (Tahoe, Reno, New Reno, Sack). Or, la cohabitation entre les protocoles correctifs et préventifs paraît impossible. Cependant, il est possible de rendre temporairement agressif un protocole préventif lorsqu'il est en concurrence avec un protocole correctif afin que l'équité soit respectée. En effet, dans [27] les auteurs proposent d'ajouter à TCP Vegas un mécanisme lui permettant de ne plus réduire préventivement le débit d'émission lorsqu'il détecte qu'il est en concurrence avec un protocole correctif.
- Primo nécessite d'être implémenté aux deux extrémités d'une connexion. Cela signifie qu'une machine peut communiquer *via* Primo uniquement si l'autre machine participant à la communication implémente également Primo. Cette restriction imposerait un déploiement massif de Primo sur l'ensemble des serveurs Web, qui ne pourraient alors plus communiquer avec les clients TCP. Cette hypothèse est bien évidemment inenvisageable.

Cependant, un déploiement de Primo serait possible si l'utilisation d'un protocole comme DCCP [64] se généralisait. Ce protocole permet, *via* une négociation à l'initialisation de la connexion, de choisir le type de contrôle de congestion en fonction de l'application. Ainsi, on peut imaginer que des applications de viso-conférence pourraient choisir d'utiliser Primo comme mécanisme de contrôle de congestion, alors que les navigateurs Web continuerait d'utiliser TCP.

7.2.2 Fonctionnalités

Quel que soit le type d'implantation choisi, si Primo doit être utilisé comme protocole de transport principal, il faudra lui ajouter des fonctionnalités. En effet, pour cette étude, nous nous sommes placés dans le cadre d'une transmission multimédia, où les pertes sont tolérées et les données sont émises dès qu'elles sont fournies.

Si Primo venait à être utilisé comme protocole de transport par des applications nécessitant une transmission fiable, il faudrait lui ajouter un mécanisme fiabilisant ses transmissions. L'utilisation d'un mécanisme de détection de pertes et de retransmissions de données, tel que celui de TCP, serait utilisable. Notons que de tels mécanismes n'altéreraient pas les performances du mécanisme de contrôle de congestion de Primo. En effet, ils agiraient uniquement sur le type des données émises (nouvelles données ou ré-émission) et non sur la gestion du débit d'émission. L'ajout d'un mécanisme de ré-ordonnancement des segments serait également indispensable. Ce mécanisme pourrait également être emprunté à TCP. Notons que ces mécanismes devront être désactivés lors de l'utilisation de Primo par des applications multimédia.

Enfin, pour améliorer le rendement de Primo (rapport entre quantité de données transmises et quantité de données utiles), il faudrait inclure un système de *piggybacking*. Ce système, qui permet d'ajouter des données à transmettre à un acquittement, ne sera pas simple à mettre en œuvre. Contrairement aux fonctionnalités précédentes, il serait risquer de ré-utiliser celui

implémenté dans TCP, qui est basé sur le retardement des acquittements. En effet, Primo utilise un délai inter-paquets et non une fenêtre d'émission : les données ne peuvent donc pas être émises à chaque fois qu'un acquittement doit être envoyé. Inversement, si le débit d'émission d'une source est faible (*i.e.*, le délai inter-paquets est élevé), les acquittements ne pourront pas être retardés trop longtemps sous peine de dégrader les performances du contrôle de congestion de l'autre source. Le développement du système de *piggybacking* nécessitera donc un compromis entre l'amélioration du rendement et la diminution des performances du contrôle de congestion.

7.2.3 Amélioration du correcteur

Le correcteur obtenu dans cette étude pourrait donner lieu à la définition d'un correcteur « adaptatif », dont la loi d'adaptation serait définie par l'état du réseau. Ce type de correcteur permettrait d'ajuster la valeur des coefficients de réduction et d'augmentation du débit en fonction de l'état de congestion du réseau. Actuellement, le correcteur utilise deux coefficients de réduction (faible et fort) et un coefficient d'augmentation pour réguler le débit. Il est possible de définir une loi d'adaptation basée sur le degré de congestion du réseau (*i.e.*, l'état du réseau), qui autoriserait une correction automatique et proportionnelle d'un unique coefficient de correction de débit (*i.e.*, le gain). Ainsi, la régulation du débit ne serait plus segmentée en quatre parties (correspondant au risque de sous utilisation du réseau, à la bonne utilisation, à une légère congestion et à une forte congestion). Il est cependant assez complexe de définir une telle loi actuellement ; une étude importante en continu sera nécessaire.

Les travaux menés durant ces trois années d'études nous ont permis d'ouvrir certaines pistes de recherche. De plus, les bons résultats obtenus par le mécanisme de contrôle de congestion proposé confirment l'intérêt à porter à l'application de la théorie de l'automatique à l'algorithmique des télécommunications.

Bibliographie

[1] J. S. Ahn and P. B. Danzid. Packet network simulation: speedup and accuaracy versus timing granularity. *IEEE transactions on networking*, 4(5):743–757, Octobre 1996.

[2] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: Emulation and experiment. In *Proceedings of SIGCOMM*, pages 185–205, 1995.

[3] M. Allman. TCP congestion control with appropriate byte counting. Internet Draft, IETF Internet Engineering Task Force, Novembre 2001.

[4] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window. Request For Comments 2414, RFC editor, http://www.rfc-editor.org/, Septembre 1998.

[5] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window. Request For Comments 3390, RFC editor, http://www.rfc-editor.org/, Octobre 2002.

[6] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. Request For Comment 2581, RFC editor, http://www.rfc-editor.org/, Mars 1999.

[7] A. Altman, C. Barakat, Laborde E., Brown P., and Collange D. Fairness analysis of TCP/IP. In *Proceeding of IEEE Conference on Decision and Control*, Sidney, Australia, Décembre 2000.

[8] P. Bagal, S. Kalyanaraman, and B. Packer. Comparative study of RED, ECN and TCP rate control. Technical report, Departement of ECSE, RPI, http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers/final.ps, 1999.

[9] F. Baker. Requirements for IP version 4 routers. Request For Comment 1812, RFC editor, http://www.rfc-editor.org/, Juin 1995.

[10] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz. The effects of asymmetry on TCP performance. In *Mobile Computing and Networking*, pages 77–89, 1997.

[11] S. Bellovin. Defending against sequence number attacks. Request For Comments 1948, RFC editor, http://www.rfc-editor.org/, Mai 1996.

[12] J. Bolliger, U. Hengartner, and T. Gross. The effectivness of end-to-end congestion control mechanisms. Technical Report 313, ETH Zürich, Février 1999.

[13] J.-C. Bolot and A. U. Shakar. Analysis of a fluid approximation to flow control dynamics. In *Proceeding of IEEE INFOCOM'92*, pages 2398–2407, 1992.

[14] S. Boulahia-Oueslati. *Qualité de service et routage des flots élastiques dans un réseau multiservice*. PhD thesis, ENST, Paris, France, Novembre 2000.

[15] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. Request For Comments 2309, RFC editor, http://www.rfc-editor.org/, Avril 1998.

[16] R.T. Braden. Requirements for Internet hosts, communication layers. Request For Comment 1122, RFC editor, http://www.rfc-editor.org/, Octobre 1989.

[17] L. S. Brakmo, A. C. Bavier, L. L. Peterson, and V. Raghavan. x-sim user's manual (version 1.0). http://www.cs.princeton.edu/courses/archive/spring99/cs461/xsim/xsim.html.

[18] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. Technical report, Departement of Computer Science, The University of Arizona, 1994.

[19] L. S. Brakmo and L. L. Peterson. TCP Vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[20] V. G. Cerf and E. A. Cain. The DoD Internet architecture model. In *Computer Networks and ISDN systems*, number 10 in 7, pages 307–318, 1983.

[21] F. Chatté, B. Ducourthial, D. Nace, and S.-I. Niculescu. Fluid modelling of packets switched networks: perspectives for congestion control. *Special issue of IJSS (International Journal of System Sciences)*, 34(10-11):583–597, Août 2003.

[22] F. Chatté, B. Ducourthial, and S.-I. Niculescu. Robustness issues of fluid approximations for congestion detection in best effort networks. In *Proceeding of IEEE ISCC 2002*, Taormina, Italy, Juillet 2002. IEEE.

[23] F. Chatté, B. Ducourthial, and S.-I. Niculescu. Robustness issues of fluid approximations for congestion detection in best effort networks (extended version). Technical report, Heudiasyc UMR CNRS 6599, Février 2002.

[24] D. Cormer. *TCP/IP : Architecture, Protocoles, Applications*. Interéditions, 1992.

[25] P. B. Danzig, Z. Liu, and L. Yan. An evaluation of TCP Vegas by live emulation. Technical Report 94-588, Computer Science Departement, USC, 1994.

[26] P. De Larminat. *Automatique : commande des systèmes linéaires (2° Ed. revue et augmentée)*. Hermse, Décembre 1996.

[27] A. De Vendictis, M. Bonacci, and Baiocchi A. TCP NewVegas: Providing good TCP performance in both homogeneous and heterogeneous environments. In *Proceeding of 15th ITC Specialist Seminar*, Wuerzburg (Germany), Juillet 2002.

[28] I. El Khayat and G. Leduc. Congestion control for layered multicast transmission. *Networking and Information Systems Journal,*, 3(3-4):559–573, 2000.

[29] I. El Khayat and G. Leduc. Smoothing the TCP rate by learning the delay versus window size dependency. In LNCS Springer-Verlag, editor, *Proceeding. of International Workshop on Multimedia Interactive Protocols and Systems, MIPS*, pages 18–21, Napoli, Italy, Novembre 2003.

[30] K. Fall and S. Floyd. Comparison of Tahoe, Reno and Sack TCP. Technical report, International Computer Science Institute ICSI, http://www-nrg.ee.lbl.gov/nrg-papers.html, 1995.

[31] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, Juillet 1996.

[32] http://www.icir.org/floyd/.

[33] S. Floyd. Connections with multiple congested gateways in packet-switched networks. *ACM SIGCOMM Computer Communication Review*, 21(5):30–47, Octobre 1991.

[34] S. Floyd. TCP and Explicit Congestion Notification (ECN). *ACM Computer Communication Review*, 24(5):10–23, Octobre 1994.

[35] S. Floyd and T. Henderson. New Reno modifications to fast recovery. Request For Comment 2582, RFC editor, http://www.rfc-editor.org/, Mars 1999.

[36] S. Floyd and F. Kevin. Promoting the end-to-end congestion control in the Internet. In *Proceeding of IEEE/ACM Transaction on Networking*, volume 7, pages 458–472, Août 1999.

[37] S. Floyd, J. D. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceeding of SIGCOMM 2000*. ACM, Mai 2000.

[38] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, Février 2001.

[39] S. Floyd and A. Romanow. TCP selective acknowledgement options. Request For Comments 2018, RFC editor, http://www.rfc-editor.org/, Octobre 1996.

[40] G. Fodor, G. Malicsko, M. Piorro, and T. Szymanski. Path optimization for elastic traffic under fairness constraints. In *Proceeding of ITC'01*, pages 2013–2017, 2001.

[41] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback control of dynamic systems*. ADDISON-WESLEY, 1991.

[42] M. Gerla, A. M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo. TCP Westwood: congestion control using bandwidth estimation. In *Proceeding of IEEE Globecom 2001*, Austin, Texas, 2001.

[43] J. Hadi Salim and U. Ahmed. Performance evaluation of explicit congestion notification (ecn) in ip networks. Request For Comments 2884, RFC editor, http://www.rfc-editor.org/, Juillet 2000.

[44] M. Handley, S. Floyd, J. D. Padhye, and Widmer J. TCP friendly rate control (TFRC): Protocol specification. Request For Comments 3448, RFC editor, Janvier 2003.

[45] M. Handley, J. D. Padhye, and S. Floyd. TCP congestion window validation. Request For Comments 2861, RFC editor, http://www.rfc-editor.org/, Juin 2000.

[46] D. Haridas. Rebecca – a receiver based congestion control algorithm for TCP. Master's thesis, University of Illinois, 2000.

[47] S. Hassan and M. Kara. Simulation-based performance comparison of TCP-Friendly congestion control protocols. In *Proceedings of the 16th Annual UK Performance Engineering Workshop (UKPEW2000)*, Durham, UK, Juillet 2000.

[48] U. Hengartner, J. Bolliger, and T. Gross. TCP Vegas revisited. In *Proceeding of the INFOCOM'00*, volume 3, pages 1546–1555, 2000.

[49] J. C. Hoe. Improving the start-up behavior of a congestion control sheme for TCP. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 26,4, pages 270–280, New York, 1996. ACM Press.

[50] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. Analysis and design of controllers for AQM routers supporting TCP flows. In *Proceeding of IEEE Transaction Automatic Control*, volume 47, pages 945–959, 2002.

[51] D. P. Hong, T. Suda, and J. Jungok Bae. Survey of techniques for prevention and control of congestion in ATM networks. In *IEEE International Conference on Communications*, Denver, Juin 1991.

[52] ISO. Basic reference manuel. Technical Report document IS 7498-1, ISO, http://www.iso.org, 1993.

[53] ISO. ISO-TP (OSI transport protocols). Technical Report document 8073, ISO, http://www.iso.org, 1997.

[54] R. Izmailov. Adaptive feedback control algorithms for large data transfer in high-speed networks. *IEEE Transaction Automatic Control*, 40:1469–1471, 1995.

[55] R. Izmailov. Analysis and optimization of feedback control algorithms for data transfers in high-speed networks. *SIAM Journal of Control and Optimization*, 34:1767–1780, 1996.

[56] V. Jacobson. Congestion avoidance and control. In *Proceeding of ACM SIGCOMM'88*, pages 314–329, 1988.

[57] V. Jacobson and R. Braden. TCP extensions for long-delay paths. Request For Comments 1072, RFC editor, http://www.rfc-editor.org/, Octobre 1988.

[58] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. Request For Comments 1323, RFC editor, http://www.rfc-editor.org/, Mai 1992.

[59] V. Jacobson and M. Karels. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM'88*, Novembre 1988.

[60] R. Jain. Congestion control and traffic management in ATM networks: Recents advances and a survey. In *Computer Networks and ISDN systems*, 1995.

[61] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. In *Proceedings of ACM SIGCOMM'87*, pages 2–7, Août 1987.

[62] F. P. Kelly. Mathematical modelling of the internet. *Mathematics unlimited*, pages 685–702, 2001.

[63] G. Kesidis, A. Singh, D. Cheung, and W.W. Kwok. Feasibility of fluid event-driven simulation for ATM networks. In *Proceeding of IEEE GLOBECOM'96*, pages 2013–2017, Anchorage, Alaska, USA, Avril 1996.

[64] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). Internet Draft, IETF Internet Engineering Task Force, Mai 2003.

[65] X. Lagrange, P. Godlewski, and S. Tabbane. *Réseaux GSM-DCS*. Hermes Sciences, 1999.

[66] Y. C. Lai and C. L. Yao. The performance comparison between TCP Reno and TCP Vegas. In *Seventh International Conference on Parallel and Distributed Systems: Workshops (ICPADS'00 Workshops) IEEE*, Iwate, Japan, July 2000.

[67] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong. Fluid simualtion of large scale networks: issues and tradeoffs. In *Proceeding of PDPTA'99*, pages 2136–2142, Las Vegas, NV, USA, Juin 1999.

[68] S. Low, F. Paganini, and J. C. Doyle. Internet congestion control. *IEEE Control System Maggazin*, 22:28–43, 2002.

[69] Q. Ma. *Quality-of-Service Routing in Integrated Service Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, Janvier 1998.

[70] G. Malkin. RIP version 2. Request For Comments 2453, RFC editor, http://www.rfc-editor.org/, Novembre 1998.

[71] S. Mascolo. Classical control theory for congestion avoidance in high speed internet. In *Proceeding of 38th IEEE Conference Decison Control*, pages 2709–2714, Phoenix AZ, 1999.

[72] L. Massoulié. Stability of distributed congestion control with heterogeneous feedback delays. Technical report, Microsoft Research, 2000.

[73] M. Mathis and J. Mahdavi. Forward acknowledgement: Refining TCP congestion control. *Computer Communication Review*, 26(4), 1996.

[74] V. Misra, W. Gong, and D. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceeding of ACM SIGCOMM'00*, 2000.

[75] J. Mo, R.J. La, V. Anantharam, and J. Walrand. Analysis and comparison of TCP Reno and Vegas. In *Proceeding of IEEE INFOCOM'99*, Mars 1999.

[76] J. Mo and J. Walrand. Fair end to end window-based congestion control. In *Proceeding of SPIE'98 Performance and Control of Network Systems II*, pages 55–63, Novembre 1998.

[77] J. Moy. OSPF version 2. Request For Comments 2328, RFC editor, http://www.rfc-editor.org/, Avril 1998.

[78] D. Nace. A linear programming based approach for computing optimal splitable fair routing. In *Proceeding of ISCC 2002*, Taormina, Italy, Juillet 2002. IEEE.

[79] J. Nagle. Congestion control in IP/TCP internetworks. Request For Comment 896, RFC editor, http://www.rfc-editor.org/, Janvier 1984.

[80] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (ds field) in the IPv4 and IPv6 headers. Request For Comments 2474, RFC editor, http://www.rfc-editor.org/, Décembre 1998.

[81] D. Nicol, M. Goldsby, and M. Johnson. Fluid-based simulation of communication networks using SSF. In *Proceeding of European Simulatioin Symposium*, Erlangen-Nuremberg, Germany, Octobre 1999.

[82] S. I. Niculecu, W. Michiels, D. Roose, D. Melchor-Aguilar, K. Gu, and F. Chatté. *Advances in communication control networks*, chapter Delay effects on the asymptotic stability of various fluid models in high performances networks. Springer-Verlag : Heidelberg collection : LNCIS, 2004.

[83] S.-I. Niculescu. *Delay effects on stability: A robust control approach*. LNCIS. Springer-Verlag, Heidelberg, Mai 2001. 383 pages.

[84] S.-I. Niculescu. On delay robustness analysis of a simple control algorithm in high-speed networks. *Automatica*, Mai 2002.

[85] http://www.ncne.org/nimi/.

[86] The ns project, http://www.isi.edu/nsnam/ns/ns-documentation.html. *The ns manual*, Août 2000.

[87] http://www.opnet.com/.

[88] T. Ott, J. Kemperman, and M. Mathis. The stationary behavior of ideal TCP congestion avoidance. Technical report, ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps, 1996.

[89] J. D. Padhye. *Towards a comprehensive congestion control framework for continuous media flows in best effort networks*. PhD thesis, University of Massachusetts, Amherst, USA, 2000.

[90] J. D. Padhye and S. Floyd. Identifying the TCP behavior of web servers. In *Proceeding of ACM SIGCOMM'01*, Août 2001.

[91] V. Paxson and M. Allman. Computing TCP retransmission timer. Request For Comments 2988, RFC editor, http://www.rfc-editor.org/, Novembre 2000.

[92] J. Postel. User Datagram Protocol. Request For Comment 768, RFC editor, http://www.rfc-editor.org/, Août 1980.

[93] J. Postel. Internet Control Message Protocol. Request For Comment 792, RFC editor, http://www.rfc-editor.org/, Septembre 1981.

[94] J. Postel. Internet Protocol. Request For Comments 791, RFC editor, Septembre 1981.

[95] J. Postel. Transmission Control Protocol. Request For Comment 793, RFC editor, http://www.rfc-editor.org/, Septembre 1981.

[96] G. Rajarshi, M. Chen, S. McCanne, and J. Walrand. A receiver-driven transport protocol for the web. In *Fifth INFORMS Telecommunication Conference*, Mars 2000.

[97] K. Ramakrishnan and S. Floyd. A proposal to add Explicit Congestion Notification (ECN) to IP. Request For Comments 2481, RFC editor, Janvier 1999.

[98] K. Ramakrishnan, S. Floyd, and D. Black. The addition of Explicit Congestion Notification (ECN) to IP. Request For Comments 3168, RFC editor, http://www.rfc-editor.org/, Septembre 2001.

[99] S. Ramakrishnan, S. Kalyanaraman, J. Wen, and H. Ozbay. Effect to time delay in network traffic control. In *Proceeding of American Control Conference*, Arlington, Juin 2001.

[100] http://www.cs.cornell.edu/skeshav/real.

[101] R. Rejaie, J. Handely, and D. Estrin. RAP : An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Proceeding of IEEE INFOCOM'99, New-York, NY, USA*, Mars 1999.

[102] I. Rhee, V. Ozdemir, and Y. Yi. TEAR: TCP emulation at receivers - flow control for multimedia streaming. Technical report, DCS, NCSU, Raleigh, CA, USA, Avril 2000.

[103] http://www.ripe.net.

[104] J. Roberts and L. Massoulié. Bandwidth sharing and admission control for elastic traffic. ITC Specialist Seminar Yokohama, Octobre 1998.

[105] P. Rolin, G. Martineau, L. Toutain, and A. Leroy. *Les réseaux : Principes fondamentaux*. Hermes Sciences, 1996.

[106] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. Request For Comments 3550, RFC editor, http://www.rfc-editor.org/, Juillet 2003.

[107] D. Sisalem and A. Wolisz. LDA+: a TCP-friendly adaptation scheme for multimedia communication. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Juin 2000.

[108] D. Sisalem and A. Wolisz. LDA+: Enhanced loss-delay based adaptation algorithm. In *IEEE International Conference on Multimedia and Expo (ICME 2000)*, Juillet 2000.

[109] R. W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Request For Comments 2001, RFC editor, Janvier 1997.

[110] R. W. Stevens. *TCP/IP illustré, Les Protocoles*. Vuibert, 1998.

[111] http://tango.isti.cnr.it.

[112] K. I. Thai, V. Vèque, and S. Znaty. *Architecture des réseaux haut débit*. Hermes Sciences, 1995.

[113] Y. Tobe, Y. Tamura, A. Molano, S. Ghosh, and H. Tokuda. Achieving moderate fairness for UDP flows by path-status classification. In *Proceeding of the IEEE Annual Conf. on Local Computer Networks (LNC'2000), Tampa, FL*, Novembre 2000.

[114] Y. Tobe, Y. Tamura, and H. Tokuda. Detection of change in one-way delay for analysing the path status. In *Proceeding of First Passive and Active Measurement Workshop*, Avril 2000.

[115] A. Veras and M. Boda. The chaotic nature of TCP congestion control. In *Proceeding of IEEE INFOCOM'00*, 2000.

[116] http://www.isi.edu/nsnam/vint/.

[117] J. Widmer, R. Denda, and M. Mauve. A survey on TCP-friendly congestion control. *IEEE Network*, 15(3):28–37, 2001.

[118] K. Zhou, J. Doyle, and K. Glover. *Robust and optimal control.* Prentice Hall, New Jersey, USA, 1995.

Annexe A

Rappels

A.1 *Internet Protocol* (IP)

Le protocole IP est au cœur du fonctionnement d'Internet, il assure, au moyen des adresses IP, le routage des paquets (appelés datagramme IP) à travers les différents réseaux interconnectés. Ce protocole offre une représentation virtuelle qui masque la réalité des réseaux physiques traversés qui peuvent utiliser différents protocoles (*e.g.*, Ethernet, PPP, *etc.*).

Le protocole IP est défini dans la RFC[1] 791 [94], republié dans le STD[2] 5.

Le protocole IP assure sans connexion un service de délivrance de paquets non fiable. Ce qui signifie qu'IP ne garantit pas que les paquets qu'il véhicule arriveront à destination. Les paquets pourront donc être perdus, dupliqués, retardés ou remis dans le désordre sans qu'IP n'informe le récepteur (ni l'émetteur) des problèmes rencontrés.

Datagrammes IP. Les datagrammes IP [94, 80]sont constitués d'une en-tête et de données. L'en-tête est principalement constituée des champs suivants :

- *Version* : ce champ indique la version du protocole IP (en général, il s'agit de la quatrième).
- *Longueur de l'en-tête* : ce champ indique la taille de l'en-tête qui peut être variable mais toujours supérieure à 20 octets.
- *Type de service (TOS[3])* : ce champ indique de quelle manière doit être géré le datagramme (priorité, minimisation du délai d'acheminement, maximisation du débit de transmission, *etc.*).
- *Longueur totale* : ce champ indique la taille du datagramme.
- *Identifications, drapeaux et déplacement de fragment* (*offset*) : ces champs interviennent lorsque le datagramme est fragmenté.
- *Durée de vie* (TTL[4]) : ce champ indique le nombre maximum de routeurs que peut traverser un datagramme avant d'être détruit.

1. *Request For Comments*, statut des *Internet Drafts* (documents de travail de l'IETF, l'*Internet Engineering Task Force*) stabilisés et acceptés par l'IESG, l'*Internet Engineering Steering Group*.

2. *STandard Document*, statut d'une RFC, devenue *Proposed Standard* (au moins deux implémentations du protocole existent), puis *Draft Standard*, après que le protocole décrit ait été éprouvé à large échelle.

3. *Type Of Service*.

4. *Time To Live*.

- *Protocole* : ce champ permet de spécifier quel protocole de plus haut niveau (TCP, UPD, ICMP[5], *etc.*) a engendré l'émission de ce datagramme.
- *Somme de contrôle d'en-tête* : ce champ permet de vérifier l'intégrité de l'en-tête du datagramme.
- *Adresses IP source et destination* : ces deux champs identifient les machines émettrice et réceptrice du datagramme.
- *Options* : ce champ est rarement utilisé car peu de machines sont aptent à gérer les options.

Les datagrammes IP ayant une taille maximale de 655535 octets, les segments fournis par la couche transport peuvent être découpés. Durant leur acheminement, les datagrammes pourront également être fragmentés (par exemple, à un point d'interconnexion entre un réseau rapide et un réseau lent) en plusieurs petits paquets IP qui suivront chacun leur route. Les paquets ainsi fragmentés pourront eux-mêmes être re-fragmentés par la suite. À leur arrivée à la destination finale, les paquets seront ré-assemblés afin de ne former qu'un seul datagramme IP transmissible à la couche transport. Notons que les fragmentations sont coûteuses, il est donc préférable d'envoyer des datagrammes de taille égale au plus petit MTU[6] (*i.e.*, à la plus petite trame de niveau 2) du chemin.

Le protocole IP (étant non fiable) n'assure pas qu'un datagramme envoyé parviendra à destination et les erreurs passagères telle que la corruption d'un datagramme ne sont pas signalées. En revanche, les erreurs dites semi-permanentes (telles que les destructions de paquets dues à l'engorgement d'une file d'attente sur un routeur) feront l'objet d'un message ICMP (émis par le routeur) afin d'avertir la source (*cf.* pargraphe A.3).

Routage. Le routage est l'une des fonctionnalité principale du protocole IP et consiste à déterminer la route que doit suivre un datagramme pour aller de la source à la destination.

Pour déterminer cette route, IP utilise un adressage hiérarchique dont les adresses sont codées sur 32 bits dans la version 4 (IPv4), généralement écrites sous la forme d'un quadruplet d'entiers inférieurs à 256 (par exemple 128.21.123.3). Une adresse IP est décomposable en une adresse de réseau et une adresse de machine dans le réseau (*i.e.*, deux machines appartenant au même réseau auront le début de leur adresse IP en commun). À chaque interface réseau d'une machine correspond une adresse IP qui peut être définie statiquement ou dynamiquement au démarrage de la machine avec le protocole DHCP[7].

Le routage des paquets IP est effectué de « proche-en-proche[8]) », ce qui signifie que l'émetteur d'un datagramme IP ne connaît pas la route complète pour aller jusqu'à la destination. Il ne connaît que l'adresse de la machine suivante sur le chemin source-destination. Pour obtenir cette adresse, chaque machine possède une table de routage, qui met en correspondance l'adresse de destination du datagramme avec l'interface réseau sur lequel il doit être envoyé. Dans le cas où le paquet est envoyé à un routeur (*i.e.*, lorsqu'il n'est pas directement envoyé à la machine destinataire), ce dernier aiguillera à son tour le paquet vers sa prochaine destination. Le routage se fait donc de « proche-en-proche ».

Bien qu'il soit possible de la définir manuellement, la table de routage d'un routeur est généralement obtenue grâce à des protocoles permettant de construire et de maintenir des routes.

5. *Internet Control Messages Protocol*.
6. *Maximum Transmission Unit*.
7. *Dynamic Host Configuration Protocol*.
8. En anglais, *hop by hop routing*.

Les protocoles RIP [9] et OSPF [10] sont les plus utilisés. Le protocole IRDP [11] permet aux machines de régulièrement trouver (ou retrouver) la route par défaut (*i.e.*, un routeur du réseau).

Les datagrammes que reçoit le protocole IP peuvent provenir du protocole de couche 4 (transport) ou d'une interface réseau. Dans le cas où le datagramme provient d'une interface réseau :

- Si l'adresse de destination correspond à celle de l'interface réseau ou à une adresse de diffusion (*broadcast* [12]), IP vérifie que le datagramme est complet (*i.e.*, que tous les fragments ont été reçus) et le transmet à la couche supérieure.
- Sinon, deux cas se présentent : soit IP n'est pas configuré pour fonctionner comme un routeur et le datagramme est détruit, soit le datagramme est ré-émis vers sa destination suivante.

A.2 *User Datagram Protocol* (UDP)

Le protocole UDP est défini dans la RFC 768 [92], republié dans le STD 6 et des précisions ont été apportées dans la RFC 1122 [16].

Caractéristiques Le protocole UDP est un protocole de couche 4 qui utilise IP pour transmettre les datagrammes d'une application d'une machine à une autre. Lors d'une transmission de donnée *via* UDP, rien ne garantit que les datagrammes émis par la source seront reçus par l'application destinataire : UDP (n'utilisant pas d'acquittement) est donc non fiable. De plus, le protocole UDP ne réordonne pas les datagrammes si ceux-ci arrivent dans le désordre.

Ce protocole n'assure pas de contrôle de flux. En effet, si le récepteur est submergé par la quantité de données envoyée par la source, UDP n'en sera pas informé et ne réduira donc pas son débit d'émission. Le protocole UDP n'implémente pas non plus de mécanisme de contrôle de congestion. Enfin, UDP ne nécessite pas l'établissement d'une connexion, il est « orienté » datagramme.

L'absence de ces fonctionnalités (qui sont présentes dans TCP) en font un protocole offrant un faible sur-coût en terme de traitement et de quantité de données de contrôle transmises. Cette caractéristique est très appréciable pour les applications où la rapidité prime sur la fiabilité (*i.e.*, transmission vidéo, flux audio, *etc.*). De plus, certaines applications ont leurs propres mécanismes de fiabilité et/ou de contrôle de flux, il est donc inutile que le protocole de transport qu'elles utilisent les incorpore. Enfin, dans le cas de diffusions *broadcast* et *multicast* [13] où l'absence d'acquittement est fortement conseillée (pour éviter de surcharger le réseau) le protocole UDP est très utilisé.

Fonctionnement. Une application cherchant à émettre des données *via* UDP doit connaître son numéro de port [14]. Le récepteur des données doit être identifié par son adresse IP ainsi que par le numéro de port de l'application destinataire. Le couple adresse IP et numéro de port est

9. *Routing Interface Protocol*.
10. *Open Shortest Path First*.
11. *Internet Router Discovery Protocol*.
12. Diffusion de un vers tous.
13. Diffusions au sein d'un groupe.
14. Point d'entrée dans le système de la machine.

appelé *socket* et une communication en utilise deux (*i.e.*, une paire de *sockets*). Notons que contrairement à TCP, deux applications communicant *via* UDP ne peuvent pas partager une même *socket*.

Les données que fournit l'application sont envoyées en un seul message UDP qui sera fragmenté s'il est trop grand par rapport au réseau IP sous-jacent (*i.e.*, par rapport au MTU). Notons que la fragmentation des données ralentie le transfert et impose au récepteur d'avoir reçu tous les fragments avant de les transmettre à la couche UDP destinataire. Il est donc préférable avec ce protocole d'envoyer des petites quantités de données.

A.3 *Internet Control Messages Protocol* (ICMP)

Le protocole ICMP est défini dans la RFC 792 [93] et inclue dans le STD 5 (standard IP). Des précisions lui ont été apportées dans les RFC 1122 et 1812 [16, 9] et il a été redéfini et clarifié dans le STD 2.

But. Le protocole ICMP permet de gérer les erreurs au niveau de la couche IP qui est non fiable. Le but de ce protocole n'est pas de fiabiliser IP, mais de lui fournir (à lui ou à couche supérieure) des informations sur des erreurs détectées dans les routeurs. Certaines erreurs sont temporaires (*e.g.*, erreur de la somme de contrôle) et ne nécessitent donc pas d'être signalées. En revanche, d'autres erreurs dites « semi-permanentes » méritent d'être signalées afin d'assurer le bon fonctionnement du réseau. Parmi ces erreurs, on peut citer : réseau inaccessible, machine inaccessible, échec de routage, réseau de destination inconnu, réseau inaccessible pour ce type de service, fragmentation nécessaire alors que le bit de non fragmentation est positionné à « vrai », *etc.*

L'ensemble des erreurs que signale ICMP, permet à IP d'adapter son comportement afin de résoudre les problèmes rencontrés et ainsi d'améliorer l'utilisation des capacités du réseau. Par exemple, IP n'a aucun intérêt à surcharger le réseau en envoyant des données vers une machine inaccessible. Le protocole ICMP a pour but de reporter les erreurs « semi-permanentes ». Il peut également être utilisé pour tester le réseau, comme, par exemple, avec la commande *ping*. Mais le protocole ICMP ne sera pas utilisé pour signaler un problème concernant des datagrammes de *broadcast*, de *multicast*, de ICMP lui-même, *etc.*

Fonctionnement. Les erreurs du protocole IP sont signalées *via* des messages ICMP, eux-mêmes véhiculés par le protocole IP. C'est pourquoi, dans certains cas, le protocole ICMP n'est pas utilisé, afin de ne pas aggraver les problèmes.

Le protocole ICMP est constitué d'une liste de comportements divers, associées à différents messages, qui sont identifiés par un numéro (*Message Type*). Trois sortes de messages peuvent être distingués : les requêtes, les réponses aux requêtes et les messages signalant une erreur. Le format des messages ICMP varie en fonction du comportement associé (*i.e.*, en fonction du numéro de message).

Les protocoles n'incluant pas de contrôle de leurs communications (*e.g.*, UDP) pourront pleinement tirer bénéficie d'ICMP. Par contre, certains messages ICMP sont redondants avec les fonctionnalités d'un protocole tel que TCP, qui contrôle déjà ses communications.

Annexe B

Équivalence modèles continu et discret

B.1 Code source du protocole utilisé pour le modèle continu sous matlab

B.1.1 Filde-attente.m

```
function [attente, val_q] = file_attente_2(debit_0, debit_1,
                            mu, tdf,file_pre,pas)
% debit_X : vecteur des valeurs du débit de la source X
% mu : débit de file d'attente
% tdf(X) : heure de l'appel de la fonction moins les temps
%          de transite dF de la source X
% file_pre : taille de la file lors de la dernière itération
% pas : durée du pas de simulation en ms

if ((tdf(1) <= 0) & (tdf(2) <= 0))
% Test si aucun paquet ne peut avoir atteint la file
   val_q = 0;
else
  val_q = file_pre; % récupère la taille précédente de la file
  if (tdf(1) > 0)
  % Testes si des données de la connexion 1 peuvent
  % être arrivés dans la file
     val_q =  val_q + debit_0(tdf(1)) * (pas/1000);
     % Ajoute dans la file les données arrivées (en fonction
     % du débit de la connexion 1)
  end;

  if (tdf(2) > 0)
  % Testes si des données de la connexion 2 peuvent
  % être arrivés dans la file
     val_q =  val_q + debit_1(tdf(2)) * (pas/1000);
     % ajoute les données arrivées dans la file en fonction
     % du débit de la connexion 2
  end;
  val_q = val_q - mu * (pas/1000);
  % Retrait de la file des données émises au débit mu
end;

if (val_q < 0)
  % Test si le file n'a pas une taille négative
  % (cas ou le débit d'enter est inférieur à celui de soutie)
  val_q = 0;
end;

% Calcul du temps que passera un paquet dans la file d'attente
attente =  (val_q / mu) * 1000; % passage en ms
```

B.1.2 Controle-debit.m

```
function [debit_0 , debit_1 , Time , taille_file] = sources_fluid_2(
                              source_0 , source_1 , mu , duree, pas)

% Les paramètres sources_0 à source_1 sont des vecteurs qui contiennent
% les valeurs suivante
% source_X(1) = alphaX : coefficient utilisé dans la phase croissante
%                        du débit de la source 1(en kbits/(s^2))
% source_X(2) = betaX : coefficient utilisé dans la phase décroissante
%                       du débit de la source 1
% source_X(3) = dFX : temps de transite entre la source X et la file
%                     d'attente (en ms)
% source_X(4) = dRX : temps de transite entre la file d'attente et le
%                     retour à la source X (en ms)
% Attention dFX et dRX doivent inclure les temps de mise sur le réseau
% source_X(5) = toleranceX :  attente tolérée dans la file (en ms)
% mu : debit de file d'attente (en kbits/s)
% duree : durée en secondes de la simulation (en s)
% pas : granularité de la simulation (en ms)

if ((pas ~=  5) & (pas ~= 1))
   disp(['Il faut que le pas de simulation soit un de 5 ou de 1 ms']);
   break;
end;

fid_0 = fopen('debit_f_0.tr','w');
fid_1 = fopen('debit_f_1.tr','w');

if (pas == 5)
  j = 1;
else
  j = 5;
end;

%Initialisation des variables
disp(['Début de l''initialisation']);

% Granulariation de la durée de simulation
duree = duree * 1000 / pas;
% Céartion du vecteur de temps granularisé
Time = [0 : duree];
% Récupération des coefs de croissance décroissance de la source 0
alpha_0 = source_0(1);
beta_0 = source_0(2);
% Délai dF en fonction de la granularité
dF_0 = source_0(3) / pas - mod(source_0(3) / pas, 1);
% Délai dR en fonction de la granularité
dR_0 = source_0(4) / pas - mod(source_0(4) / pas, 1);
% Temps aller retour (RTT) en fonction de la granularité
d_0 = dF_0 + dR_0;
% Récupération ce la tolérance de la source 0
tolerance_0 = source_0(5);
% Initialisation de la période de croissance décroissance
periode_0 = 0;
% Initialisation du nombre de périodes
nb_periode_0 = -1;
% Initilaisation d'un variable temporaitre utilisée
% pour caculer la période
t_temp_0 = -1;
% Drapeau utilisé pour savoir dans quelle phase on se trouve
% (si debit_0_flag = 1 : phase croissante linéaire,
% si debit_0_flag = 0 : phase décroissante exponentielle)
debit_0_flag = 1;
% Sert à connaitre l'heure du dernier passage ne phase
% croissante linéaire
debit_0_lin_temp = 0;
% Sert à connaitre l'heure du dernier passage ne phase
% décroissante exponentielle
debit_0_expo_temp = 0;
```

```
% Sert à connaitre la valeure du débit lors du dernier passage
% en phase décroissante exponentielle
debit_0_max_temp = 0;
% Sert à connaitre la valeure du débit lors du dernier passage
% en phase linéaire croissante
debit_0_min_temp = 0;
% Sert à connaitre la plus petite valeure du débit atteinte
% durant la simulation
debit_0_min = mu;
% Sert à connaitre la plus grande valeure du débit atteinte
% durant la simulation
debit_0_max = 0;

% Récupération et initialisation des paramètres de la source 1
alpha_1 = source_1(1);
beta_1 = source_1(2);
dF_1 = source_1(3) / pas - mod(source_1(3) / pas, 1);
dR_1 = source_1(4) / pas - mod(source_1(4) / pas, 1);
d_1 = dF_1 + dR_1;
tolerance_1 = source_1(5);
periode_1 = 0;
nb_periode_1 = -1;
t_temp_1 = -1;
debit_1_flag = 1;
debit_1_lin_temp = 0;
debit_1_expo_temp = 0;
debit_1_max_temp = 0;
debit_1_min_temp = 0;
debit_1_min = mu;
debit_1_max = 0;

% Initialisation de la taille maximale atteinet par la file
taille_max_file = 0;
i_max= size(Time);
% Le vecteur taille_file(i) représentera la taille de la
% file d'attente à l'inatant i
% Les vecteurs debit_0(i) et debit_1(i) le débit des sources
% à l'intant i
% Initialisation du  vecteur duree_attente_sortie(i) le temps passé
% dans la file d'attente par un pquet en sortant à l'instant i
duree_attente_sortie(1:i_max(2))=-1;

disp(['Fin de l''initialisation']);

disp(['Début de la simulation']);

for i=1:i_max(2)
   t = Time(i);  % Instant t en ms
   t_sec = (t * pas) / 1000; % Instant t en s

   % Calcul de l'heure de départ d'un paquet arrivant dans
   % la file à l'instant t
   Td(1) = (t - dF_0);
   Td(2) = (t - dF_1);
   % Calcul du temps passé dans la file d'attente
   [duree_attente(i), taille_file(i)] = file_attente_2(debit_0,
     debit_1, mu, Td, taille_file_reel(i-1), pas);
   % Calcul de l'emplacement où ranger le temps que passera un
   % paquet dans la file d'attente
   ds = (duree_attente(i)/pas)- mod((duree_attente(i)/pas),1);
   ds = ds+i;
   duree_attente_sortie(ds) = duree_attente(i);
   % Vérification que la ième case du vecteur durée attente est bien remplie
   if (duree_attente_sortie(i)==-1);
     duree_attente_sortie(i)=duree_attente_sortie(i-1);
   end;
   % Mise à jour la taille maximale qu'a atteint la file d'attente
   if (taille_file(i) > taille_max_file)
     taille_max_file = taille_file(i);
```

```
   end;

  if (t-d_0) >= 0
  % Test si un acquittement peut avoir atteint la sources
    T_0 = (t - dR_0);
    % Calcul l'heure à laquelle le paquet ayant généré
    % l'acquittement à quitté la file d'attente
  else
    T_0 = 1;
    % Valeur par défaut
  end;

  if (duree_attente_sortie(T_0) <= tolerance_0)
  % Test si la tolérance de durée d'attenet dans la file n'a
  % pas été dépassé
    if (debit_0_flag == 0)
     % Test si l'on se trouve en phase décroissante
       % Passage en phase croissante
       debit_0_flag = 1;
       % Récupération de la valeur du débit avant de changement de phase
       debit_0_min_temp = debit_0(i-1);
       % Récupération de l'heure du changement de phase
       debit_0_lin_temp = t_sec;
        if (debit_0_min_temp < debit_0_min)
         % Test si le débit d'émission est le plus bas enregisté depuis
         % le début de la simulation
           debit_0_min = debit_0_min_temp;
        end;
     end;
     % Mise à jour du débit d'émission en phase croissante
     debit_0(i) = alpha_0 * (t_sec - debit_0_lin_temp) + debit_0_min_temp;
  else
    % Cas où la tolérance à été dépassé
     if (debit_0_flag == 1)
      % Test si l'on se trouve en phase croissante
       % Passage en phase croissante
       debit_0_flag = 0;
       % Récupération de la valeur du débit avant de changement de phase
       debit_0_max_temp = debit_0(i-1);
       % Récupération de l'heure du changement de phase
       debit_0_expo_temp = t_sec;
       if (t_temp_0 == -1)
         % Test s''il s'agit du premier passage en phase décroissante
         t_temp_0 = debit_0_expo_temp;
       end;
       % Calcul de la période (phases croissante + décroissante)
       periode_0 = periode_0 + debit_0_expo_temp - t_temp_0;
       % Calcul du nombre de périodes
       nb_periode_0 = nb_periode_0 + 1;
       % Récupération de l'heure de début de la nouvelle période
       t_temp_0 = debit_0_expo_temp;
       if (debit_0_max_temp > debit_0_max)
         % Test si le débit d'émission est le plus haut enregisté depuis
         % le début de la simulation
         debit_0_max = debit_0_max_temp;
       end;
     end;
    % Mise à jour du débit d'émission en phase décroissantecroissante
    debit_0(i)= debit_0_max_temp*exp(-(t_sec-debit_0_expo_temp)/beta_0);
    if (debit_0(i) < 3)
     % Test si le nouveau débit n'est pas trop faible
       debit_0(i) = 3;
    end;
  end;

% La partie de  code suivante est idem à la précédente mais
% s'applique à la connexion 1

  if (t - d_1) >= 0
```

```
    T_1 = (t - dR_1);
  else
    T_1 = 1;
   end;

  if (duree_attente_sortie(T_1) <= tolerance_1)
    if (debit_1_flag == 0)
       debit_1_flag = 1;
       debit_1_min_temp = debit_1(i-1);
       debit_1_lin_temp = t_sec;

       if (debit_1_min_temp < debit_1_min)
          debit_1_min = debit_1_min_temp;
       end;
    end;
    debit_1(i) = alpha_1*(t_sec-debit_1_lin_temp)+debit_1_min_temp;

  else
    if (debit_1_flag == 1)
       debit_1_flag = 0;
       debit_1_max_temp = debit_1(i-1);
       debit_1_expo_temp = t_sec;
       if (t_temp_1 == -1)
          t_temp_1 = debit_1_expo_temp;
       end;
       periode_1 = periode_1 + debit_1_expo_temp - t_temp_1;
       nb_periode_1 = nb_periode_1 + 1;
       t_temp_1 = debit_1_expo_temp;
       if (debit_1_max_temp > debit_1_max)
         debit_1_max = debit_1_max_temp;
       end;
    end;
    debit_1(i)= debit_1_max_temp*exp(-(t_sec-debit_1_expo_temp)/beta_1);
    if (debit_1(i) < 3)
       debit_1(i) = 3;
     end;
  end;

% Ecrit dans le fichier fid_X la valeur actuel du débit d'émission
fprintf(fid_0,'%f \n',debit_0(i));
fprintf(fid_1,'%f \n',debit_1(i));
end;
disp(['Fin de la simulation']);

%Fermeture des fichier fid_X
fclose(fid_0);
fclose(fid_1);
```

B.2 Code source du protocole utilisé pour le modèle discret sous ns

B.2.1 Protocole-Discret.h

```
// Fabien Chatté
// UTC Mai 2001

#ifndef ns_mon_protocole_h
#define ns_mon_protocole_h
#define TAILLE_NOM 10
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "random.h"
#include <math.h>

struct hdr_mon_protocole {
  char ret;
  double send_time;
  double c; // variance de la taille des paquets
  // Header access methods
  static int offset_; // required by PacketHeaderManager
  inline static int& offset() { return offset_; }
  inline static hdr_mon_protocole* access(const Packet* p)
    {
      return (hdr_mon_protocole*) p->access(offset_);
    }
};


class Mon_protocoleAgent : public Agent
{
 private:
  char monNom[TAILLE_NOM]; // Nom de l'agent
  double tempo; // temps à attendre avant le entre le dernier
                //paquet envoyé et le suivant
  int premier_pkt_data;  // flag servant à savoir si l'on à
                         //déjà reçu ou non un paquet de donée
  int premier_ack; // flag servant à savoir si l'on à déjà reçu
                   // ou non un ack
  double rtt_min; // Valeur du premier rrt mesuré
  double rtt_inst; // Valeur du dernier rtt mesuré
  int phase; // flag servant à savoir dans quelle phase du débit crois-
             // sante (phase=1) ou décroissante (phase=0) on se trouve
  double min_rate; // Débit lors du dernier changement de la phase
                   // décroissante vers une phase croissante du débit
  double max_rate; // Débit lors du dernier changement de la phase crois-
                   // sante vers une phase décroissante du débit
  double rate; // Débit actuel en bits par seconde
  double t_max_rate; // heure du changement de la dernière phase crois-
                     // sante vers une phase décroissante du débit
  double  t_min_rate; // heure du changement de la dernière phase
                      //décroissante vers une phase croissante du débit
  double t_recv; // heure de réception du dernière aquittement
  double alpha, beta; // coefficients utiliser dans les équation
                      // gérant le débit
  double c; // variance de la taille des paquets (0 par défaut)
  double last_recv; // heure de réception du dernier paquet de données
  double inter_recv; // Délai inter réception de paquets de données
  double tolerance; //tolerance sur la comparaison des rtt et rttmin
  int nb_pkts_recv;//nombre de paquete reçus
  double periode_tot, t_temp, periode_temp; // Variables tempopraires
                                    //servant au calcule de la période
  int nb_periode; // nombre de période
```

```
  double deb_max; // Débit max atteint durant la simulation


public:
  Mon_protocoleAgent(const char*, const char*, const char*, const char*) ;
  virtual int command(int argc, const char*const* argv);
  virtual void recv(Packet*, Handler*);

  char *nomme(const char *nom) //positionne ou retourne le nom
   {
    if( nom != NULL )
      strcpy(monNom, nom);
    return monNom;
   }

  void update_rate() // met à jour le débit d'émission
  double  inter_packets_delay() // calcul le délais interpaquets
};
#endif
```

B.2.2 Protocole-Discret.cc

```
// Fabien Chatté
// UTC Mai 2001

#include "mon_protocole.h"
#include <stdlib.h>

int hdr_mon_protocole::offset_;
static class Mon_protocoleHeaderClass : public PacketHeaderClass {
public:
    Mon_protocoleHeaderClass() : PacketHeaderClass("
        PacketHeader/Mon_protocole", sizeof(hdr_mon_protocole)) {
    bind_offset(&hdr_mon_protocole::offset_);
    }
} class_mon_protocolehdr;

static class Mon_protocoleClass : public TclClass {
public:
Mon_protocoleClass() : TclClass("Agent/Mon_protocole") {}
TclObject* create(int argc, const char*const* argv) {
  if (argc == 8)
    return (new Mon_protocoleAgent(argv[4],argv[5],argv[6], argv[7]));
  else
    printf ("Il faut entrer lors de la création d'un agent, le
       nom du noeud, les coefficients alpha et beta, et la tolerance");

}
} class_mon_protocole;




Mon_protocoleAgent::Mon_protocoleAgent(const char* nom, const char* A,
const char* B,const char* tol) : Agent(PT_MON_PROTOCOLE)
{
  if( strlen(nom) <= TAILLE_NOM)
    (void)strcpy(monNom, nom);
  else
    {
      (void)strncpy(monNom, nom, TAILLE_NOM-1);
      (void)strcat(monNom, ".");
    }

  // Initialisation des variables
  // Pour plus de détail sur les variable voir le .h
  c=0;
  tempo = 0;
  phase = 1;
```

```
  premier_pkt_data = 1;
  rtt_min = 0;
  rtt_inst = 0;
  premier_ack = 0;
  min_rate = 0;
  rate = 0;
  max_rate = 0;
  t_max_rate = 0;
  t_min_rate = 0;
  alpha= atof(A)*1000 ; // Passage de kbits/s en bits/s
  beta = atof(B) ;
  last_recv = 0;
  inter_recv = 0;
  tolerance = atof(tol) + 0.0001;
  nb_pkts_recv = 0;
  periode_tot = 0;
  nb_periode = -1;
  t_temp = 0;
  deb_max = 0;

// Partage des variables
  bind("packetSize_", &size_);
  bind("rtt_min_",&rtt_min);
  bind("rtt_inst_", &rtt_inst);
  bind ("inter_paquet_",&tempo);
  bind ("rate_",&rate);
  bind ("max_rate_", &max_rate);
  bind("c_",&c);
  bind("inter_recv_",&inter_recv);
  bind("nb_pkts_recv_",&nb_pkts_recv);
  bind("nb_periode_",&nb_periode);
  bind("periode_tot_",&periode_tot );

 printf("Création de l'agent Dec net %s, ayant pour coefficient
  alapha : %f kbits/s^2 et beta : %f s, ayant pour tolérance : %f\n",
  nom, alpha, beta,  tolerance);
 fflush(stdout);
}

int Mon_protocoleAgent::command(int argc, const char*const* argv)
{

  if(argc == 2)
    {
      if (strcmp(argv[1], "resultats") == 0)
      {
       periode_tot = periode_tot / nb_periode ;
       printf("\n La periode de l'agent %s est de %f s, et il y a eu %d
        périodes \n",monNom, periode_tot, nb_periode);
       printf("L'agent %s à eu un débit max de %f bit/s\n",
        monNom, deb_max);
       return TCL_OK;
      }

      if (strcmp(argv[1], "nb_paquets") == 0)
      {
       printf("\n L'agent %s a reçu %d paquets\n",monNom, nb_pkts_recv);
       return TCL_OK;
      }

    if (strcmp(argv[1], "send") == 0) {
      // Creation d'un nouveau paquet
      Packet* pkt = allocpkt();
      // Acces au header du paquet
      hdr_mon_protocole* hdr = hdr_mon_protocole::access(pkt);
      // On met ret à 0 car c'est un paquet de donnée (ainsi le
      // récepteur envera un ack)
      hdr->ret = 0;
      // On indique l'heure d'émission du paquet afin de calculer
```

```
      // le rtt lors du retour de l'ack
      hdr->send_time = Scheduler::instance().clock();
      // Envoie du paquet
      send(pkt, 0);
      // Appel de la fonction inter_packets_delay pour calculer
      // le nouveau delai inter-paquets
      tempo =  inter_packets_delay();
      Tcl& tcl = Tcl::instance();
      // Appel récursif de la commande send à la date :
      // now + le delai calculer précédemment
      tcl.evalf ("
      global ns
      set now [$ns now]
      $ns at [expr $now+ %f] \"$%s send\"
      ",tempo,monNom);
      return (TCL_OK);
    }
  }
  return (Agent::command(argc, argv));
}

void Mon_protocoleAgent::recv(Packet* pkt, Handler*)
{
  // Acces au  header IP :
  hdr_ip* hdrip = hdr_ip::access(pkt);
  // Acces au header Mon_protocole headert:
  hdr_mon_protocole* hdr = hdr_mon_protocole::access(pkt);
  // Vérification du type de paquet (donnée ou ack)
  if (hdr->ret == 0)
    {
      // C'est un paquet de donnée donc on envoie un ack
      t_recv = Scheduler::instance().clock();
      // Récupération de l'heure d'emission du paquet pour pouvoir
      // l'indiquer dans l'ack
      double stime = hdr->send_time;
      // Destruction du paquet reçu
      Packet::free(pkt);
      // Création d'un nouveau paquet
      Packet* pktret = allocpkt();
      hdr_mon_protocole* hdrret = hdr_mon_protocole::access(pktret);
      // On met ret à 1 pour spécifier qu'il s'agit d'un ack
      hdrret->ret = 1;
      // On indique dans l'entête l'heure d'émission
      // récupérée précédemment
      hdrret->send_time = stime;
      // Test s'il s'agit du premier paquet de donnée reçu
      nb_pkts_recv ++; // incrémente le nombre de paquets reçus

    if  (premier_pkt_data == 1)
      {
        // c'est le premier paquetreçu, on récupère l'heure
        // de sa réception dans la variable last_recv
        last_recv = t_recv;
        // On met le flag à 0 pour spécifier qu'un
        // premier ack à été reçu
        premier_pkt_data = 0;
      }
    else
      {
        // On calcul le temps écoulé entre la réception de ce
        // paquet et celle du dernier.
        inter_recv = t_recv - last_recv;
        // On met à jour l'heure de la dernière réception
        last_recv = t_recv;
      }
     // Envoi de l'ack
    send(pktret, 0);
    }
```

```
else
  {
  // Réception d'un ack
  // Récupération de l'heure de la réception de l'ack
  t_recv = Scheduler::instance().clock();

  // Test s'il s'agit du premier ack reçu
  if (premier_ack == 0)
    {
      // Si c'est le premier ack, on calcul son rtt qui servira de
      // référence pour savoir si la file est vide ou non
      premier_ack = 1;
      rtt_min = t_recv - hdr->send_time;
      printf("\n Rtt_min = %f pour la connexion ayant pour
        émetteur %s\n",(rtt_min), monNom);
      // On se place dans une phase croissnte du débit
      phase = 1;
    }
  else
    {
     // Ce n'est pas le premier ack reçu
     // On calcul le rtt
     rtt_inst = t_recv - hdr->send_time;
     // Si le rtt est supérieur au rtt_min (la référence)
     // c'est que la file n'est pas vide
     if ((rtt_inst > (rtt_min + tolerance))  && (phase==1))
       {
         if (nb_periode < 0)
          {
            t_temp = t_recv;
            nb_periode = 0;
            periode_tot = 0;
          }
         else
          {
            periode_temp = t_recv - t_temp;
            if (periode_temp >= 1)
             {
              periode_tot = periode_tot + periode_temp;
              t_temp = t_recv;
              nb_periode++ ;
              printf("%s : changement de phase à t= %f s, %d ème
              période\n", monNom, t_recv, nb_periode);
             }
          }

          update_rate();
          // On note le débit atteint au moment où l'on détecte
          // que la file est non vide
          max_rate = rate;

          if (max_rate > deb_max){
             deb_max = max_rate;
            }
          // On note l'heure de la détection que la file est non vide
          t_max_rate =  t_recv;
          // passage en phase décroissante
          phase = 0;
          printf("A t = %f\t %s passe en phase décroissante le taux max
           est de : %f\n",t_recv,monNom,  rate);
       }

     // Si le rtt est égale au rtt_min (la référence) c'est
     // que la file est vide
     if ((rtt_inst <= (rtt_min + tolerance))&&(phase==0))
       {
         update_rate();
         // On note le débit atteint lors de la
         // détection que la file est vide
```

```
           min_rate = rate;
           // On note l'heure de la détection que la file est  vide
           t_min_rate = t_recv;
           // passage en phase croissante
           phase = 1;
          }
       }

    // Destruction du paquet reçu
    Packet::free(pkt);
   }
}

Mon_protocoleAgent :: update_rate()
  {
   // Mise à jour du débit d'émission
   float t;
   t = Scheduler::instance().clock();
   if (phase==1)
    {
     rate = min_rate + alpha * (t-t_min_rate);
    }
   else
     {
      rate = max_rate * exp (-((t-t_max_rate)/beta));
      if (rate < 3000)
rate = 3000;
     }
   }

Mon_protocoleAgent :: inter_packets_delay()
 {
  // calcul du délia inter-paquet
  float delais_interpaquet ;
  int size_bit;
  float t;
  t = Scheduler::instance().clock();
  update_rate();
  size_bit = (int)size_ * 8;
  delais_interpaquet = (size_bit / rate);
  return(delais_interpaquet);
 }
```

Annexe C

Code source de Primo

C.1 Code source du correcteur de Primo utilisé sous matlab

```
function [debit] = correcteur_pi(FTT_inst,RTT_inst, t, debit, correcteur, deb_in)
% Cette fonction calcule le débit auquel elle devra émettre, en fonction des informations récuprées.
% Premier seuil de détection de congestion (faible congestion)
dep_sup = debit.FTT_cons(t-RTT_inst)*correcteur(1).dep_sup;
% Premier seuil de détection de congestion (forte congestion)
n_dep_sup = dep_sup * correcteur(1).sueil_red;
% Sueil de détection d'un risque de sous utilisation du réseau
dep_inf = debit.FTT_cons(t-RTT_inst)*correcteur(1).dep_inf;

% Récupération du débit précédent
x_prec = debit.x(t-RTT_inst);
%Comparaison du FTT consigne et du FTT effectif
debit.var_FTT(t) = FTT_inst - debit.FTT_cons(t-RTT_inst);

% Vérification que la simulation dure depuis suffisament longtemps
% pour pouvaoir exploiter son historique
if (t-debit.retard>0)
   % Si c'est la cas, utiliser la variation du FTT mesurée il y a
   % retard seconde
   var_FTT = debit.var_FTT(t-debit.retard);
else
   % Sinon considérer que l'erreur est nulle
   var_FTT = 0;
end;

% Evaluation du dégré de congestion
if (var_FTT > dep_sup)
     if (var_FTT > n_dep_sup)
       % Si la congestion est importante, utiliser le coefficiebt de
       % réducti reduc 1
        var_deb = x_prec - correcteur.reduc1*deb_in;
     else
       % Si la congestion est faible utiliser reduc2
        var_deb = x_prec - correcteur.reduc2*deb_in;
     end;
     X = x_prec + correcteur.alpha_deb  * var_deb;

  else
     if (var_FTT < dep_inf)
       % Risque de sous-utilisation du réseau
        var_deb = x_prec - correcteur.acroiss*deb_in;
       % Le débit augmente en utilisant le coefficient alpha_deb
        X = x_prec + correcteur.alpha_deb  * var_deb;
     else
        % Cas ou le réseau n'est ni congestion ni en situation de
        % sous-utilisation potenielle
        X=x_prec;
```

```
     end;
  end;

% Vérification que la part de bande passante disponible n'a pas augmenté.
if (debit.FTT_ref > FTT_inst)
   % si elle a augmenté, mettre a jour le FTT de référence et la consigne
   debit.FTT_cons(t) = FTT_inst*correcteur.dep_ftt;
   debit.FTT_cons(t) = FTT_inst;
else
   % Sinon continuer d'utiliser la même consigne
   debit.FTT_cons(t) = debit.FTT_cons(t-1);
end;
```

C.2 Code source du protocole Primo

C.2.1 Primo.h

```
// Fabien Chatté
// UTC Mai 2003

#ifndef ns_primo_h
#define ns_primo_h
#define TAILLE_NOM 10
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "random.h"
#include <math.h>

struct hdr_primo {
  int ret;          // Détermine si le paquet contient un acquittement (oui = 1, non = 0)
  double heure_ems; // Heure d'émission du paquet
  double deb_ems;   // Debit d'émission du paquet
  double deb_recp;  // Valeur du dernier débit de réception mesuré
  double deb_recv_liss; // Valeur lissé du débit de réception
  double FTT_inst;      // Valeur du dernier FTT mesuré
  double deb_ems_don;   // Débit d'émission du paquet de données ayant engendrée cet acquittement
  double heure_ems_don; // Heure  d'émission du paquet de données ayant engendrée cet acquittement
  double FTT_cons; // Valeur du FTT consigne lors de l'émission du paquet ayant engendré
                   // l'émission de l'ack
  int seqno;       // Numéro de seéquence du paquets
  int acked_seqno; // Numéro de séquence du paquet acquitté
  // Header access methods
  static int offset_; // Nécessité par le PacketHeaderManager
  inline static int& offset() { return offset_; }
  inline static hdr_primo* access(const Packet* p) {
  return (hdr_primo*) p->access(offset_);
   }
};

class primoAgent;

class primo{
    // Valeur calculée ou récupérée
    double DEBIT;           // Débit d'émission actuel
    double *VAR_FTT;        // Tableau (de taille REATRD_G) contenant les valeurs de la variation du FTT
    int RETARD;             // Retard utilisé par le correcteur PI
    int RETARD_G;           // Retard prenant en compte la granularité de l'horloge
    int RETARD_M;           // Retard modulo utiliser pour gérer le tableau circulaire DEBIT
    int EMPLACEMENT_PREC;   // Varibale permettant de gérer le tableau circulaire

    public :
      primo();  // Constructeur
      // Initilaisation du tableau et des variables
      double primo_init(double, double, double, double, double, double, int, double, double, double);
      // Calcul du débit en fonction des dernières valeurs reçus
      double calcul_debit(double, double, double, double, double, double, double, double, double,
                          double, double, double);
      // Calcul et rangement de la dernière mesure de la variation du FTT
      double rangement_var_FTT(double, double, double, double);
      // Fonctions d'accèes et mise à jour de variables privées
      void set_DEBIT(double);
      double access_retard();
      double access_debit();
  };


class IPG_Timer : public TimerHandler {
 public:
```

```
  IPG_Timer(primoAgent *a) : TimerHandler() { a_ = a; } // Constructeur

 protected:
  virtual void expire(Event *e); // Fonction déclenchée en cas d'expiration du timer
  primoAgent *a_;
};

class PrimoTimer : public TimerHandler {
 public:
  PrimoTimer(primoAgent *a) : TimerHandler() { a_ = a; } // Constructeur

 protected:
  virtual void expire(Event *e); // Fonction déclenchée en cas d'expiration du timer
  primoAgent *a_;
};

class primoAgent : public Agent
{
private:

  /**********************Varialbles de paramétrage************************/

  char monNom[TAILLE_NOM]; // Nom de l'agent
  double alpha_deb;        // Gain proportionnel du correcteur PI
  double tau;              // Retard pur
  double dep_FTT;          // Dépassement souhaité du FTT
  double dep_deb;          // Dépassement souhaité du débit de réception
  double borne_sup;        // Borne permettant de calculé le seuil de détection de congestions
  double borne_inf;        // Borne permettant de calculé le sueil de sous utilisation du réseau
  double reduc1;           // Coefficient de réduction du débit en cas de congestion importante
  double reduc2;           // Coefficient de réduction du débit en cas de petite congestion
  double augmente1;        // Coefficient d'acroissement du débit en cas de sous utilisation du réseau
  double n;                // Coefficient permettant de calculer le seuil de détection de
                           // congestions importantes
  int granularite;         // Granularité des calcules donnée en ms
  int running;             // Permet d'arreter et de redémarrer une connexion

/*******************Variable de mesures**********************/

  double FTT;          // Valeur instantanée du FTT
  double FTT_cons;     // Valeur consigne du FTT récupérée dans les acquittements reçus
  double FTT_cons_ems; // Valeur consigne du FTT lors de l'émission
  double FTT_ref;      // Valeur du plus petit FTT mesuré
  double var_FTT;      // Erreur constatée entre le FTT instantané et la consigne
  double RTT;          // Valeur instantané du RTT
  double SRTT;         // Valeur lissée du RTT
  double RTT_ref;      // Vleaur inital du RTT
  double RTO;          // Valeur de déclenchement du timeout
  double NB_TO;        // Nomnbre de timeout successif
  double deb_recp;     // Valeur instantanée du débit de réception
  double deb_ems;      // Valeur du débit d'émission
  double deb_ems_prec; // Débit de la source lors de l'émission du paquet de donnée ayant engendré
                       // l'émission du dernier acquittement reçu
  double retard;       // Valeur du retard (tau * FTT_ref)
  double d_ip_ems;     // Délais inter-paquet à l'émetteur
  double d_ip_recp;    // Délais inter-paquet au récepteur
  double deb_init;     // Débit initial
  int nb_pkts_recv;    // Nombre de paquets de donnée reçus
  int nb_pkts_send;    // Nombre de paquets de données émis
  double last_recv;    // Heure de réception du dernier paquet
  int nb_pkt;          // Variable utiliser pour calculer l'emplacement du débit de réception à
                       // envoyer à l'émetteur
  int max_seq;         // Plus grand numéro de séquence émis

  /*******************Variables de contrôle ************************************/
  int TO_experienced;       // Permet de savoir si au moins un paquet a été recu depuis le dernier TO
  int premiers_pkts_data;   // Variable permettant de savoir si les deux premier paquets de
                            // données ont été reçus
  int init;                 // Variable premttant de savoir la phase d'initialisation est terminée
```

```
  int h_ack;               // Numéro de sequence du dernier paquet recu de manière ordoné.
  int seq_no;              // Numéro de sequence du dernier paquet recu
  int reorg_seq_no[100][2]; // Tableau permettant de réordonner le paquet reçus hors séquence
  primo kiki;              // Objet primo utiliser pour les calcul de débit, ftt, etc.
  PrimoTimer primo_to;     // Timer utilisé pour le timeout
  IPG_Timer primo_ipg;     // Timer utilisé pour le calcul du délai inter paquets

public:
  primoAgent(const char*); // Constructeur
  virtual int command(int argc, const char*const* argv); // Récupère et execute les commandes
                                                         // provenants du script TCL
  virtual void recv(Packet*, Handler*);      // Réception d'un paquet de tout type
  virtual void timeout();                    // Executer en cas d'expiration du timer
  virtual void ipg_expire();
};

#endif //ns_primo
```

C.2.2 Primo.cc

```
// Fabien Chatté
// UTC 8 octobre 2003

#include "primo.h"
#include <stdlib.h>
#include <time.h>

int hdr_primo::offset_;
static class primoHeaderClass : public PacketHeaderClass {
public:
  primoHeaderClass() : PacketHeaderClass("PacketHeader/primo",
                                         sizeof(hdr_primo)) {
    bind_offset(&hdr_primo::offset_);
  }
} class_primohdr;

static class primoClass : public TclClass {
public:
  primoClass() : TclClass("Agent/primo") {}
  TclObject* create(int argc, const char*const* argv) {
    if (argc == 5)
      return(new primoAgent(argv[4]));
    else
      printf("Il faut entrer le nom du noeud lors de la création d'un agent");
  }
}class_primo;




/*****************************************************************************************************/
/***                                                                                               ***/
/***                          IMPLEMENTATION DES METHODES DE L'OBJET PRIMO                         ***/
/***                                                                                               ***/
/*****************************************************************************************************/

//
//Constructeur
//
primo :: primo()
{
  printf("Création de l'objet primo\n");
}

//
// Fonction d'initialisation des valeurs du protocole
//
double primo :: primo_init(double RTT_inst, double FTT_inst, double DEB_init, double T, double DEP_FTT,
```

```
                          double TAU, int GRANULARITE, double BORNE_SUP, double BORNE_INF, double N)
{

  int i=0;
  double FTT_REF, FTT_CONS, DEP_INF;
  RETARD = int(RTT_inst * TAU * 1000);  // Calcul du retard (en ms) utilisé par le correcteur PI
  FTT_REF = FTT_inst;                   // Initialisation du FTT de référence avec le premier FTT mesuré
  FTT_CONS = FTT_REF * DEP_FTT;         // Calcul du FTT consigne en fonction de la valeur de
                                        // réfrence et du dépassement souhaité
  RETARD_G = int(RETARD / GRANULARITE); // Retard tenant compte de la granularité de la simulation
  RETARD_M = RETARD_G + 1;              // Modulo du retard, valeur utilisée pour le tableau
                                        // circulaire contenant les valeur du débit
  VAR_FTT = new double [RETARD_M];      // Allocation dynamique du tableau circulaire VAR_FTT
  EMPLACEMENT_PREC = int(T * 1000 / GRANULARITE);
  EMPLACEMENT_PREC = EMPLACEMENT_PREC % RETARD_M; // Calcul de l'emplacement du rangement initial



  DEP_INF = FTT_CONS * BORNE_INF;  // Calcul du sueil de détection du risque de sous utilisation du réseau
  // Le tableau est initialisé à DEB_INF de manière à ce que pendant
  // le premier RTT le débit d'émission soit celui mesuré initialement
  while (i < RETARD_M)
    {
      VAR_FTT[i]= DEP_INF ;
      i++;
    }

  DEBIT = DEB_init; // Le débit durant le RTT suivant sera celui mesuré initialement
  return(FTT_CONS);
}



//
// Fonction permettant de calculer le débit de la source à l'instant t
// en fonction des dernières mesures effectuées (débit de réception,
// debit d'émission)
//
double primo :: calcul_debit(double T, double deb_recv, double deb_send, double BORNE_SUP,
                             double BORNE_INF, double N, double ALPHA_DEB, double REDUC1,
                             double REDUC2, double GRANULARITE, double FTT_CONS, double AUGMENTE1)
{
  int EMPLACEMENT;
  double var;
  double DEP_SUP, DEP_INF, N_DEP_SUP;
  double new_deb;

  T = T* 1000 / GRANULARITE;
  DEP_SUP = FTT_CONS * BORNE_SUP;  // Calcul du sueil de détection de congestion
  N_DEP_SUP = N * DEP_SUP;         // Calcul du sueil de détection de grosse congestion
  DEP_INF = FTT_CONS * BORNE_INF;  // Calcul du sueil de sous utilisation du réseau
  EMPLACEMENT = int(T * 1000 / GRANULARITE); // Calcul de l'emplacement ou récupérer la
                                             // variation du FTT
  EMPLACEMENT = (EMPLACEMENT - RETARD_G) % RETARD_M; // calcul de l'emplacement relatif
  // (tableau circulaire)
  var = VAR_FTT[EMPLACEMENT]; // Récuparation de la variation du RTT correspondant au calcul
                              // du débit à l'insatant T
  if (var >= N_DEP_SUP)       // Cas de grosse congestion : forte réduction du débit
    {
      new_deb = deb_send + ALPHA_DEB * ( deb_send - (REDUC1 * deb_recv));
      if (new_deb > deb_send)
        new_deb =deb_send;
    }
  else
    {
      if ( var > DEP_SUP )   // Cas d'une congestion légère : petite réduction de débit
        {
          new_deb = deb_send + ALPHA_DEB * ( deb_send - (REDUC2 * deb_recv));
          if (new_deb > deb_send)
```

```
            new_deb = deb_send;
        }
      else
        {
          if (var > DEP_INF) // Cas du réseau bien utilisé : le débit reste inchangé
            new_deb = deb_send;
          else               // Risque de sous utilisation du réseau : augmentation du débit
            {
              new_deb = deb_send + ALPHA_DEB * ( deb_send - (AUGMENTE1 * deb_recv));
              if (new_deb > (1.1 * deb_send))
                    new_deb = deb_send;
              else
                {
                  if (new_deb < deb_send)
                    new_deb = deb_send;
                }
            }
        }
    }
  // Lissage des variations du débit
  DEBIT= 0.9 * DEBIT + 0.1 * new_deb;
  return(DEBIT);
}



//
// Méthode calcul l'écart entre le FTT consigne et le FTT mesuré. La
// valeur obtenu est stockée dans un tableau pour être utilisé
// ultérieurement dans le calcul du débit.
//

double primo :: rangement_var_FTT(double FTT_inst, double T, double FTT_CONS, double GRANULARITE)
{
  double var;
  int EMPLACEMENT;
  int i;

  var = FTT_inst - FTT_CONS;                // Calcul de l'écart entre la consigne et la valeur mesurée
  EMPLACEMENT = int(T * 1000 / GRANULARITE); // Calcul de l'emplacement où ranger la valeur calculée
  EMPLACEMENT = EMPLACEMENT%RETARD_M;
  VAR_FTT[EMPLACEMENT] = var;               // Rangement de la valeur dans le tableu
  i = EMPLACEMENT_PREC + 1;                 // Initialisation de la variable de boucle i
  // Cette boucle permet de remplir les espaces laissés vide entre deux réceptions d'acquittement
  // Le remplissage se fait avec l'avant dernière valeur de l'écart mesuré.

  i = i%RETARD_M;
  while  (i != EMPLACEMENT)
    {
      VAR_FTT[i]=VAR_FTT[i-1];
      i++;
      i = i%RETARD_M;
    }

  EMPLACEMENT_PREC = EMPLACEMENT; // mise à jour du dernier emplacement
                                  // remplis avec une nouvelle valeur.
  return(var);
}



//
// Méthodes permettant d'acceder à des variables privées de l'objet
//

double primo :: access_retard()
{
  return(RETARD) ;
```

```
}

double primo :: access_debit()
{
  return(DEBIT);
}

void primo :: set_DEBIT(double deb)
{
  DEBIT =deb;
}


/*****************************************************************************************************/
/***                                                                                               ***/
/***                      IMPLEMENTATION DES METHODES DE L'OBJET PRIMOAGENT                        ***/
/***                                                                                               ***/
/*****************************************************************************************************/


primoAgent::primoAgent(const char* nom) : Agent(PT_PRIMO), primo_to(this),primo_ipg(this)
{

  if( strlen(nom) <= TAILLE_NOM)
    (void)strcpy(monNom, nom);
  else
    {
      (void)strncpy(monNom, nom, TAILLE_NOM-1);
      (void)strcat(monNom, ".");
    }

  /******************** Partage des variables avec les scripts TCL ******************/

  bind("packetSize_", &size_);
  bind("alpha_deb_", &alpha_deb);
  bind("tau_", &tau);
  bind ("dep_FTT_", &dep_FTT);
  bind ("dep_deb_", &dep_deb);
  bind ("borne_sup_", &borne_sup);
  bind("borne_inf_", &borne_inf);
  bind("reduc1_", &reduc1);
  bind("reduc2_", &reduc2);
  bind("augmente1_", &augmente1);
  bind("n_", &n);
  bind("granularite_", &granularite);
  bind("FTT_", &FTT);
  bind("FTT_cons_", &FTT_cons);
  bind("FTT_ref_", &FTT_ref);
  bind("var_FTT_", &var_FTT);
  bind("RTT_", &RTT);
  bind("SRTT_", &SRTT);
  bind("RTO_",&RTO);
  bind("deb_recp_", &deb_recp);
  bind("deb_ems_", &deb_ems);
  bind("deb_ems_prec_", &deb_ems_prec);
  bind("retard_", &retard);
  bind("d_ip_ems_", &d_ip_ems);
  bind("d_ip_recp_", &d_ip_recp);
  bind("deb_init_", &deb_init);
  bind("nb_pkts_recv_", &nb_pkts_recv);
  bind("nb_pkts_send_", &nb_pkts_send);
  bind("max_seq_", &max_seq);
  bind("nb_pkt_",&nb_pkt);

  printf("\nCréation de l'agent primo %s\n",nom);
  srand(time(NULL)); // initialisation du générateur de nombre alléatroire
  max_seq = 0;       // création du premier numéro de séquence
  h_ack = 0;         // initialisation du plus grand numéro de réquence reçu de manière ordonnée
  fflush(stdout);
```

```
}

// Fonction récupérant et executant les commandes lancée à partir du script TCL
int primoAgent::command(int argc, const char*const* argv)
{
  int i;
  double t, tempo;

  if(argc == 2)
    {
      if (strcmp(argv[1], "start") == 0)  // Démarrage de la connexion
        {
          running = 1; // Premt par la suite de savoir si l'on est autoriser à 'émettre des paquets
          // Initialisation des variables de contrôle
          premiers_pkts_data = 0;
          init = 0;
          tempo = rand()%100;
          tempo = tempo / 100;
          primo_ipg.sched(0);
          return (TCL_OK);
        }
      if (strcmp(argv[1], "stop") == 0) // Arret des émissions
        {
          running = 0; // Empeche toute émission
          deb_ems=0;
          primo_to.cancel(); // Arret du timer
          primo_ipg.cancel();
          return (TCL_OK);
        }
    }
  return (Agent::command(argc, argv));
}


void primoAgent::recv(Packet* pkt, Handler*)
{
  static double stime;    // Heure d'émission du paquet reçu
  double sdeb;            // Débit d'émission du paquet reçu
  static double deb_recv; // Débit contenu dans l'acquittement reçu
  double ftt_cons;        // Valeur du FTT consigne contenue dans l'entête ddu paquet de donnée reçu
  static double ftt_init; // Valeur du FTT initial mesuré au récepteur
  static double d_ip_recv;// Variable servant à stocker le délai inter paquet
  double deb_recp_temp;   // Variable sevant à la mesure de la bande passante disponible
  int i;                  // Variable de boucle
  int h_ack_next;         // Priochain numéro de séquence attendu au récepteur
  static int h_max=0;     // Nombre de paquets à réordonner

  // Acces au  header IP :
  double t_recv; //  heure de réception du paquet ou de l'acquittment
  hdr_ip* hdrip = hdr_ip::access(pkt);
  // Acces au header primo headert:
  hdr_primo* hdr = hdr_primo::access(pkt);
  // Vérification si ret <= 0 (pour émettre ou non un ack)
  if (hdr->ret <= 0)
    {
      /********************* Réception d'un paquet de donnée *********************/
      /* Calucul du débit de réception, du FTT insatantané, et émission d'un     */
      /* acquittement                                                            */
      /***************************************************************************/

      // heure de réception du paquet
      t_recv = Scheduler::instance().clock();
      // Récupération du débit d'émission des données
      sdeb = hdr->deb_ems;
      // Récupération du FTT consigne
      ftt_cons =hdr->FTT_cons;
      // Récupération de l'heure d'emission du paquet pour pouvoir la remettre dans l'ack
      stime = hdr->heure_ems;
      // récupération du numéro de séquence
```

```
seq_no = hdr -> seqno;
// incrémente le nombre de paquets reçus
nb_pkts_recv ++;
// Création d'un nouveau paquet
Packet* pktret = allocpkt();
hdr_primo* hdrret = hdr_primo::access(pktret);
// On met ret à 1 pour spécifier qu'il s'agit d'un ack
hdrret->ret = 1;
// Recopie de l'heure d'émission du premier paquet de données
hdrret->heure_ems_don = stime;
// Recopie du numéro de séquence dans l'acquittement.
hdrret->acked_seqno = seq_no;

// Test s'il s'agit d'un paquet de donnée
if (hdr->ret ==0)
  {
    if  (premiers_pkts_data == 1)
      {
        // Si ce n'est pas le premier paquet reçu après l'initialisation de la connexion
        // On calcul le temp écoulé entre la réception de ce paquet et celle du
        // dernier paquet reçu.
        d_ip_recp = t_recv - last_recv;
        // Calcul du débit instantané.
        hdrret->deb_recp = (size_ * 8)  / d_ip_recp; // Passage d'octets en bits
        // Mise à jour l'heure de la dernière réception
        last_recv = t_recv;
        // Calcule du FTT instantané
        hdrret->FTT_inst = t_recv - stime;
        // Recopie du débit d'émission du paquet de donnée
        hdrret->deb_ems_don = sdeb;
        // Recopie du FTT consigne
        hdrret->FTT_cons = ftt_cons;
        // Envoi de l'ack
        send(pktret, 0);
      }
    else
      {
        // Si c'est le premier paquet reçu après l'initialisation
        // On utilise le dernier délai inter paquets calculé
        d_ip_recp = d_ip_recv;
        hdrret->deb_recp = (size_ * 8)  / d_ip_recp; // Passage d'octets en bits
        last_recv = t_recv;               // Mise à jour l'heure de la dernière réception
        hdrret->FTT_inst = t_recv - stime; // Calcule du FTT instantané
        hdrret->deb_ems_don = sdeb;        // Recopie du débit d'émission du paquet de donnée
        hdrret->FTT_cons = ftt_cons;       // Recopie du FTT consigne
        premiers_pkts_data ++;             // Incrémente le nombre de paquets reçus
        send(pktret, 0);                   // Envoi de l'ack
      }
  }
else
  {
    if (hdr->ret == -1)
      {
        /******************* Réception du premier paquet d'initialisation*********/
        /* Calcul du débit de réception, du FTT insatantané, et émission d'un     */
        /* acquittement                                                           */
        /**************************************************************************/

        // Si c'est le premier paquet de donnée reçu, on récupère l'heure de sa réception
        // dans la variable last_recv
        last_recv = t_recv;
        // Calcul du premier FTT
        ftt_init = t_recv - stime;
        // Recopie du débit d'émission du paquet de donnée
      }
    else
      {
        if (hdr->ret < -3)
          {
```

```
                hdrret->FTT_inst = ftt_init;
                d_ip_recp = t_recv - last_recv;
                hdrret->deb_recp = 3*(size_ * 8)  / (d_ip_recp); // Passage d'octets en bits
                // Recopie du FTT consigne
                hdrret->FTT_cons = ftt_cons;
                // Envoi de l'ack
                send(pktret, 0) ;
              }
            }
        }
      // Destruction du paquet reçu
      Packet::free(pkt);
    }
              /************************************/
              /* Réception d'un paquet de contrôle */
              /************************************/
  else
    {
      t_recv = Scheduler::instance().clock();
      if (hdr->ret == 1)
        {
             /************************ Réception d'un acquittement *************************/
             /* Récupération des informations contenues dans l'acquittement, calcul et       */
             /* rangement de la variation du FTT                                             */
             /*******************************************************************************/
          // Test s'il s'agit du premier ack reçu
          if (init == 1)
            {
              // Si c'est le premier ack, on calcul le rtt qui servira de référence
              init = 2;
              // Calcul du RTT instantané, et du timer du Time Out
              RTT = t_recv - hdr->heure_ems_don;
              SRTT = RTT;
              RTO = 2*SRTT;
              // Récupération du RTT de Référence servant à calculer les période de mesure
              // de la bande passante.
              RTT_ref= RTT;
              // Calcul du FTT de référence, et du débit initilalement mesuré
              FTT_ref = hdr->FTT_inst;
              deb_init = hdr->deb_recp;
              // Initialisation de l'objet primo, qui effectue le calcul du débit
              FTT_cons = kiki.primo_init(RTT, FTT_ref, deb_init,t_recv, dep_FTT, tau, granularite,
                                         borne_sup, borne_inf, n);
              FTT_cons_ems = FTT_cons;  // Mise à jour du ftt consigne
              deb_recp = hdr->deb_recp; // Mise à jour du débit de réception mesuré
              deb_ems_prec = deb_recp;  // Pour le premier ack  le débit d'émission précédent est
                                        // initialisé avec la valeur du débit de réception.
              TO_experienced =0;        // Le nombre d'expiration consécutive du TO est réinitialisé
              primo_to.resched(RTO);    // Le timer est réarmé avec la dernière valeur calculé du RTO
              primo_ipg.resched(0);     // Le délai inter-paquets est nul pour la réception
                                        // du premier ack.
            }
          else
            { // Ce n'est pas le premier ack reçu
              // Calcul du RTT instantané, du RTT lissé (SRTT) ainsi que de Time out (RTO)
              RTT = t_recv - hdr->heure_ems_don; // Calcul du RTT instantané
              SRTT = 0.8 * SRTT + 0.2* RTT;      // Calcul du RTT lissé
              RTO =d_ip_ems + FTT_cons;          // Calcul du RTO
              if (RTO < 1)                       // Le RTO ne peut être inférieur à 1s
                {
                  RTO=1;
                }
              FTT = hdr->FTT_inst;  // Récupération du FTT instantané
              if (FTT < FTT_ref)    // Eventuelle mise à jour du FTT de référence
                {
                  FTT_ref = FTT;
                  FTT_cons_ems = FTT_ref * dep_FTT;
                }
              // Récupération du débit de réception instantané
```

```
              deb_recp = hdr->deb_recp;
              // Récupération de la consigne du FTT et du débit d'émission qu'avait la source
              // lors de l'émission du paquet ayant engendré cet ack
              FTT_cons = hdr->FTT_cons;
              // Calcul et rangement de la variation du FTT
              var_FTT = kiki.rangement_var_FTT(FTT, t_recv, FTT_cons, granularite);
              deb_ems_prec = hdr->deb_ems_don;
              TO_experienced =0;    // Réinitialisation du nombre d'expiration consécutive du TO
              primo_to.resched(RTO);// Réarmement du chien de garde est
            }
    }
    // Destruction du paquet
    Packet::free(pkt);
  }
}

void primoAgent :: timeout()
{
  double t;
  double tempo;
  printf("%s TO\n\n",monNom);
  fflush(stdout);
  NB_TO++;
  if (init >= 2) // Test si la phase d'initialisation est terminée
   {
      if(NB_TO > 4)
        { // Arret de la connexion après 5 Time Out consecutifs
          running = 0; // Empeche toute émission
          deb_ems=0;
          primo_to.cancel(); // Arret du timer de retransmission
          primo_ipg.cancel();// Arret du timer d'émission
          printf("Fin de la connexion : 5 timeout\n");
        }
      else
        {
          if (max_seq != seq_no) // Test s'il y a encore des données en transit sur le réseau
            {
              t = Scheduler::instance().clock();
              if (TO_experienced <= 2)   // Le débit ne pourra pas être réduit plus de trois fois de suite
                deb_ems = deb_ems * 0.5; // Mise à jour du débit en le divisant par deux
              if (deb_ems < 5000)        // Le débit d'émissiion ne peut être inférieur à 5kbits/s
                deb_ems =5000;
              kiki.set_DEBIT(deb_ems);
              RTO = 2*RTO;
              RTO = RTO + t;
              if (RTO > 5) // La durée du Time Out ne peut pas exceder 5 s
                RTO = 5;
              primo_to.resched(RTO); // Réarmement du timer
              printf("%s resched après un TO : init 2 à t = %f RTO = %f\n",monNom,t,RTO);
              TO_experienced ++; // incrémentation du nombre d'expiration consécutives
            }
        }
    }
  else
    { // Si la phase d'initialisation n'estpas terminée
      if(NB_TO > 4)
        { // Arret de la connexion après 5 Time Out consecutifs
          running = 0; // Empeche toute émission
          deb_ems=0;
          printf("Fin de la connexion : 5 timeout\n");
        }
      else
        {
          t = Scheduler::instance().clock();
          tempo = rand()%100; // Valeur tirée au sort permettant d'éviter une
          // synchronisation de différentes sources durant la phase d'initilisation
          tempo = tempo / 100;
          RTO = RTO + tempo;
          if (RTO > 5)// La durée du timeout ne peut pas exceder 5s
```

```
            RTO = 5;
          printf("%s resched après un TO : init 0 à t = %f avec un RTO =%f\n",monNom,t,RTO);
          init = 0;
          primo_ipg.resched(0); // Emission imédiate d'une nouvelle rafale
        }
    }
}

void primoAgent :: ipg_expire()
{
  double t;
  t = Scheduler::instance().clock();

  if(init==0)
    {
      /****************** Phase d'initialisation de la connexion ******************/
      /** émission de paquets ayant un délai inter paquet nul (ie. débit infinit)  **/
      /** puis attente de la réception des premiers acquittements                  **/
      /************************************************************************/

      max_seq = max_seq + size_; // Mise à jour du numéro de séquence avant l'émission du paquet
      Packet* pkt = allocpkt(); // Creation d'un nouveau paquet
      hdr_primo* hdr = hdr_primo::access(pkt); // Acces au header du paquet primo pkt
      // Ret est mis à -1 pour que le récepteur sache qu'il s'agit du premier
      // paquet d'initialisation
      hdr->ret = -1;
      // L'heure d'émission du paquet est inclue afin de calculer le RTT lors du retour de l'ack
      hdr->heure_ems = Scheduler::instance().clock();
      hdr->seqno = max_seq; //Le numéro de séquence du paquet est indiqué dans l'entête
      send(pkt, 0);         // Envoie du paquet
      nb_pkts_send ++;       // incrémentation du nombre de paquets envoyés

      Packet* pkt2 = allocpkt(); // Creation d'un nouveau paquet
      hdr_primo* hdr2 = hdr_primo::access(pkt2);
      // ret est mis à -2 pour que le récepteur sache qu'il s'agit du 2eme
      // paquet d'initialisation
      hdr2->ret = -2;
      hdr2->heure_ems = Scheduler::instance().clock();
      hdr2->deb_ems = 0;
      max_seq =max_seq + size_;
      hdr2->seqno = max_seq;
      send(pkt2, 0);
      nb_pkts_send ++;

      Packet* pkt3 = allocpkt();
      hdr_primo* hdr3 = hdr_primo::access(pkt3);
      // ret est mis à -3 pour que le récepteur sache qu'il s'agit du 3eme
      // paquet d'initialisation
      hdr3->ret = -3;
      hdr3->heure_ems = Scheduler::instance().clock();
      hdr3->deb_ems = 0;
      max_seq =max_seq + size_;
      hdr3->seqno = max_seq;
      send(pkt3, 0);
      nb_pkts_send ++;

      Packet* pkt4 = allocpkt();
      hdr_primo* hdr4 = hdr_primo::access(pkt4);
      // ret est mis à -4 pour que le récepteur sache qu'il s'agit du 4eme
      // paquet d'initialisation
      hdr4->ret = -4;
      hdr4->heure_ems = Scheduler::instance().clock();
      hdr4->deb_ems = 0;
      max_seq =max_seq + size_;
      hdr4->seqno = max_seq;
      primo_to.sched(RTO);
      send(pkt4, 0);
      nb_pkts_send ++;
      init =1; // Fin de l'envoi de la rafale d'émission
```

```
    }

  if (init == 2)
    {
      /********************** Régime permanant de la connexion *******************/
      /** Dans cette phase la connexion émet des paquets avec un itnrevale de    **/
      /** temps de "d_ip_ems" qui est caculé grâce aux débit de la connexion     **/
      /*****************************************************************************/

      Packet* pkt = allocpkt();
      hdr_primo* hdr = hdr_primo::access(pkt); // Acces au header du paquet primo pkt
      max_seq = max_seq + size_; // Mise à jour du numéro de séquence avant
                                 //l'émission du paquet
      deb_ems = kiki.calcul_debit(t, deb_recp, deb_ems_prec, borne_sup, borne_inf, n, alpha_deb,
                                  reduc1, reduc2, granularite, FTT_cons, augmente1);
      // ret est mis à 0 pour que le récepteur envoie un ack (ie c'est un pkt de données)
      hdr->ret = 0;
      // L'heure d'émission du paquet est indiqué afin de pouvoir
      // calculer le rtt lors du retour de l'ack
      hdr->heure_ems = Scheduler::instance().clock();
      hdr->seqno = max_seq; // Le numéro de séquence du paquet est indiqué dans l'entête
      // Le débit d'émission du paquet est indiqué afin de calculer le
      // futur débit d'émission au retour de l'ack
      hdr->deb_ems = deb_ems;
      hdr->FTT_cons = FTT_cons_ems;     // La consigne du FTT est spécifiée dans l'entête du paquet
      send(pkt, 0);                     // Envoie du paquet
      nb_pkts_send ++;                  // Incrémentation du nombre de paquet émis
      d_ip_ems =  (size_ * 8) /deb_ems; // multiplication par huit pour passer en bits
      primo_ipg.resched(d_ip_ems);      // Réarmement du chien de garde, déclanchant la
                                        // prochaine émission, avec le délai d_ip_ems
    }
}

void PrimoTimer :: expire(Event *e)
{
  a_->timeout();
}

void IPG_Timer :: expire(Event *e)
{
  a_->ipg_expire();
}
```

Annexe D

Tableaux de résultats de simulations

D.1 Influence de la bande passante

D.1.1 Sans pertes d'acquittements

Primo.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	79,83	79,96	99,81	99,88	100	97,06	34,58
10	97,75	99,43	99,81	99,89	100	95,98	37,5
15	93,27	93,33	99,88	99,9	100	94,86	37,93
20	97,68	99,65	99,93	99,92	100	93,78	39,92
30	97,41	99,71	99,98	99,94	100	91,6	40,45
50	96,12	97,67	98,3	99,96	100	86,68	48,35
80	95,99	97,86	98,1	99,97	100	79,79	46,45
moyenne	94,01	95,37	99,4	99,92	100	91,39	40,74

Reno.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	11,78	16,76	95,73	96,15	88,1	28,1	76,01
10	26,12	31,46	93	96,15	91,49	30,81	73,17
15	41,52	46,18	94,49	96,15	93,13	32,84	70,18
20	53,75	58,19	96,66	96,16	94,03	34,7	67,92
30	69,75	74,91	93,77	96,15	95,31	39,14	61,47
50	74,26	78,83	93,6	93,47	96,05	55,48	32,15
80	59,53	63,33	84,37	82,59	98,01	74,51	11,92
Moyenne	48,1	52,81	93,09	93,83	93,73	42,23	56,12

Sack.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	12,99	16,81	96,17	96,15	88,01	25,86	77,74
10	27,45	31,35	93,04	96,15	91,51	28,65	74,9
15	42,72	46,79	96,02	96,15	93,11	31,13	71,62
20	55,29	58,04	96,16	96,15	94,04	32,96	69,71
30	70,74	73,45	93,75	96,15	95,3	37,34	63,31
50	83,83	89,58	96	96,14	95,53	47,78	48,22
80	82,83	85,88	94,52	93,63	97,35	56,49	32,7
Moyenne	53,69	57,41	95,09	95,79	93,55	37,17	62,6

TFRC.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	78,11	79,43	94,5	100	90,19	14,44	87,28
10	93,47	94,91	98,81	100	94,11	11,28	93,07
15	85,58	86,43	84,98	100	94,21	12,14	91,86
20	94,21	96,03	99,29	100	94,77	13,01	91,73
30	90,53	91,69	94,07	100	95,89	14,33	91,05
50	91,19	92,34	96,63	100	97,18	16,65	89,42
80	88,05	89,07	81	100	98,8	23,34	85,75
Moyenne	88,73	89,99	92,75	100	95,02	15,03	90,02

Vegas.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	39,94	80	99,98	100	100	84,2	91,61
10	70,95	71,11	76,94	100	100	84,17	91,61
15	91,5	91,69	75,03	100	100	85,04	88,55
20	90,81	91,1	99,97	100	100	88,33	85,16
30	90,49	90,77	99,99	100	100	82,59	92,56
50	90,32	91,16	100	100	100	83,47	98,3
80	99,61	100	100	100	100	93,36	99,15
Moyenne	81,95	87,98	93,13	100	100	85,88	92,42

D.1.2 Avec pertes d'acquittements

Primo.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	79,71	79,86	99,61	99,35	100	96,34	23,11
10	97,33	98,41	99,37	99,35	100	94,48	22,22
15	93,29	93,32	99,47	99,48	100	93,86	25,57
20	97,35	98,77	99,43	99,48	100	92,62	25,9
30	97,43	99,08	99,63	99,6	100	90,5	27,92
50	96,46	97,69	98,35	99,86	100	85,21	40,97
80	96,34	97,97	98,1	99,86	100	77,51	41,53
Moyenne	93,99	95,01	99,14	99,57	100	90,07	29,6

Reno.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	13,68	3,58	63,69	55,98	87,2	79,82	19,99
10	3,41	6,99	62,08	53,25	87,77	77,25	11,97
15	8,05	9,89	63,44	67,22	87,98	68,72	2,33
20	11,53	15,4	85,63	85,31	94,85	60,19	26,15
30	22,48	22,59	82,9	85,33	95,7	61,3	23,37
50	30,1	31,77	96,29	83,56	96,21	66,11	9,12
80	53,49	55,75	73,47	80,47	98,27	77,79	16,93
Moyenne	20,39	20,85	75,36	73,02	92,57	70,17	15,69

Sack.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	14,76	32,46	79,39	78,66	85,89	62,92	21,53
10	24,61	32,34	77,41	84,5	86,81	60,28	25,48
15	32,58	35,88	96,39	90,24	85,58	56,27	38,89
20	62,3	72,03	90,39	93,56	91,92	42,15	59,37
30	69,29	74,89	93,92	94,33	93,88	43,1	57,78
50	68,26	72,01	95,48	93,75	95,73	51,46	41,02
80	81,59	83,42	92,91	91,35	97,68	63,84	17,97
Moyenne	50,48	57,58	89,41	89,48	91,07	54,29	37,43

TFRC.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	76,54	81,97	85,48	85,09	69,76	72,94	22,24
10	73,75	79,04	82,42	82,68	82,75	71,12	16,12
15	90,06	90,72	95,75	96,33	94,83	18,08	88,53
20	87,93	88,35	95,55	97,36	90,75	18,76	86,75
30	90,62	91,6	96,78	98,14	95,03	16,77	89,24
50	90,23	91,35	92,14	98,85	97,24	19,82	87,29
80	87,34	88,37	98	99,31	98,61	21,58	86,67
Moyenne	85,21	87,34	92,3	93,97	89,85	34,15	68,12

Vegas.

Rapport de bande passante (%)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	28	24,09	17,74	17,65	100	99,53	91,8
10	3,96	4,31	24,66	24,59	100	99,48	90,4
15	5,69	0,81	21,47	22,38	100	99,06	85,06
20	82,21	82,28	72,82	97,11	100	91,71	72,48
30	91,21	91,49	94,16	98,08	100	78,88	89,29
50	96,09	96,56	96,02	98,85	100	91,02	74,62
80	97,93	98,14	99,28	99,28	100	90,19	78,29
Moyenne	57,87	56,81	60,88	65,42	100	92,84	83,14

D.2 Influence de la taille de d'attente (*Drop Tail*)

D.2.1 Sans pertes d'acquittements

Primo.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	85,23	91,85	98,8	99,21	85,14	70,19	34,72
10	93,27	93,33	99,88	99,9	100	84,57	37,93
15	93,27	93,33	99,88	99,9	100	89,71	37,93
20	93,27	93,33	99,88	99,9	100	92,28	37,93
30	93,27	93,33	99,88	99,9	100	94,86	37,93
40	93,27	93,33	99,88	99,9	100	96,14	37,93
50	93,27	93,33	99,88	99,9	100	96,91	37,93
Moyenne	92,12	93,12	99,73	99,8	97,88	89,24	37,47

Reno.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	76,51	85,4	90,94	91,03	77,96	64,46	21,03
10	70,98	79,4	94,72	94,74	84,81	48,46	41,64
15	65,21	70,24	96,15	96,15	88,45	40,01	56,96
20	56,66	61,89	96,15	96,15	90,56	36,25	66,21
30	41,52	46,18	94,49	96,15	93,14	32,84	70,18
40	33,36	37,6	96,07	96,15	93,98	30,47	73,63
50	27,04	31,86	96,03	96,15	94,26	29,88	74,24
Moyenne	53,04	58,94	95,4	95,22	89,02	40,34	57,7

Sack.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	77,62	83,07	88,2	88,16	78,37	61,65	26,03
10	74,25	79,94	94,42	94,4	84,95	48,51	41,15
15	66,03	69,14	90,35	96,15	88,52	38,83	59,96
20	57,18	61,86	95,51	96,15	90,64	33,99	67,71
30	42,72	46,79	96,02	96,15	93,11	31,13	71,62
40	33,87	37,23	96,15	96,15	93,99	28,43	75,79
50	28,6	30,88	96,02	96,15	94,26	28,11	76,3
Moyenne	54,33	58,41	93,81	94,76	89,12	38,66	59,79

TFRC.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	89,31	90,8	96,53	99,99	96,19	40,18	83,42
10	86,32	87,62	93,21	100	81,99	21,86	90,48
15	91,01	92,31	98,79	100	86,81	17,72	90,73
20	90,86	91,66	99,02	100	90,99	15,18	91,39
30	85,58	86,43	84,98	100	94,21	12,14	91,86
40	89,42	91,26	94,92	100	94,98	14,92	88,24
50	89,75	92,1	99,02	100	96,7	12,75	89,6
Moyenne	88,89	90,31	95,21	100	91,7	19,25	89,39

Vegas.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	76,58	77,02	2,12	100	96,89	65,01	68,85
10	91,5	91,69	75,03	100	100	55,13	88,55
15	91,5	91,69	75,03	100	100	70,09	88,55
20	91,5	91,69	75,03	100	100	77,56	88,55
30	91,5	91,69	75,03	100	100	85,04	88,55
40	91,5	91,69	75,03	100	100	88,78	88,55
50	91,5	91,69	75,03	100	100	91,03	88,55
Moyenne	89,37	89,59	64,61	100	99,56	76,09	85,74

D.2.2 Avec pertes d'acquittements

Primo.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	84,66	91,5	99,81	99,39	75,64	69,06	22,43
10	93,29	93,33	99,38	99,5	99,47	81,6	25,64
15	93,29	93,32	99,47	99,48	100	87,71	25,57
20	93,29	93,32	99,47	99,48	100	90,78	25,57
30	93,29	93,32	99,47	99,48	100	93,86	25,57
40	93,29	93,32	99,47	99,48	100	95,39	25,57
50	93,29	93,32	99,47	99,48	100	96,31	25,57
Moyenne	92,06	93,06	99,51	99,47	96,44	87,82	25,13

Reno.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	0,25	0,64	44,57	58,45	88,4	87,25	38,72
10	8,85	9,84	66,43	60,12	87,62	83	19,06
15	2,67	0,54	49,6	62,18	87,66	77,05	12,69
20	6,49	6,02	66,34	62,47	85,78	70,78	2,43
30	8,05	9,89	63,44	67,22	87,98	68,72	2,33
40	4,15	8,84	61,97	75,08	91,11	68,88	11,08
50	1,99	5	66,03	78,82	92,42	66,02	20,97
Moyenne	4,64	5,82	59,77	66,33	88,71	74,53	15,33

Sack.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	26,93	26,52	79,91	74,48	84,77	79,44	7,08
10	20,63	22,5	67,38	79,72	85,4	71,89	11,44
15	28,14	30,34	81,33	84,1	83,67	65,35	22
20	14,24	12,37	92,19	78,64	85,97	67,19	12,23
30	32,58	35,88	96,39	90,24	85,58	56,27	38,89
40	25,59	27,07	77,16	92,12	89,02	48,71	49,34
50	33,4	38,69	85,15	93,46	91	43,1	59,79
Moyenne	25,93	27,63	82,79	84,68	86,49	61,71	28,68

TFRC.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	81,87	83,63	69,42	95,11	79	63,24	50,56
10	90,02	91,24	94,63	96,3	82,45	37,7	73,81
15	89,5	90,84	97,38	96,31	86,3	33,75	74,51
20	88,46	89,38	91,84	96,31	91,58	23,86	84,63
30	90,06	90,72	95,75	96,33	94,83	18,08	88,53
40	87,88	88,79	99,21	96,44	93,07	15,98	88,82
50	83,68	86,28	86,77	96,23	95,74	15,17	88,55
Moyenne	87,35	88,7	90,72	96,15	89	29,68	78,49

Vegas.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	14,92	13,16	23,61	23,91	98,87	97,65	90,1
10	25,32	21,35	21,58	22,17	99,91	98,53	91,52
15	5,69	0,81	21,47	22,38	100	98,11	85,06
20	5,69	0,81	21,47	22,38	100	98,59	85,06
30	5,69	0,81	21,47	22,38	100	99,06	85,06
40	5,69	0,81	21,47	22,38	100	99,29	85,06
50	5,69	0,81	21,47	22,38	100	99,43	85,06
Moyenne	9,81	5,51	21,79	22,57	99,83	98,67	86,7

D.3 Influence de la taille de d'attente (RED)

D.3.1 Sans pertes d'acquittements

Primo.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	11,97	4,68	11,43	99,43	9,78	17,64	81,4
10	93,41	93,45	98,92	99,97	100	83,76	51,18
15	93,41	93,45	98,92	99,97	100	89,17	51,18
20	93,41	93,45	98,92	99,97	100	91,88	51,18
30	93,41	93,45	98,92	99,97	100	94,59	51,18
40	93,41	93,45	98,92	99,97	100	95,94	51,18
50	93,41	93,45	98,92	99,97	100	96,75	51,18
Moyenne	81,78	80,77	86,42	99,89	87,11	81,39	55,49

Reno.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	76,26	87,27	91,53	91,56	80,15	52,48	21,31
10	71,12	77,94	90,93	96,12	85,52	42,05	50,47
15	55,02	58,87	99,09	96,15	87,95	39,45	72,04
20	53,61	58,64	96,75	96,14	89,9	40	74,61
30	41,43	47,6	97,58	96,15	91,83	42,99	78,36
40	34,27	40,16	89,5	96,15	92,79	44,81	80,62
50	31,48	34,46	91,52	96,15	93,08	46,06	81,77
Moyenne	51,88	57,85	93,84	95,49	88,74	43,98	65,6

Sack.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	75,15	83,02	88,4	88,38	80,46	50,14	24,97
10	70,55	74,93	87,23	96,09	85,64	39,35	48,97
15	61,18	63,44	97,66	96,15	88,3	38,76	73,68
20	52,72	55,17	99,91	96,15	89,93	38,55	76,48
30	46,06	48,64	92,03	96,15	91,96	41,98	80,79
40	36,31	37,81	96,26	96,15	92,72	43,74	82,98
50	34,01	35,46	98,69	96,15	93,05	44,82	83,37
Moyenne	53,71	56,92	94,17	95,03	88,87	42,48	67,32

TFRC.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	85,15	86,51	98,51	99,99	74,35	31,34	74,77
10	84,63	85,86	95,1	100	80,23	33,65	71,07
15	86,09	86,88	99,76	100	85,25	38,55	73,3
20	85,83	86,2	97,87	100	88,74	39,52	80,44
30	86,02	87,79	95,09	100	90,82	42,66	84,07
40	84,9	87,51	99,97	100	92,4	45,4	76,87
50	86,21	88,79	96,51	100	94,1	46,38	80,65
Moyenne	85,55	87,08	97,54	100	86,56	39,64	77,31

Vegas.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	91,24	91,43	71,43	100	99,84	47,66	81,66
10	91,5	91,69	75,03	100	100	55,13	88,55
15	91,5	91,69	75,03	100	100	70,09	88,55
20	91,5	91,69	75,03	100	100	77,56	88,55
30	91,5	91,69	75,03	100	100	85,04	88,55
40	91,5	91,69	75,03	100	100	88,78	88,55
50	91,5	91,69	75,03	100	100	91,03	88,55
Moyenne	91,46	91,65	74,52	100	99,98	73,61	87,57

D.3.2 Avec pertes d'acquittements

Primo.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	11,31	14,21	10,18	99,47	27,84	30,39	71,89
10	93,52	93,59	98,87	99,65	99,85	81,09	47,35
15	93,52	93,55	98,87	99,61	100	87,11	46,23
20	93,52	93,55	98,87	99,61	100	90,34	46,23
30	93,52	93,55	98,87	99,61	100	93,56	46,23
40	93,52	93,55	98,87	99,61	100	95,17	46,23
50	93,52	93,55	98,87	99,61	100	96,13	46,23
Moyenne	81,78	82,22	86,2	99,6	89,67	81,97	50,05

Reno.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	0,76	3,38	47,06	56,52	88,37	82,61	29
10	1,82	3,49	57	60,36	88,96	81	20,05
15	1,64	0,13	54,76	57,38	89,15	79,08	20,01
20	23,77	29,43	69,95	69,7	90,21	71,25	4,49
30	9,6	12,47	79,42	76,64	91,89	70,72	0,1
40	12,63	16,2	94,29	87,48	93,45	58,8	28,57
50	27,06	30,07	92,36	88,46	92,94	51,86	43,31
Moyenne	11,04	13,6	70,69	70,94	90,71	70,76	20,79

Sack.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	41,32	41,94	71,26	76,65	81,55	70,52	4,7
10	41,65	43,76	90,14	83,98	82,46	63,45	20,11
15	27,63	30,21	66,79	87,26	82,18	57,87	30
20	45,82	52,23	89,29	90,41	83,11	51,79	43,98
30	49,54	57,21	89,77	92,85	87,99	46,38	57,82
40	47,79	52,23	98,56	93,41	90,1	45,23	62,04
50	40,05	44,19	99,73	93,46	91,89	45,95	63,42
Moyenne	41,97	45,97	86,51	88,29	85,61	54,46	40,3

TFRC.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	89,28	91,76	97,03	95,43	78,19	49,35	50,85
10	86,62	87,65	95,73	96,1	79,11	40,56	60,2
15	84,68	85,28	93,45	96,18	82,93	42,76	59,79
20	85,4	85,89	94,2	96,19	83,81	41,15	64,66
30	84,44	85,07	94,67	96,25	87,24	42,32	68,3
40	88,21	88,81	99,84	96,44	87,96	41,86	68,97
50	84,69	86,36	98,34	96,65	89,95	43,62	71,42
Moyenne	86,19	87,26	96,18	96,18	84,17	43,09	63,46

Vegas.

Taille de la file d'attente (paquets)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	59,05	66,29	65,83	68,99	95,59	93,54	72,2
10	85,49	86,11	83,7	87,17	99,91	80,12	6,11
15	85,22	86,78	87,78	87,75	100	85,72	4,03
20	85,22	86,78	87,78	87,75	100	89,29	4,03
30	85,22	86,78	87,78	87,75	100	92,86	4,03
40	85,22	86,78	87,78	87,75	100	94,65	4,03
50	85,22	86,78	87,78	87,75	100	95,72	4,03
Moyenne	81,52	83,76	84,06	84,99	99,36	90,27	14,07

D.4 Influence des temps de propagation homogènes

D.4.1 Sans pertes d'acquittements

Primo.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	93,38	93,39	99,02	99,95	100	96,09	51,02
10	93,27	93,33	99,88	99,9	100	94,86	37,93
20	93,23	93,33	99,89	99,95	100	92,23	39,41
30	93,2	93,33	99,99	99,97	100	89,62	41,32
50	93,12	93,33	99,96	99,97	100	84,54	42,76
100	93,12	93,33	99,96	100	100	71,25	46,57
200	92,92	93,33	99,99	100	100	44,92	51,99
Moyenne	93,18	93,34	99,81	99,96	100	81,93	44,43

Reno.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	47,7	52,36	96,07	96,15	92,36	31,42	73,1
10	41,52	46,18	94,49	96,15	93,14	32,84	70,18
20	34,47	40,21	96,12	96,16	93,73	38,9	62,85
30	29,61	34,69	95,77	96,15	94,09	44,61	54,35
50	19,9	25,38	93,12	92,09	93,53	59,71	24,5
100	5,12	8,27	88,16	86,4	97,53	70,74	0,34
200	10,53	7,71	70,21	72,07	98,42	85,52	33,81
Moyenne	26,98	30,69	90,56	90,74	94,69	51,96	45,59

Sack.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	49,28	51,51	96,19	96,15	92,41	29,47	74,89
10	42,72	46,79	96,02	96,15	93,11	31,13	71,62
20	36,27	40	96,15	96,15	93,75	36,99	65,01
30	30,49	34,66	95,85	96,15	94,11	42,57	56,75
50	16,56	23,04	90,09	94,83	93,03	54,5	36,41
100	2,53	5,47	82,89	89,38	97,56	67,99	6,57
200	1,81	1,08	75,89	80,95	98,52	82,18	28,29
Moyenne	25,67	28,94	90,44	92,82	94,64	49,26	48,5

TFRC.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	89,81	91,95	95,68	100	93,22	12,05	91,93
10	85,58	86,43	84,98	100	94,21	12,14	91,86
20	91,64	92,46	94,75	100	95,52	14,76	90,17
30	91,51	93,23	99,53	100	96,46	17,51	86,37
50	86,38	88,25	81,28	99,69	95,04	26,2	74,59
100	79,46	82,05	85,63	99,52	97,02	30,54	70,91
200	74,76	77,65	84,11	92,28	96,54	71	10,3
Moyenne	85,59	87,43	88,42	98,78	95,43	26,31	73,73

Vegas.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	79,9	80	83,35	100	100	85,82	90,72
10	91,5	91,69	75,03	100	100	85,04	88,55
20	66,34	66,66	99,97	100	100	96,22	78,62
30	35,85	0,09	73,4	100	100	87,96	85,55
50	31,12	0,04	79,9	100	100	88,31	82,56
100	5,21	6,61	86,13	99,05	100	87,12	86,95
200	14,93	12,8	75,34	92,47	100	86,81	55,12
Moyenne	46,41	36,84	81,87	98,79	100	88,18	81,15

D.4.2 Avec pertes d'acquittements

Primo.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	93,5	93,4	98,64	99,81	100	95,22	37,23
10	93,29	93,32	99,47	99,48	100	93,86	25,57
20	93,24	93,33	99,55	99,6	100	91,08	26,99
30	93,2	93,33	99,72	99,68	100	88,74	28,4
50	93,12	93,33	99,79	99,81	100	83,46	32,39
100	93,12	93,33	99,9	99,93	99,97	70,78	37,19
200	92,93	93,33	99,95	100	98,64	46,42	49,44
Moyenne	93,2	93,34	99,58	99,76	99,8	81,36	33,89

Reno.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	6,98	11	57,61	63,53	87,95	68	5,18
10	8,05	9,89	63,44	67,22	87,98	68,72	2,33
20	10,23	11,43	70,7	67,11	90,05	81,31	16,65
30	3,74	8,89	67,81	65,33	91,01	87,03	30,58
50	4,44	2,64	57,99	65,61	92,75	86,27	26,1
100	6,04	10,68	81,23	67,71	97,77	87,16	22,21
200	12,27	9,69	66,19	63,74	98,82	90,68	23,47
Moyenne	7,39	9,18	66,42	65,75	92,33	81,31	18,08

Sack.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	13,39	14,06	62,54	91,43	90	53,95	41,76
10	32,58	35,88	96,39	90,24	85,58	56,27	38,89
20	37,74	40,94	89,48	89,74	89,2	59,01	33,64
30	21,09	26,18	78,66	85,27	90,53	69,44	11,48
50	22,01	26,34	81,38	84,92	92,81	75,68	3,35
100	2,71	0,32	67,88	84,12	96,61	78,23	0,94
200	20,81	19,05	56,94	80,34	98,68	84,52	7,27
Moyenne	21,48	23,25	76,18	86,58	91,92	68,16	19,62

TFRC.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	86,34	88,7	98,67	96,52	86,14	17,03	89,47
10	90,06	90,72	95,75	96,33	94,83	18,08	88,53
20	86,59	88,26	98,46	96,41	93,13	22,45	84,87
30	84,06	85,92	87,97	96,24	95,88	26,28	79,52
50	87,84	90,76	88,27	96,41	96,04	34,49	65,93
100	82,94	85,03	97,48	96,67	97,92	47,78	51,54
200	71,7	74,54	73,04	85,49	97,61	90,18	33,44
Moyenne	84,22	86,28	91,38	94,87	94,51	36,61	70,47

Vegas.

Temps de propagation du lien partagé (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
5	24,45	28,36	19,35	19,32	100	99,27	92,04
10	5,69	0,81	21,47	22,38	100	99,06	85,06
20	3,23	2,21	20,66	20,66	100	99,61	91,8
30	0,55	3,55	19,59	19,57	100	99,29	91,2
50	0,25	1,3	22,5	22,54	100	99,23	90,3
100	8,95	7,53	46,05	49,69	99,92	98,72	68,57
200	6,74	8,77	50,41	65,24	99,83	95,49	13,72
Moyenne	7,12	7,5	28,58	31,34	99,96	98,67	76,1

D.5 Influence des temps de propagation hétérogènes

D.5.1 Sans pertes d'acquittements

Primo.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	93,27	93,33	99,88	99,9	100	94,86	37,93
1 - 2	93,25	93,3	99,89	99,91	100	93,82	35,57
1 - 5	92,77	93,35	99,26	99,93	94,74	91,37	23,9
1 - 10	92,87	93,62	98,44	99,93	94,95	90,75	26,19
1 - 15	92,91	93,82	98,08	99,89	93,73	90,05	26,71
1 - 20	93,22	94,07	97,74	99,86	94,68	90,54	29,05
1 - 50	93,67	96,4	95,19	99,74	95,9	89,26	30,41
Moyenne	93,14	93,98	98,35	99,88	96,29	91,52	29,96

Reno.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	41,52	46,18	94,49	96,15	93,14	32,84	70,18
1 - 2	43,1	47,69	84,77	96,16	92,58	34,96	67,73
1 - 5	52,49	59,42	64,68	96,15	92,74	36,77	68,06
1 - 10	12,54	18,78	26,17	96,15	92,39	35,3	66,7
1 - 15	3,03	0,99	20,33	96,15	93,06	35,87	66,62
1 - 20	12,73	9,91	14,72	96,15	92,6	34,94	67,8
1 - 50	26,26	24,09	10,94	96,15	90,72	36,01	68,8
Moyenne	27,38	29,58	45,16	96,15	92,46	35,24	67,98

Sack.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	42,72	46,79	96,02	96,15	93,11	31,13	71,62
1 - 2	43,89	48,09	95,97	96,15	92,02	32,5	71,99
1 - 5	28,84	25,85	25,63	96,15	94,31	32,56	72,45
1 - 10	91,85	96,62	0	96,15	96,07	32,25	72,54
1 - 15	51,25	58,79	42,28	96,15	91,52	31,49	71,37
1 - 20	38,41	44,49	40,01	96,15	90,58	31,97	71,7
1 - 50	91,7	96,62	0,02	96,15	95,41	32,78	72,26
Moyenne	55,52	59,61	42,85	96,15	93,29	32,1	71,99

TFRC.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	85,58	86,43	84,98	100	94,21	12,14	91,86
1 - 2	90,13	91,05	96,87	100	93,88	13,85	90,9
1 - 5	91,34	92,09	98,1	100	95,53	13,45	91,07
1 - 10	91,11	92,08	93,92	100	95,31	12,64	91,75
1 - 15	80,21	84,58	86,48	100	95,05	15,03	89,73
1 - 20	89,41	92,15	98,52	100	94,34	14,94	88,73
1 - 50	78,06	80,1	63,02	100	96	16,16	89,2
Moyenne	86,55	88,36	88,84	100	94,9	14,03	90,46

Vegas.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	91,5	91,69	75,03	100	100	85,04	88,55
1 - 2	92,96	93,33	99,99	100	100	85,81	88,18
1 - 5	94,08	94,26	97,32	100	100	86,53	86,59
1 - 10	90,35	90,91	72,73	100	100	85,65	84,16
1 - 15	99,03	100	80	100	100	88,78	85,54
1 - 20	80,52	81,25	34,49	100	100	89,2	79,94
1 - 50	96,68	98,2	80,77	100	100	87,47	87,25
Moyenne	92,16	92,81	77,19	100	100	86,93	85,74

D.5.2 Avec pertes d'acquittements

Primo.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	93,29	93,32	99,47	99,48	100	93,86	25,57
1 - 2	93,24	93,33	99,57	99,53	99,73	91,77	17,61
1 - 5	92,57	93,31	99,73	99,52	91,98	89,32	13,17
1 - 10	92,6	93,32	99,85	99,56	92,45	89,28	15,87
1 - 15	92,6	93,44	99,36	99,6	90,94	88,16	15,36
1 - 20	92,66	93,63	98,7	99,67	89,95	87,76	19,98
1 - 50	92,85	94,55	97,18	99,7	91,65	86,24	25,61
Moyenne	92,83	93,56	99,12	99,58	93,81	89,49	19,03

Reno.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	8,05	9,89	63,44	67,22	87,98	68,72	2,33
1 - 2	11,67	14,5	75,81	71,37	89,24	73,41	1,78
1 - 5	3,19	6,33	88,23	72,74	89,52	72,27	4,12
1 - 10	15,41	18,05	65,49	66,03	91,09	75,69	10,32
1 - 15	22,48	28,53	65,92	68,34	90,9	76,37	2,38
1 - 20	20,31	25,62	42,09	64,01	88,95	72,21	4,37
1 - 50	4,23	11,45	42,71	66,49	89,72	78,81	0,04
Moyenne	12,19	16,34	63,38	68,03	89,63	73,93	3,62

Sack.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	32,58	35,88	96,39	90,24	85,58	56,27	38,89
1 - 2	15,63	22,92	71,81	92,23	82,95	41,91	62,46
1 - 5	34,15	38,94	65,06	90,54	83,38	45,73	55,3
1 - 10	45,8	53,2	92,14	86,42	87,76	60,8	30,5
1 - 15	49,18	55,67	95,67	89,9	86,83	52,2	42,93
1 - 20	59,88	67,37	80,7	88,82	87,87	56,78	38,83
1 - 50	37,95	45,95	48,71	88,35	88,77	61,83	32,2
Moyenne	39,31	45,7	78,64	89,5	86,16	53,65	43,02

TFRC.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	90,06	90,72	95,75	96,33	94,83	18,08	88,53
1 - 2	88,08	88,3	91,79	96,34	91,85	18,68	87,28
1 - 5	89,36	90,31	91,89	96,5	91,23	17,95	88,94
1 - 10	84,25	84,19	78,33	96,33	94,42	20,44	86,57
1 - 15	85,62	88,92	87,85	96,48	94,6	21,08	85,41
1 - 20	82,98	84,6	71,81	96,19	96,66	24,13	83,25
1 - 50	73,11	77,83	57,49	96,48	95,35	25,5	81,95
Moyenne	84,78	86,41	82,13	96,38	94,13	20,84	85,99

Vegas.

Temps de propagation des liens d'accès (ms)	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
1 - 1	5,69	0,81	21,47	22,38	100	99,06	85,06
1 - 2	9,19	7,44	17,84	17,76	100	98,82	68,92
1 - 5	16,45	18,53	17,19	17,57	100	99,14	74,92
1 - 10	17,73	16,7	18,78	19,59	100	99,43	79,36
1 - 15	31,98	32	19,88	20,18	100	99,37	79,98
1 - 20	26,83	25,94	19,63	19,89	100	99,32	76,96
1 - 50	10,02	9,93	20,44	22,38	100	99,17	79,74
Moyenne	16,84	15,91	19,32	19,97	100	99,19	77,85

D.6 Influence du nombre de sources

D.6.1 Sans pertes d'acquittements

Primo.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	93,27	93,33	99,88	99,9	100	94,86	37,93
3	94,7	94,73	99,88	99,88	100	92,49	39,62
5	97,98	99,47	99,79	99,84	100	88,75	41,57
7	96,18	96,66	99,68	99,8	100	84,96	40,86
10	93,51	93,52	99,34	99,79	100	85,07	42,19
15	94,32	94,3	99,31	99,77	100	85,63	40,54
20	93,41	93,47	99,14	99,77	100	85,4	40,54
Moyenne	94,77	95,07	99,57	99,82	100	88,16	40,46

Reno.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	41,52	46,18	94,49	96,15	93,14	32,84	70,18
3	54,8	58,97	96,19	96,15	90,83	35,85	67,24
5	34,83	38,08	81,4	94,81	88,22	45,86	49,51
7	16,51	19,25	77,02	94,94	87,54	47,29	47,49
10	32,45	34,87	78,66	95,12	85,88	44,8	48,94
15	39,68	44,45	64,57	95,88	86,77	40,04	57,05
20	47,05	52,76	77,65	95,85	85,76	39,08	55,7
Moyenne	38,12	42,08	81,43	95,56	88,31	40,82	56,59

Sack.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	42,72	46,79	96,02	96,15	78,37	31,13	71,62
3	55,2	57,42	96,09	96,15	84,95	33,27	68,19
5	58,82	60,55	90,59	96,16	81,24	31,36	71,67
7	58,99	59,99	84,16	96,14	76,09	31,75	70,22
10	58,99	59,76	82,01	96,15	75,53	29,2	72,15
15	60,81	62,17	86,22	96,15	74,89	25,15	76,59
20	63,39	66,29	70,23	96,15	78,51	29,58	68,36
Moyenne	56,99	58,99	86,47	96,15	78,51	30,21	71,26

TFRC.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	85,58	86,43	84,98	100	94,21	12,14	91,86
3	87,08	88,8	95,06	100	90,75	9,95	95,21
5	90,12	91,21	93,97	100	82,08	10,9	94,63
7	85,82	86,65	95,95	100	76,91	10,93	95,21
10	86,03	85,94	94,85	100	74,09	8,66	95,75
15	86,91	86,83	95,25	100	73,24	6,67	96,72
20	88,14	87,64	91,06	100	74,01	5,96	96,65
Moyenne	87,1	87,64	93,02	100	80,76	9,32	95,15

Vegas.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	91,5	91,69	75,03	100	100	85,04	88,55
3	94,85	95	87,5	100	100	77,12	96,29
5	91,35	91,48	67,6	100	100	66,7	99,45
7	92,68	92,87	68,62	100	100	50,93	96,92
10	91,31	93,33	71,43	100	100	48	97,49
15	92,52	90,91	74,97	100	99,98	51,65	97,87
20	92,24	92,41	71,94	100	99,98	45,94	98,9
Moyenne	92,35	92,53	73,87	100	99,99	60,77	96,5

D.6.2 Avec pertes d'acquittements

Primo.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	93,29	93,32	99,47	99,48	100	93,86	25,57
3	95,08	95,08	99,37	99,46	100	91,3	28,87
5	97,56	98,76	99,4	99,5	100	87,61	33,15
7	95,98	96,33	99,47	99,54	100	83,73	35,54
10	93,51	93,51	99,39	99,62	100	84,12	36,98
15	94,49	94,54	99,18	99,63	100	84,82	37,11
20	93,5	93,54	98,85	99,69	100	84,88	38,32
Moyenne	94,77	95,01	99,31	99,56	100	87,19	33,65

Reno.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	8,05	9,89	63,44	67,22	87,98	68,72	2,33
3	47,06	50,59	77,47	91,62	91,15	48,3	45,16
5	28,95	30,84	84,11	91,79	89,33	51,81	38,3
7	23,44	25,56	77,61	91,57	87,73	53,49	33,68
10	31,48	36,05	67,78	94,06	87,22	47,13	45,52
15	27,5	29,64	77,16	95,04	87,02	43,28	50,66
20	40,58	42,17	75,88	95,57	87,34	39,95	55,86
Moyenne	29,58	32,11	74,78	89,55	88,25	50,38	38,79

Sack.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	32,58	35,88	96,39	90,24	85,58	56,27	38,89
3	60,25	68,42	86,11	94,01	89,39	39,6	59,52
5	58,81	61,71	91,05	94,84	82,13	34,49	67,37
7	63,22	64,76	83,81	94,81	78,79	38,42	59,85
10	60,22	61,66	83,12	95,44	77,31	31,91	68,58
15	60,68	61,84	87,12	95,69	76,17	27,48	73,22
20	60,72	57,38	38,56	95,82	77,14	25,75	75,01
Moyenne	56,64	58,81	80,88	94,41	80,93	36,28	63,21

TFRC.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	90,06	90,72	95,75	96,33	94,83	18,08	88,53
3	86,33	86,99	92,83	97,74	85,29	13,06	92,98
5	87,09	87,13	95,2	98,58	79,43	11,83	94,43
7	86,92	86,93	95,35	98,98	72,5	11,27	94,96
10	84,73	85,01	91,03	99,25	72,22	9,35	95,83
15	85,75	86,03	95,13	99,49	71,91	7,02	96,3
20	87,11	87,33	92,92	99,62	72,46	5,86	96,75
Moyenne	86,85	87,16	94,03	98,57	78,38	10,92	94,26

Vegas.

Nombre de sources	Constance du débit d'émission	Constance du débit de réceprtion	Équité moyenne	Taux d'occupation du lien partagé	Taux de bonne transmission	Taux d'espace libre dans la d'attente	Constance de la taille de la file d'attente
2	5,69	0,81	21,47	22,38	100	99,06	85,06
3	93,23	93,55	98,04	97,7	100	69,09	91,19
5	92,89	93,29	66,6	98,56	100	53,27	93,85
7	90,44	90,96	43,28	98,95	99,91	49,76	95,47
10	90,71	90,93	74,04	99,23	99,95	45,77	96,32
15	92,59	92,61	72,43	99,48	99,92	39,04	97,99
20	91,73	91,91	57,74	99,62	99,98	41,78	98,48
Moyenne	79,61	79,15	61,94	87,99	99,97	56,82	94,05

Glossaire des abréviations et sigles employés

ack : *acknowledgment.*
AIMD : *Additive Increase Multiplicative Decrease.*
API : *Application Programming Interface.*
ARP : *Address Resolution Protocol.*
ARPA : *Advanced Research Projects Agency.*
ATM : *Asynchronous Transfert Mode.*
CE : *Congestion Experienced.*
cwnd : *congestion window.*
CWR : *Congestion Window Reduced.*
DARPA : *Defense Advanced Research Projects Agency*.
DHCP : *Dynamic Host Configuration Protocol.*
DISA : *Defense Information Systems Agency.*
ECE : *ECN-Echo.*
ECN : *Explicit Congestion Notification.*
FDDI : *Fiber Distributed Data Interface.*
FIFO : *First In First Out.*
FTP : *File Transfert Protocol.*
IANA : *Internet Assigned Number Authority.*
ICMP : *Internet Control Messages Protocol.*
INRIA : Institut National de la Recherche Informatique et en Automatique.
IP : *Internet Protocol.*
IPG : *Inter Packets Gap.*
IRDP : *Internet Router Discovery Protocol.*
ISN : *Initial Sequence Number.*
ISO : *International Standardisation Organisation.*
IW : *Initial Window.*
LDA+ : *Loss Delay Adaptation Algorithm.*
MRU : *Maximum Receive Unit.*
MSL : *Maximum Segment Lifetime.*
MSS : *Maximum Segment Size.*
MTU : *Maximum Transmission Unit.*
NFS : *Network File System.*
NIMI : *National Internet Measurement Infrastructure.*
NSF : *National Science Foundation.*

OSI : *Open System Information.*
OSPF : *Open Shortest Path First.*
OTCL : *Object TCL.*
PASTRA : *PAth STatus RAte control.*
PDU : *Protocol Data Unit.*
PRNET : *Packet Radio NETwork.*
RAP : *Rate Adaptation Protocol.*
RARP : *Reverse Address Resolution Protocol.*
RED : *Random Early Detection.*
RIP : *Routing Interface Protocol.*
RIPE : Réseaux IP Européens.
RFC : *Request For Comments.*
ROTT : *Relative One way Trip Time.*
RTC :Réseau Téléphonique Commuté.
RTCP : *RTP Control Protocol.*
RTO : *Retransmit Time Out.*
RTP : *Real-time Transport Protocol.*
RTT : *Round Trip Time.*
Sack : *Selective acknowledgement.*
Fack : *Forward acknowledgement.*
SATNET : *SATellite atlantic packet NET.*
SMTP : *Simple Mail Transfert Protocol.*
ssthresh : *slow start threshold.*
STD : *STandard Document.*
TCP : *Transmission Control Protocol.*
TFRC : *TCP Friendly Rate Control.*
TOS : *Type Of Service.*
TP : *Transport Protocol.*
TTL : *Time To Live.*
UDP : *User Datagram Protocol.*
WALI : *Weighted Average of Loss Interval.*
XDR : *eXternal Data Representation.*

Index

Table des matières