TCP sur lien asymétrique : analyse des phénomènes et étude de solutions de faible empreinte mémoire ou de bout-en-bout

Tristan Braud

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE la Communauté UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Tristan BRAUD-MESSAGER

Thèse dirigée par **Martin HEUSSE**

et codirigée par **Guillaume URVOY-KELLER**

préparée au sein **UMR 5217 - LIG - Laboratoire d'Informatique de Grenoble**

et de **École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique (EDMSTII)**

TCP sur lien asymétrique
Analyse des phénomènes et étude de solutions de faible empreinte mémoire ou de bout-en-bout

Thèse soutenue publiquement le **11 Juillet 2016**,

devant le jury composé de :

Mme. Isabelle GUÉRIN LASSOUS

Professeur, Université Lyon I, Présidente

Mme. Catherine ROSENBERG

Professeur, Université de Waterloo, Rapporteur

M. André-Luc BEYLOT

Professeur, ENSEEIHT, Rapporteur

M. Martin HEUSSE

Professeur, ENSIMAG, Directeur de thèse

M. Guillaume URVOY-KELLER

Professeur, Université Nice Sophia Antipolis, Co-Directeur de thèse

Table des matières

Chapitre 1

Introduction

1.1 TCP, lien asymétrique et utilisateur final

La pile TCP/IP est la clé de voute des communications numériques actuelles, et, tout particulièrement d'Internet. Ainsi, le protocole TCP, à l'origine développé pour des applications simples comme la consultation de pages web ou l'envoi de courriel est de nos jours impliqué dans un grand nombre d'usages ; et, de plus en plus, en support de trafic interactif (navigation web, certaines applications de voix sur IP) ou à délai contraint (*streaming* vidéo HTTP).

Ces utilisations variées prennent place sur des liens de types divers, qui présentent des caractéristiques variées : filaires, sans fil, à délai ou bande passante variable, asymétriques ou symétriques, etc. Les liens asymétriques sont particulièrement représentés en bout de chaîne, que ce soit dans le cas d'une connexion ADSL ou cellulaire. Ceux-ci sont souvent le goulot d'étranglement pour le trafic, et sont plus susceptibles de subir des épisodes de congestions avec leurs conséquences : pertes, latences, etc.

L'utilisation de TCP sur des liens asymétriques peut donc entraîner des baisses de performances marquées, et nuire fortement à la qualité de service ressentie par l'utilisateur. En effet, il est fréquent qu'un lien asymétrique présente des tampons de même taille à chaque extrémité. De ce fait, dans le cas de connexions antiparallèles, de fortes latences apparaissent sur la voie montante, dues à un effet nommé *bufferbloat*. Ces latences sont souvent accompagnées par une sous-utilisation du lien descendant. À ces latences peuvent de surcroît s'ajouter celles induites par la perte de paquets en début de connexion, qui peuvent gravement handicaper un flux naissant.

Afin de contrer ces effets, diverses solutions existent, que ce soit de bout en bout par des modifications de la pile TCP/IP, ou en cœur de réseau avec divers mécanismes d'ordonnancement.

Cependant, les modifications effectuées sur la pile TCP/IP ont souvent une portée limitée, ou sup-

portent mal le passage à l'échelle. D'autres solutions de bout-en-bout ont pu voir le jour, au prix d'incompatibilités avec les implantations actuelles de TCP.

À l'autre extrémité du spectre, les techniques d'ordonnancement en cœur de réseau ont des caractéristiques qui limitent leur utilisation immédiate. Certaines d'entre elles demandent de garder longtemps en mémoire les flots qui ont utilisé le réseau pour pouvoir associer une priorité à chaque paquet qui se présente au goulot d'étranglement, moyennant une occupation mémoire importante. D'autres utilisent des algorithmes complexes, ce qui entraîne une consommation excessive de temps CPU (*Central Processing Unit* – Processeur).

Ceci est problématique car les routeurs sont souvent contraints en capacité de calcul et en mémoire. Par ailleurs, cette famille de politiques peut avoir des conséquences négatives sur certains types de trafic. Par exemple, des solutions comme *Least Attained Service (LAS)* bride les connexions longues, causant faibles débits et fortes latences. Ce phénomène, connu sous le nom de *starvation* (famine), peut être acceptable pour certains types de flux, mais est fortement handicapant dans d'autres cas. Ainsi, les communications voix sur IP, bien que de longue durée, nécessitent un débit modéré mais constant, et surtout de faibles délais (ces derniers doivent représenter moins de 300 ms aller-retour pour ne pas perturber une conversation [14]).

Un premier objectif de cette thèse est d'explorer comment un résultat similaire à celui obtenu par ce type d'ordonnancement sur le goulot d'étranglement peut être obtenu en travaillant de bout-en-bout, c'est-à-dire là où les ressources de calcul et de mémoire sont les plus abondantes. Ce questionnement est accompagné par une analyse en profondeur des phénomènes causant une dégradation des performances, ainsi que l'évaluation des solutions existantes. Finalement, des solutions sont apportées, testées sur banc d'essai et sur réseaux réels.

Expériences reproductibles

Un rappel de l'état de l'art amène très rapidement à se poser la question de la validité et de l'extensivité des expériences. Paramètres manquants, méthodologie incomplète et statistiques peu détaillées marquent de nombreux articles.

De plus, la recherche en réseaux de communications présente la contrainte de se focaliser sur l'analyse des échanges entre plusieurs systèmes. Outre les contraintes matérielles (matériel exotique et/ou coûteux), l'environnement dans lequel l'expérience est effectuée influence les résultats.

L'objectif du chapitre 3 est ainsi de se questionner sur une méthodologie, simple et bien identifiée, à adopter dans le but d'effectuer des expériences en environnement proche du réel, tout en garantissant la reproductibilité des résultats, *via* notamment la définition d'une manière de redistribuer

les sources.

Une fois la méthodologie expérimentale définie, il est possible de s'attaquer au cœur du sujet : les fortes latences qui découlent des tampons surdimensionnés aux extrémités du réseau.

Effet de balancier et *Bufferbloat* sur liens asymétriques

Les dynamiques des connexions TCP soumettent les tampons d'un goulot d'étranglement à un effet de balancier : un seul tampon peut être occupé à un instant donné, tandis que le lien de retour est légèrement sous-utilisé.

Ce phénomène a très peu d'impact sur les liens symétriques ; toutefois, dans le cas de liens asymétriques, le temps passé dans chaque tampon est directement fonction de leur taille respective. Les liens asymétriques, fréquents aux extrémités du réseau (lien cellulaire, ADSL, etc.), ont souvent tendance à présenter des tampons de taille égale sur le lien montant et descendant, causant ainsi de fortes latences, qui se répercutent à la fois l'*upload* et le *download*.

L'analyse de ces phénomènes et de ses conséquences est faite en deux temps : tout d'abord, à travers une modélisation du phénomène dans le chapitre 4, suivie par une étude extensive du comportement de différentes variantes de TCP face à ces conditions au chapitre 5.

Une solution originale aux problématiques liées aux tampons surdimensionnés

Si les problématiques liées au *bufferbloat* ont été soulevées très tôt dans l'histoire des réseaux informatiques, les solutions sont relativement récentes, et, de ce fait nécessitent une intervention sur les infrastructures existantes.

Certains protocoles de transport permettent, par effet de bord, de minimiser l'impact des tampons surdimensionnés, mais ces derniers sont souvent limités, et supportent difficilement la cohabitation avec d'autres connexions.

Une des solutions envisageables serait de pouvoir réduire virtuellement la taille des tampons en jouant sur la manière dont le trafic est envoyé. Pour ce faire, il est possible d'exploiter le fait que la plupart des files d'attente sont dimensionnées en paquets et d'envoyer un trafic "de fond" composé de petits paquets, rapides à transmettre, afin qu'un certain nombre d'emplacements du tampon soient occupés en permanence. Cette solution atypique est présentée en détail dans le chapitre 6.

Le *bufferbloat* est-il la seule cause de latence dans les réseaux ?

Si le *bufferbloat* est une des causes principales de fortes latences dans le réseau, la majorité du trafic interactif est issu de petites connexions, très sensibles à la perte de certains paquets.

La plupart des solutions pour contrer latences induites par des tampons surdimensionnés se fondent très souvent sur un système de pertes de paquets fréquents afin de forcer les connexions TCP à réduire leur rythme d'émission. Ces mécanismes présentent plusieurs inconvénients : en plus de nécessiter un déploiement massif sur les infrastructures, leurs algorithmes suppriment les paquets de manière aveugle. Or, certains paquets de signalisation doivent être fortement protégés afin d'éviter de longs temps de récupération, qui peuvent dépasser de plusieurs ordres de grandeur le temps de transmission des données.

Combiner délais contraints et protection des paquets de signalisation permet alors d'éviter les longs délais en début et fin de connexion à la suite de pertes non récupérables par les algorithmes traditionnels. Pour ce faire, une des contributions principales du chapitre 7 repose sur la protection de certaines catégories de paquets par des politiques de gestion des files fondées sur les délais, en comparaison des principaux algorithmes de gestion des files d'attente existants.

Contributions

Au cours de ce document, les contributions suivantes seront présentées :

— Si le phénomène de balancier de données est connu depuis peu, il était jusqu'à présent impossible de prédire de quel coté le tampon tendra à se remplir, ainsi que l'influence des différents paramètres du système sur les débits observés pour les connexions montantes et descendantes. Le Chapitre 4 résout cette problématique à travers un modèle mathématique qui permet d'expliquer la majorité des phénomènes à l'origine du comportement de connexions TCP qui transfèrent des données en sens opposés sur les même liens (connexions antiparallèles).

— Cette analyse théorique est suivie par une étude expérimentale systématique des conséquences de l'effet de balancier pour diverses variantes de TCP. L'utilisation du banc d'essai décrit chapitre 3 permet la mise en place d'expériences reproductibles avec de bons intervalles de confiance, ainsi que la réalisation de tests poussés dans de multiples configurations.

— À la suite de ces constatations, tant théoriques qu'expérimentales, une solution atypique permettant de résoudre ces problématiques – et plus largement réduire les latences dues au *bufferbloat* – est proposée. L'étude de l'impact de cette solution est effectuée dans des conditions similaires à celle du Chapitre 5.

— Enfin, les latences excessives peuvent avoir d'autres causes que des tampons surdimensionnés.

Sur ce point, l'étude des pertes de SYN comme cause de latence dans les communications TCP/IP est très peu abordée dans la littérature. Le Chapitre 7 se focalise sur cette étude à travers l'analyse de traces publiques et la proposition d'une solution combinant faible latence et protection des paquets de données. Cette solution est ensuite comparée à un nombre conséquent de politiques de files d'attente, selon la même méthodologie expérimentale suivie au cours de ce manuscrit.

Chapitre 2

État de l'art

Nice shot Doc ! You're not gonna believe this, we gotta go back to 1955.

Marty McFly – Back to the Future Part II

D'ARPANET à Internet tel que nous le connaissons aujourd'hui, la pile TCP/IP *Transmission Control Protocol/Internet Protocol* a été une composante incontournable de l'essor des réseaux informatiques. Dans de ce chapitre, les principales contributions qui ont participé au développement des technologies qui composent Internet seront présentées, avec une emphase particulière sur les problématiques de contrôle de congestion, délai et bande passante, que ce soit en cœur de réseau ou de bout-en-bout.

2.1 TCP et Internet

Le succès du *Transmission Control Protocol* réside dans ses propriétés, définies dès la naissance du protocole et qui sont restées d'actualité tout au long de ces 40 dernières années. En plus de garantir une transmission fiable et ordonnée d'un flux de données au sein de réseaux à commutation de paquets, TCP se focalise sur une utilisation optimale des ressources à sa disposition. Pour ce faire, les algorithmes de contrôle de congestion recherchent la capacité disponible, puis se fondent sur les pertes pour adapter le rythme de transmission. Dans le cas d'une perte, des mécanismes de retransmission permettent de récupérer les paquets afin d'assurer la transmission de l'intégralité du flux de données. Ce fonctionnement permet également à plusieurs connexions TCP de cohabiter sur un même lien de manière équitable. De bout-en-bout, l'utilisation est régulée en jouant sur la quantité de données en cours d'acheminement dans le réseau qui doit correspondre au produit délai-bande passante afin de garantir une bonne utilisation de la capacité disponible.

2.1.1 Les balbutiements des réseaux informatiques

Les premiers concepts de réseaux de communication à commutation de paquets apparaissent très tôt dans l'histoire de l'informatique, dès le début des années 1960. Pour replacer ces idées dans leur contexte, à la même époque, les transistors discrets commencent tout juste à remplacer les tubes électroniques dans la conception des ordinateurs, tandis que les circuits intégrés n'apparaîtront qu'à la fin de la décennie (*cf.* Figure 2-1 [1]).

La première description documentée de communication entre ordinateurs remonte à 1960, dans une série d'articles écrits par J. C. R. Licklider.

Le premier article, *Man-Computer Symbiosis* [15] développe les premières idées d'interaction homme-machine. Deux ans plus tard, *On-line man-computer communication* [16] propose de nouvelles initiatives telles que les systèmes à temps partagés et la communication inter-processus afin d'améliorer non seulement la coopération entre homme et ordinateur, mais aussi les interactions entre humains.

Finalement, en avril 1963, Licklider adresse un mémo aux *Membres et Affiliés du Réseau Informatique Intergalactique* [2] [17], suite logique des deux premiers articles. Cette note pose les bases d'interactions entre ordinateurs, toujours dans l'objectif de parfaire la collaboration entre l'homme et la machine. Licklider souhaite en effet permettre aux usagers d'accéder aux ressources de machines autres que la leur. En effet, à cette époque, chaque ordinateur était le plus souvent affecté à une unique tâche.

Il prend pour exemple un utilisateur qui souhaite effectuer des calculs scientifiques sur un jeu de

1. Cette image a été originellement postée sur Flickr par Hiddenloop à l'adresse `http://www.flickr.com/photos/hiddenloop/307119987/` sous licence cc-by-2.0 (`https://creativecommons.org/licenses/by/2.0/`).
2. *Members and Affiliates of the Intergalactic Computer Network*

FIGURE 2-1: Le PDP-1 : un ordinateur à la pointe de la technologie (et de la miniaturisation) en 1960

données. Toutefois, cet utilisateur ne possède qu'un ensemble limité de programmes : il lui manque entre autres certains algorithmes d'ajustement de courbes. Sans réseau informatique, il doit donc demander les bandes magnétiques qui contiennent ces programmes aux usagers qui les possèdent, ce qui est possible lorsque ladite bande magnétique est détenue par un utilisateur voisin, mais impensable dans le cas de supports stockés dans différentes villes.

Une autre solution serait de pouvoir accéder aux ressources d'autres machines que la sienne sans avoir à se déplacer. Dans ce but, Licklider introduit les premiers concepts d'un réseau informatique, qui comprendrait "au moins quatre gros ordinateurs, peut-être six ou huit petits ordinateurs, et un grand assortiment de disques et de lecteurs de bande magnétique"[3] (Le réseau intergalactique de l'époque).

En parallèle, en 1961, Leonard Kleinrock soumet sa proposition de thèse [18] dans lequel sont évoqués pour la première fois les réseaux à commutation de paquets. Pour compenser les problèmes apparaissant dans les réseaux de communications, il propose alors de diviser le trafic en unités d'information stockée dans des tampons en cœur de réseau.

Ces idées ne tardent pas à se propager, et en 1965, Lawrence Roberts tente la première communication par commutation de paquets entre deux machines situées respectivement au Massachusetts et en Californie à travers une ligne téléphonique [19]. Quelques années plus tard, divers réseaux appa-

3. "at least four large computers, perhaps six or eight small computers, and a great assortment of disc files and magnetic tape units"

raissent, parmi lesquels CYCLADES (France), NPL (Angleterre) et ARPANET (US). Néanmoins, de par leurs architectures disparates, ces réseaux ne peuvent pas communiquer entre eux.

Dans un effort de standardisation, Steve Crocker publie la première *Request For Comment* (RFC) le 7 avril 1969, premier d'une longue liste de documents de discussion et de définition des protocoles réseaux. Toutefois, si les spécifications sont désormais ouvertes, la compatibilité entre les protocoles des différents réseaux n'est pas toujours de mise.

Très rapidement, le besoin de communication entre réseaux quelle que soit l'architecture sous-jacente apparaît, et deux chercheurs — Vinton Cerf (Stanford University) et Robert E. Khan (DARPA) — commencent à travailler sur le sujet. En mai 1974, ils proposent une reformulation intégrale des principes de base des réseaux à travers un article intitulé *A protocol for Packet Network Intercommunication* [20], qui leur vaudra la réputation de pères fondateurs d'Internet.

Ils définissent alors un protocole monolithique afin de partager des ressources de manière transparente entre ordinateurs connectés à différents réseaux de paquets. Bien que différent des versions modernes de TCP dans son architecture – les couches réseau et transport sont regroupées au sein d'un même protocole, le contrôle de flux est géré par des notifications explicites du récepteur – ce protocole introduit les concepts majeurs de la pile TCP/IP actuelle tels que :

- Les nœuds sont classés en deux catégories : les *passerelles* font l'interface entre deux réseaux, tandis que les *hôtes* communiquent de manière transparente, en utilisant les passerelles pour faire transiter l'information.
- Le protocole opère au-dessus d'une couche liaison réduite au strict minimum.
- Les deux extrémités peuvent effectuer de manière transparente segmentation et réassemblage d'un flux de données.
- Une transmission fiable est assurée grâce à des mécanismes de détection et récupération des pertes.
- Des algorithmes permettent un contrôle de flux basique.
- Le protocole est orienté connexion, avec des dispositifs spécifiques de signalisation pour l'établissement et la fin d'un flux de données.

À la suite de cet article, les auteurs publient la RFC 675 [21], dans le but de formaliser le protocole. Le développement de TCP continue alors sur ces bases au cours des années suivantes, avec la publication de *Specification of Internet Transmission Control Program TCP (version 2)* [22] en mars 1977 par Vinton Cerf.

Cependant, en août 1977, Jon Postel exprime des inquiétudes quant à l'architecture de TCP, à travers une note intitulée *Comments on Internet Protocols and TCP* [23] et qui commence par cette déclaration sans ambiguïté : *"Nous sommes en train de foirer le design des protocoles d'Internet*

en violant le principe d'encapsulation"[4]. En d'autres mots, dans l'approche monolithique jusqu'alors proposée, il n'y a aucune distinction entre les opérations au niveau de l'hôte et celles qui permettent l'acheminement à travers le réseau. Cette architecture a pour résultat un mélange entre les opérations de routage avec celles concernant l'empaquetage des données, alors que les modèles d'encapsulation existants (type modèle OSI) suggèrent une conception plus modulaire et donc plus lisible.

À la suite de cette déclaration, les opérations de routages sont séparées de TCP dans un nouveau protocole, nommé Internet Protocol (IP). La désormais célèbre pile TCP/IP, décrite dans la RFC 760 [24] était née.

Ce document est rapidement suivi par les RFC 791 [25], 792 [26] et 793 [27] qui formalisent respectivement les protocoles IP, ICMP (Internet Control Message Protocol) et TCP.

Ces nouveaux standards sont très vite adoptés, et le premier janvier 1983, la pile TCP/IP devient l'unique protocole autorisé sur ARPANET.

2.1.2 Réguler le trafic pour répondre aux congestions

Alors que de plus en plus d'hôtes rejoignent ce nouveau réseau d'intercommunication, de nouvelles problématiques telles que les effondrements dus à la congestion (*congestion collapse*) apparaissent. Ces effondrements surviennent lorsqu'une passerelle subit une telle charge que la congestion empêche le trafic de transiter.

Des algorithmes de contrôle de congestions robustes deviennent alors nécessaires. En effet, les premières versions de TCP se basent uniquement sur la fenêtre annoncée par le récepteur pour effectuer le contrôle de flux. Compte tenu de l'absence de mécanismes adaptatifs, la seule technique efficace pour récupérer d'un épisode de congestion est alors l'arrêt pur et simple de la transmission.

John Nagle [28] répond à ces problématiques en 1984. Il identifie alors les deux problèmes suivants :

Les petits paquets : Lorsque seulement quelques octets doivent être transmis sur un réseau, 40 octets d'en-têtes sont rajoutés par la pile TCP/IP, ce qui représente un *overhead* conséquent pouvant atteindre plus de 95% des données transmises. Ce phénomène arrive fréquemment pour des sessions Telnet, où les données sont envoyées caractère par caractère, soit octet par octet. Les conséquences sont dramatiques sur des réseaux fortement congestionnés. La solution apportée à ce problème réside dans la temporisation de l'envoi des petits paquets : dès qu'un petit paquet est en vol, et tant qu'il n'est pas acquitté, l'émetteur retarde la transmission jusqu'à avoir l'équivalent d'un MSS (*Maximum Segment Size*) de données à envoyer. Cet algorithme est connu sous le nom d'algorithme de Nagle.

4. *"We are screwing up in our design of internet protocols by violating the principle of layering"*

Source quench : Une passerelle sous congestion peut envoyer un paquet ICMP de type *Source Quench* (mécanisme aujourd'hui obsolète) pour signaler l'engorgement aux hôtes. La RFC propose une optimisation de cet algorithme en deux points du réseau : tout d'abord, une passerelle doit envoyer ces messages avant que le tampon ne déborde, car la récupération de perte est coûteuse sans contrôle de congestion. côté émetteur, une première ébauche de contrôle de congestion est défini : à la réception d'un paquet *Source Quench*, l'émetteur doit arrêter de transmettre le temps de recevoir un certain nombre d'accusés de réception (10 selon la RFC).

Ces recommandations représentent la première tentative de résolution des problèmes de plus en plus fréquents de congestion, avec un succès très relatif. En octobre 1986 a lieu un épisode d'effondrement majeur, premier d'une longue série de perturbations réseau. Ces épisodes paralysent des portions entières du réseau, occasionnant des baisses de débit de plusieurs ordres de grandeur. Fasciné par ce phénomène, Van Jacobson *et al.* proposent une solution basée sur le "principe de conservation des paquets" [29]. Ce principe établit que pour une fenêtre complète de données en transit, un nouveau paquet ne peut être envoyé que lorsqu'un autre quitte le réseau. Ainsi, le même nombre de paquets reste en transit, et la congestion est évitée.

Selon cet article, seules trois situations peuvent mener à une violation du principe :

— une connexion n'arrive pas à l'équilibre.
— un nouveau paquet est injecté avant qu'un segment ait quitté le réseau.
— une connexion ne s'adapte pas pour atteindre un nouvel équilibre.

En se basant sur ces postulats, un nouvel algorithme de contrôle de congestion est défini, dans l'objectif de résoudre une fois pour toutes ces perturbations.

Sur l'observation qu'un récepteur ne peut pas générer des accusés de réception plus rapidement qu'il reçoit des données, les auteurs décrivent un algorithme à "cadencement naturel" (*self clocking*), adaptatif selon les conditions du réseau.

Ce mécanisme repose sur l'introduction d'une fenêtre côté émetteur nommée "fenêtre de congestion" (*congestion window*). Cette fenêtre est distincte de celle annoncée par le récepteur : la quantité de données que l'émetteur est autorisé à envoyer est calculée comme le minimum des deux fenêtres.

La fenêtre de congestion est utilisée en coordination avec un algorithme de contrôle de congestion qui respecte le principe de conservation des paquets en répondant aux trois situations problématiques listées plus haut.

La connexion n'arrive pas à l'équilibre

Pour "démarrer le cadencement", et ainsi permettre à un flux d'atteindre l'équilibre, l'auteur propose l'algorithme *slow start* : une connexion commence avec une fenêtre de congestion minimale – un paquet. Pour chaque accusé de réception reçu, la fenêtre est incrémentée d'un paquet. Ainsi, la fenêtre de congestion double à chaque RTT (*Round Trip Time* – Temps d'Aller-Retour). Cette progression exponentielle ainsi permet d'explorer rapidement plusieurs ordres de grandeur.

Cette phase permet de rapidement atteindre la bande passante du lien avec un impact négligeable sur les performances. Un flux sort de cette phase à la suite de la première perte. Ainsi, si les tampons en cœur de réseau sont bien dimensionnés, l'émetteur ne peut dépasser la bande passante disponible que d'un facteur deux.

Un nouveau paquet est injecté avant qu'un segment ait quitté le réseau

Pour garantir la sortie d'un paquet du réseau avant tout nouvel envoi de données, la manière dont le *timer* de retransmission est calculé a été revue : l'estimation est effectuée en calculant la moyenne et la variance en lieu et place du filtre passe bas précédemment utilisé. Cette nouvelle méthode permet de limiter les retransmissions superflues qui peuvent advenir lorsqu'un paquet subit des latences en cœur de réseau.

Cette technique permet donc d'augmenter la certitude qu'une perte a bien eu lieu au moment de l'expiration du *timer* de retransmission. De plus, une perte étant la plupart du temps (99%) due à une congestion, l'expiration d'un *timer* devient ainsi un indicateur fiable de congestion.

Une connexion ne s'adapte pas pour atteindre un nouvel équilibre.

En cas de perte, l'article propose de diviser la fenêtre de congestion pour forcer la source du trafic à ralentir proportionnellement à la vitesse de remplissage des tampons.

Entre deux pertes, une connexion doit toujours essayer d'exploiter toute la bande passante disponible. Elle doit donc effectuer un sondage constant de l'état du lien pour explorer la bande passante. Pour ce faire, la fenêtre de congestion est incrémentée par une faible quantité pour chaque acquittement reçu.

La fenêtre de congestion est ainsi incrémentée de 1 à chaque acquittement, et divisée par 2 en cas de perte. Ceci permet de suivre le principe *AIMD (Additive Increase Multiplicative Decrease)* décrit par Jain *et al* [30] en 1987. Cette règle permet d'atteindre progressivement un partage équitable du débit du lien entre les différents flots comme montré Figure 2-2.

Ces modifications ont été intégrées en 1988 dans 4.3-BSD, mieux connue sous le nom de BSD *Tahoe*

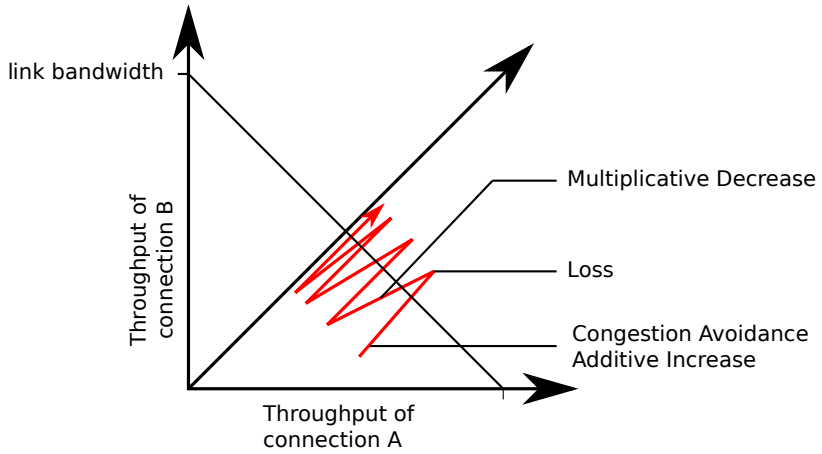


FIGURE 2-2: Atteindre l'équité pour deux connexions partageant le même lien

(d'où l'appellation TCP *Tahoe*). L'adoption de ces révisions a rapidement eu un effet notable sur l'état du réseau, en limitant les effondrements dus aux congestions.

Si la majeure partie de cet algorithme est encore utilisée de nos jours, deux révisions majeures, nommées respectivement *Reno* (1990 - RFC 2001 [31], 2581 [32] et 5681 [33]) et *New Reno* (1996, RFC 6582 [34]), ont été développées afin d'améliorer la détection et la récupération de pertes, grâce à l'introduction des algorithmes *fast retransmit* et *fast recovery*.

Avant TCP *Reno*, les pertes étaient uniquement détectées et résolues suite à l'expiration d'un *timer* de retransmission. Avec *fast retransmit* et *fast recovery*, pour chaque acquittement (ACK) dupliqué reçu, un nouveau paquet est envoyé afin de garder le lien actif. Lorsque trois acquittements dupliqués sont reçus, le paquet correspondant est considéré comme perdu et il est immédiatement retransmis. La fenêtre de congestion est alors divisée par deux, et la transmission reprend lorsque suffisamment d'acquittements ont été reçus pour contrebalancer cette réduction.

Si ce mécanisme permet de récupérer efficacement d'une perte unique, la connexion s'appuie souvent sur l'expiration d'un *timer* dans le cas de pertes multiples, avant de repartir en *slow start* avec une fenêtre de congestion remise à sa valeur initiale. En effet, dans le cas de pertes multiples, TCP *Reno* divise la fenêtre de congestion par deux pour chaque perte, jusqu'à un minimum de 2 paquets.

La solution à ce problème est apportée par TCP *New Reno* qui introduit le concept d'"épisodes de congestion". Au cours d'un de ces épisodes un acquittement considéré comme partiel peut être reçu. Ce segment accuse réception de nouvelles données sans que l'intégralité de la fenêtre ait été reçue. À la réception de cet acquittement partiel, l'émetteur considère le paquet correspondant comme perdu et le retransmet immédiatement. L'algorithme s'arrête lorsque l'intégralité de la fenêtre de congestion au moment de la perte a été acquittée. La fenêtre de congestion n'est divisée qu'une fois par épisode de congestion, évitant ainsi de faire appel à l'expiration d'un *timer*.

2.2 Optimiser l'utilisation du lien

L'introduction de TCP *Tahoe* et de ses successeurs a posé les bases du contrôle de congestion, et résolu la majeure partie des problèmes liés aux effondrements dus aux fortes congestions. L'intérêt s'est donc porté sur l'occupation du lien, et l'équité des flots le partageant, dans l'objectif d'atteindre le comportement idéal suivant :

> *n connexions se partageant un goulot d'étranglement de capacité C utilisent chacune une fraction C/n de cette dite capacité quelle que soit leur origine.*

Plusieurs stratégies visant à optimiser l'occupation du goulot d'étranglement seront décrites dans cette partie : tout d'abord, le protocole original permet une extension de ses fonctionnalités par l'introduction d'une variété d'options. Il est également possible de modifier les algorithmes de contrôle de congestion, depuis des variantes rétrocompatibles de TCP jusqu'à la définition de protocoles de transport inédits. En parallèle, de nombreuses techniques ont été développées en cœur de réseau pour optimiser la gestion des tampons conjointement au contrôle de congestion effectué de bout-en-bout.

2.2.1 La signalisation par les options TCP

Le protocole TCP tel que décrit dans la section précédente permet d'ajouter diverses options pour améliorer la (re)transmission des données. De par leur influence sur le comportement des deux hôtes, la plupart d'entre elles doivent être négociées au démarrage de la connexion.

Window Scaling [1]

Avec l'augmentation exponentielle des capacités des liens, le champ *Window size* des paquets TCP est rapidement devenu trop limité pour prendre en compte les bandes passantes les plus élevées pour des délais usuels. En effet, ce champ est codé sur 16 bits, soit des valeurs comprises entre 0 et 65535 octets, soit un débit maximal de 3.5 Mb/s pour un délai de 100 ms.

L'option *Window Scaling* permet d'étendre l'encodage des valeurs des fenêtres annoncées à 32 bits en spécifiant une valeur multiplicative à appliquer aux 16 bits du champ *Window size*. Cette valeur, encodée comme une puissance de 2 a une valeur maximale de 14, ce qui permet à deux hôtes, après négociation, d'atteindre des fenêtres de $2^{16} << 14 = 1,073,725,440$ octets.

Selective Acknowledgement – SACK [2]

Si TCP *Reno* a apporté des améliorations significatives aux mécanismes de contrôle de congestion de TCP, cette révision ne pouvait gérer qu'une perte à la fois. Les pertes suivantes ne pouvaient alors

être résolues que par l'expiration d'un *timer* de retransmission (*Retransmission Timeout* – RTO), comme nous l'avons vu section 2.1.2.

La solution proposée par SACK réside dans le fait d'acquitter individuellement chaque bloc de paquets correctement reçu dans chaque accusé de réception. À partir de cette information, l'émetteur peut déduire quels segments ont été perdus et les retransmettre immédiatement.

Néanmoins, SACK ne permet pas à un hôte de signaler les segments dupliqués qu'il reçoit, ce qui peut causer une certaine confusion dans les retransmissions. Une extension nommée *Duplicate SACK* [35] a été créée pour pallier ce problème.

Cette solution a été développée en parallèle à TCP *New Reno*, qui s'attaque au même problème sous un autre angle, comme nous l'avons montré section 2.1.2.

Explicit Congestion Notification – ECN [3]

ECN est une méthode de détection de congestion qui fait intervenir les routeurs en cœur de réseau. Ces derniers signalent la congestion aux hôtes finaux qui réagissent alors en conséquence à la place des pertes de paquets. Ainsi, le flux n'est pas interrompu par les récupérations de pertes.

Dans ce cas précis, non seulement les deux hôtes négocient l'utilisation d'ECN au cours de l'ouverture de la connexion, mais les routeurs intermédiaires doivent aussi supporter l'option pour que l'algorithme fonctionne.

Lorsqu'un paquet arrive dans une file dont l'occupation a dépassé un certain palier, le routeur le marque avec le *flag Congestion encountered* dans l'entête IP au lieu de le retirer du réseau. Un hôte qui reçoit un paquet marqué retransmet alors l'information dans l'entête TCP à son homologue. La réception d'un tel paquet entraîne une réduction de la fenêtre de congestion de manière similaire à une perte.

2.2.2 Variantes du contrôle de congestion

Si TCP *New Reno* est encore employé de nos jours dans la majorité des systèmes, de nombreuses variantes ont été développées. Au contraire des options TCP, ces variantes ne nécessitent pas de négociation préalable et sont inter-compatibles avec toutes les autres versions de TCP.

TCP Vegas

TCP *Vegas* [36], développé en 1994, utilise le délai des paquets au lieu des pertes pour détecter la congestion. En effet, une augmentation du temps d'aller-retour est signe d'une augmentation d'une file d'attente au goulot d'étranglement, et donc correspond à l'apparition d'une congestion.

Pour ce faire, TCP *Vegas* estime le débit à travers la formule suivante : $X = cwnd/RTT$. Le résultat de ce calcul est ensuite soustrait au débit théorique estimé par $X_{th} = cwnd/RTT_{min}$. Soit *Diff* cette différence. Deux paliers $\alpha < \beta$ sont définis, tels que :

— $Diff < \alpha$: *Vegas* augmente la fenêtre de congestion de manière linéaire pendant le prochain RTT pour explorer la bande passante disponible

— $\alpha < Diff < \beta$: la fenêtre de congestion reste constante pour le prochain RTT.

— $Diff > \beta$: la fenêtre de congestion diminue linéairement pendant le prochain RTT afin de résoudre la congestion

TCP *Vegas* modifie également la phase de *slow start*, potentiellement coûteuse en terme de pertes. À cette fin, la fenêtre de congestion double à chaque RTT. Toutefois, lorsque *Diff* est plus grande qu'un palier γ, l'algorithme passe en *congestion avoidance*.

Les phases de *slow start* et *congestion avoidance* sont représentées Figure 2-3.

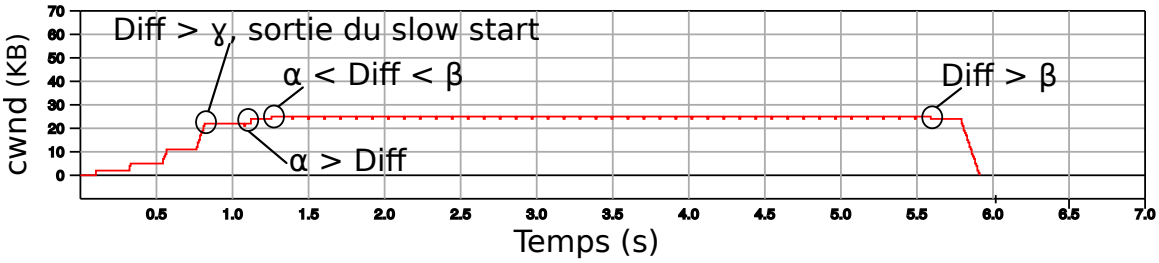


FIGURE 2-3: Évolution de la fenêtre de congestion de TCP *Vegas* au cours du temps

Cette approche a été largement critiquée, en particulier les augmentations et diminutions linéaires qui ne permettent pas d'atteindre l'équité pour plusieurs connexions partageant un même lien contrairement aux algorithmes AIMD. De plus, *Vegas* cohabite très mal avec Reno, et subit alors de fortes baisses de performances [37].

BIC TCP

Binary Increase Congestion Control (2004) [38] a été créé pour les liens à fort produit délai-bande passante. Cette variante introduit un nouvel algorithme appelé *Binary Search Increase* basé sur les minimums et maximums locaux atteints par la fenêtre de congestion.

Lorsque la fenêtre de congestion est réduite suite à une perte, ces deux valeurs sont mises à jour, et la fenêtre est fixée au point médian. Si la distance entre la fenêtre de congestion et sa dernière valeur est supérieure à S_{max}, l'algorithme rentre dans une phase dite *Additive Increase*, dans laquelle la fenêtre augmente de S_{max} à chaque RTT. À la suite de cette phase, et si aucune perte n'a eu lieu, *Binary Search Increase* se met en route. La fenêtre de congestion est incrémentée selon l'équation suivante pour chaque acquittement reçu :

$$cwnd+ = \frac{W_{max} - cwnd}{2 * cwnd} \tag{2.1}$$

Ainsi, la fenêtre de congestion remonte rapidement suite à une perte avant de se stabiliser aux alentours de la valeur maximale atteinte par la fenêtre lors du précédent épisode de congestion. Lorsque cette valeur est dépassée, l'augmentation devient alors linéaire. Les variations de la fenêtre de congestion sont représentées Figure 2-4 [38].

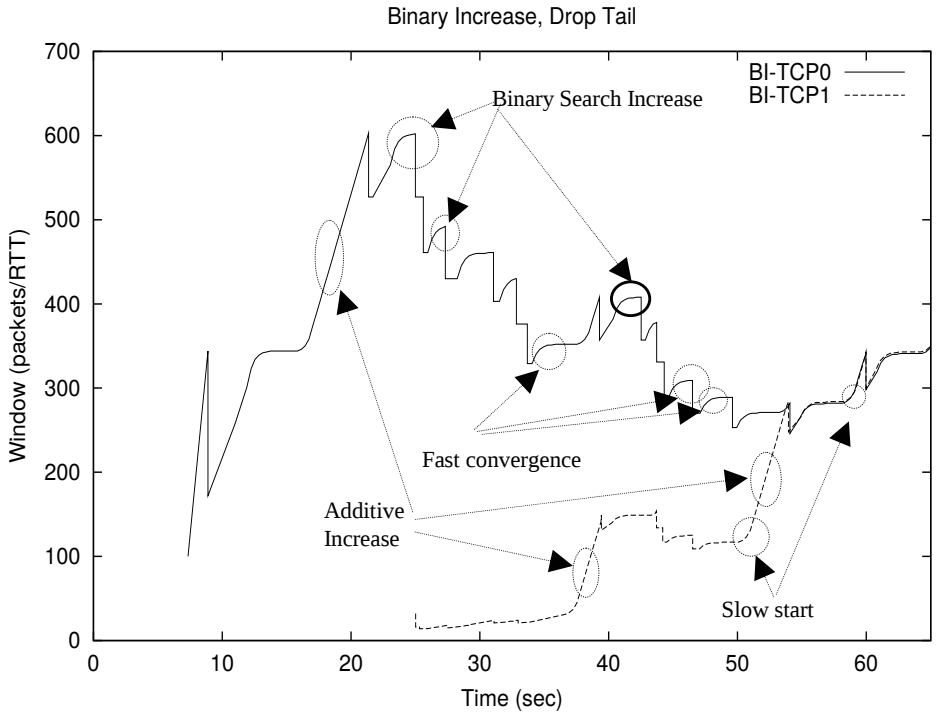


FIGURE 2-4: Évolution de la fenêtre de congestion de BIC TCP pour deux connexions BI-TCP0 et BI-TCP1

Toutefois, ce mode de fonctionnement peut être trop agressif pour de faibles RTT. Le successeur de BIC, *CuBIC* (2008) [39] met à jour la fenêtre de congestion proportionnellement à une fonction cubique du temps écoulé depuis la dernière congestion, centrée autour de la dernière valeur maximale atteinte. Ainsi, *CuBIC* réduit l'accélération de la fenêtre de congestion autour de la dernière valeur maximale. Les différences entre BIC et *CuBIC* sont présentées Figure 2-5 [40]. Cette nouvelle version de l'algorithme n'est cependant pas exempte de défauts : mauvais positionnement du plateau, synchronisation des pertes etc. [41]

Ces deux versions ont été implantées et régissent la pile TCP par défaut dans le noyau Linux depuis 2004. Depuis quelques années, elles sont également proposées en option dans les noyaux FreeBSD.

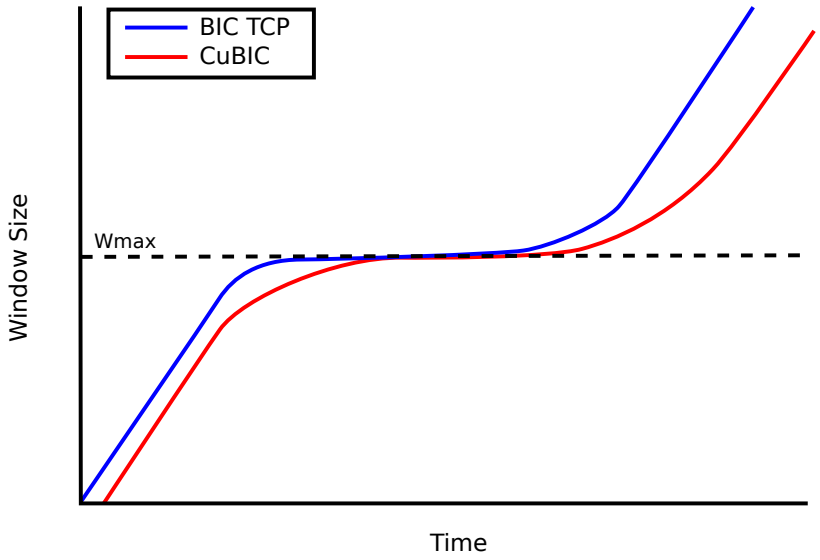


FIGURE 2-5: Augmentation de la fenêtre de congestion de *CuBIC* et BIC aux alentours de la dernière valeur maximale

Compound TCP

Compound TCP [42], présenté par Microsoft en 2006, vise à combiner l'approche de TCP *Vegas* (basée sur le délai) à la fenêtre AIMD de TCP Reno.

Une nouvelle fenêtre de congestion globale est définie comme la somme de la fenêtre AIMD et d'une fenêtre dont les variations sont basées sur le délai. Lorsque des temps d'aller-retour faibles sont observés sur le réseau, la fenêtre basée sur le délai grandit, ce qui permet d'atteindre rapidement la capacité du lien. Lorsque le délai augmente, donc qu'une file d'attente se remplit en cœur de réseau, la fenêtre basée sur le délai décroît, tandis que la fenêtre AIMD continue de grandir, comme représenté Figure 2-6 [42].

TCP *Compound* permet ainsi d'atteindre rapidement le produit délai-bande passante du lien, tout en modérant la vitesse de transmission lorsqu'une congestion est détectée.

Bien qu'implanté sur un large parc de machines (toutes les versions de Windows depuis 2007), cette variante n'est activée par défaut que sous Windows 2008, qui ne représente pas plus de 33% des serveurs, et moins de 1% des clients web.

Autres Variantes

Au contraire des variantes présentées ci-dessus, d'autres algorithmes ont été conçus pour des applications très spécifiques telles que TCP *Hybla* pour les communications par satellite (fort délai, faible débit) [43], TCP *Westwood* et son successeur *Westwood+* pour les liens sans fils [44], et *HSTCP (High Speed TCP)* pour les liens à fort produit délai bande passante [45].

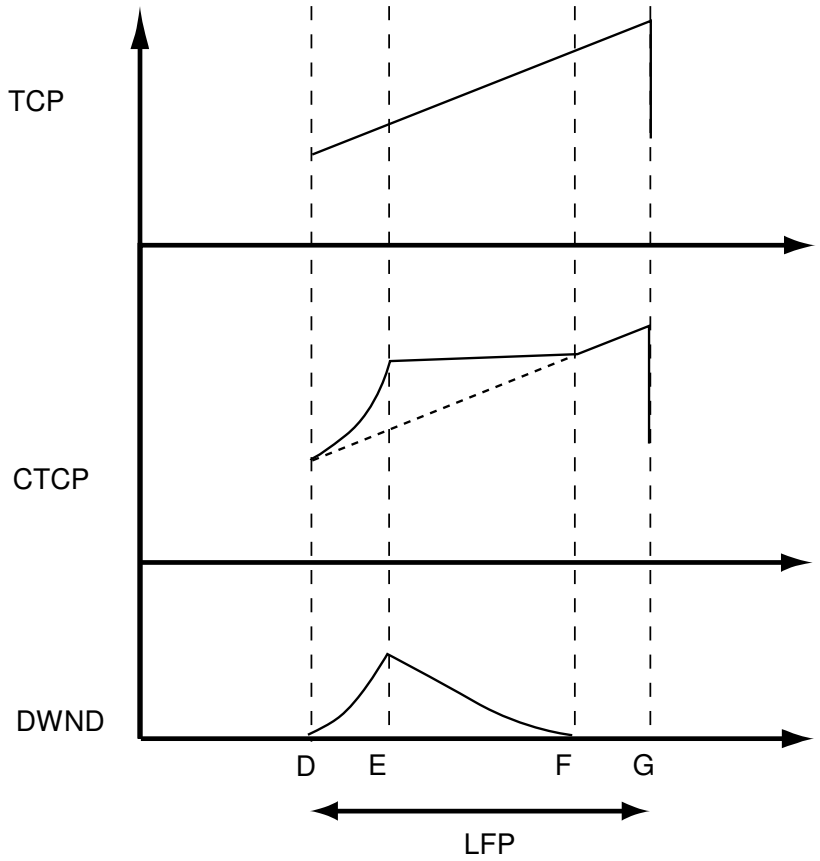


FIGURE 2-6: Fenêtre de congestion, fenêtre basée sur le délai et fenêtre totale de *Compound* TCP

2.2.3 En cœur de réseau

En sus des protocoles de transport, le contrôle de congestion est fortement dépendant de la gestion du trafic dans les routeurs en cœur de réseau. Adapter la politique de gestion des files d'attente reste la manière la plus efficace d'optimiser l'utilisation du lien, au prix d'une complexité de déploiement bien plus élevée.

Dimensionner une file FIFO : plusieurs règles empiriques

La majorité des files d'attente suivent la politique dite du "Premier Arrivé, Premier Servi" ou FIFO (*First In, First Out*). De nombreuses études ont porté sur le problème du dimensionnement d'un tampon suivant ce type de politique. La première règle explicite est attribuée à Curtis Villamizar et Cheng Song en 1994 [46].

Dans le cas d'une unique connexion TCP dans un goulot d'étranglement, la fenêtre de congestion oscille entre deux valeurs : une valeur maximale atteinte lorsque la file d'attente déborde, et cette valeur maximale divisée par deux à la suite d'une perte. Afin de maximiser l'occupation du réseau, la

fenêtre doit à tout moment être supérieure à la valeur du produit délai-bande passante du lien. Ainsi, $W_{max}/2 \geq C*RTT$, et $W_{max} \geq C*RTT+Q$, et donc $Q \geq C*RTT$. Toutefois, une valeur supérieure au produit délai-bande passante ne fait qu'entraîner des latences indésirables. En conséquence, la taille du tampon est régie par la règle suivante, dite *rule of thumb* :

$$Q = C * RTT \tag{2.2}$$

Selon les auteurs, un utilisateur ne peut déterminer le nombre d'applications partageant le même lien. La taille du tampon doit donc être fixée à cette valeur pour une utilisation optimale du réseau. Toutefois, pour les liens présentant un fort produit délai-bande passante, les délais dus aux tampons peuvent toujours devenir importants.

Dans un article appelé *Sizing Router Buffers* [47], Appenzeller *et al.* considèrent le cas d'un mélange de flux courts et longs partageant le tampon d'un routeur. Les connexions courtes ont tendance à envoyer de courtes rafales de paquets, et contribuent donc peu au remplissage de la file sur le long terme. En revanche, le nombre de longs flux varie peu au cours du temps. Si un nombre moyen N de connexions longues partagent un tampon, l'utilisation maximale du lien est obtenue pour un tampon de taille :

$$Q = C * RTT/sqrt(N) \tag{2.3}$$

D'autres études montrent que la taille de tampon optimale peut être encore réduite, entre 10 [48] et 63% [49] de l'équation précédente, certaines [50] allant jusqu'à affirmer qu'un tampon de 20 paquets représente une solution universelle et suffisante en cœur de réseau.

Gestion active des files d'attente

Étant donné la complexité du dimensionnement des tampons, d'autres mécanismes de gestion active des files d'attente (*Active Queue Management* – AQM) ont été développés, dans l'objectif de réduire les délais, diminuer la variance du RTT et améliorer les débits.

RED (*Random Early Discard* [51]), développé par Sally Floyd et Van Jacobson, est une des premières politiques d'AQM. Lorsque l'occupation de la file atteint un premier palier, tout paquet arrivant dans le tampon a une probabilité fixe d'être supprimé du tampon. Après un second palier (fixé à la taille maximale du tampon), les paquets sont automatiquement retirés du réseau. Cet algorithme permet d'apporter plus d'équité aux connexions qui envoient des rafales de paquets en répartissant les pertes entre tous les flots. En effet, une rafale de paquets (qui peut typiquement provenir d'une connexion en *slow start*) est plus susceptible d'entraîner une rafale de pertes en cas de congestion, tandis qu'une

connexion plus “fluide” (en *congestion avoidance*) ne subira qu'une unique perte.

Cette politique permet également de désynchroniser les pertes entre les divers flux de données. La synchronisation des pertes est en effet responsable d'oscillations de l'occupation des tampons, ce qui mène à une forte variance des délais.

Malgré ces avantages, le nombre de paramètres à adapter aux conditions du réseau rend RED plus complexe à déployer et configurer que le simple dimensionnement d'une file FIFO. Il faut en effet désormais dimensionner la file, déterminer le premier palier et la probabilité de pertes pour chaque réseau.

Adaptive RED (ARED) [52] apporte une première réponse à ce problème. L'occupation moyenne de la file est utilisée pour déduire la probabilité de perte. De ce fait, le nombre de paramètres de l'algorithme est réduit, bien que les deux paliers restent à définir.

BLUE [53] est une autre variante plus récente et sans paramètres. Lorsque le tampon déborde, la probabilité de perte de paquets augmente légèrement. Cette probabilité diminue lorsque la file est vide, et converge ainsi vers une valeur qui garantit une utilisation optimale du lien.

Files d'attente et équité

Si TCP et les politiques de gestion des files d'attente créent une certaine équité entre les connexions, celle-ci n'est pas toujours respectée. Ainsi, les connexions ayant un fort RTT ont tendance à s'effacer par rapport aux flux présentant une faible latence. De même, une perte est plus dommageable à une connexion naissante qu'à un flot long. Enfin, le réseau n'est pas à l'abri d'un hôte se comportant de manière nuisible pour les autres connexions. Ces problématiques ont été soulevées par John Nagle dans la RFC 970 [54] en 1985.

Afin de répondre à ces phénomènes et maximiser l'équité, des politiques de gestion de files d'attente par flux (*Fair Queuing* – FQ) ont été développées en parallèle de la gestion active des files d'attente.

Au lieu d'avoir un unique tampon partagé par tous les flux de données, le trafic est séparé entre de multiples files d'attente (idéalement une par connexion TCP). Ces tampons sont ensuite servis selon diverses politiques conçues pour approximer le modèle du *Processor Sharing* [55], dans lequel tous les flux sont servis simultanément, et reçoivent une part égale de la bande passante disponible.

La politique la plus connue est l'algorithme *Round Robin* [56]. Les flux de paquets sont répartis entre plusieurs files d'attente, et chaque file contenant des données est servie à tour de rôle. Bien qu'équitable en termes de débits en paquets, l'algorithme favorise les flux de grands paquets, au détriment des connexions qui présentent un MTU (*Maximum Transmission Unit*) faible. Cette constatation mène à la création en 1989 d'une version basée sur le nombre d'octets servis au lieu du nombre

de paquets [57]. Une autre variante, *Weighted Round Robin* [58], permet de réaliser des opérations de QOS (*Quality Of Service*) sur les flux de données en leur assignant des quantités différentes de service selon leur origine.

Une implantation de l'algorithme de *Fair Queuing* tel qu'imaginé par John Nagle est *Stochastic Fair Queuing* (SFQ), développé pour le noyau Linux. Les informations de la connexion (ip/port source et destination) sont hachées et une file est attribuée par empreinte. S'il est vrai que les chances de collision entre empreintes sont faibles, elles entraînent immédiatement une perte de l'équité souhaitée.

2.3 Le problème des transferts de taille finie

Jusqu'ici, toutes les solutions décrites ont été pensées pour des connexions continues de taille infinie. Néanmoins, les études montrent que si les flux longs (considérés ici comme infinis) représentent effectivement plus de 80% du trafic en volume, on observe un nombre bien supérieur de courts transferts. En d'autres termes, 20% des connexions génèrent 80% du trafic réseau.

De nouveaux problèmes apparaissent lorsque les connexions qui n'ont pas atteint le régime stable sont considérées. Ces phénomènes touchent plus particulièrement les flux courts ou intermittents. De plus, considérer une connexion de taille finie revient à prendre en compte son début et sa fin.

De ce fait, les premiers et derniers paquets d'un épisode de transmission doivent être traités avec prudence, car leur perte résulte dans la majorité des cas en une expiration du *timer* de retransmission, avec pour conséquence des temps de retransmission de l'ordre de plusieurs secondes pour des durées de transmission ordinaire de l'ordre de la centaine de millisecondes.

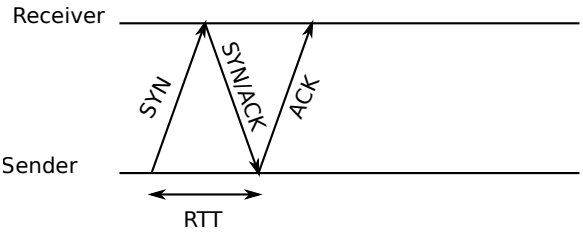
De plus, comme nous avons pu le voir dans la section précédente, les connexions naissantes ont tendance à être traitées de manière inéquitable en comparaison des flux déjà établis. Elles n'ont en effet pas encore exploré toute la bande passante disponible et envoient les données par rafales ce qui les rend plus susceptibles de subir de multiples pertes. Or, ces connexions particulièrement fragiles sont majoritaires sur le réseau et représentent très souvent du trafic interactif comme la navigation web, qui requiert une faible latence. Cette situation est souvent évoquée sous le nom de "Problème des souris et des Éléphants" [59].

2.3.1 Établissement de la connexion

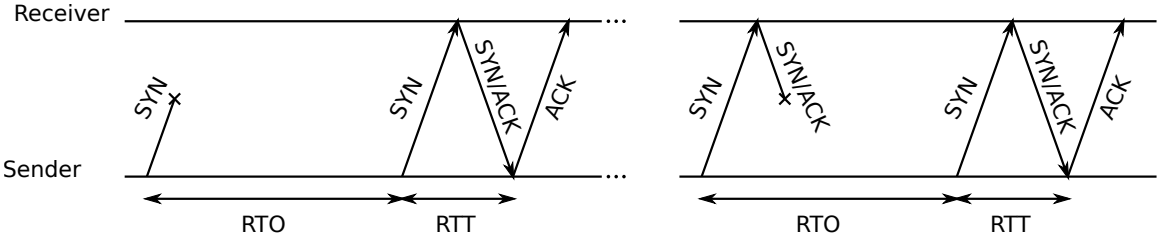
TCP établit ses connexions par une triple poignée de mains (*three way handshake*). Pour initier la connexion, l'émetteur envoie un paquet avec le *flag* SYN activé. Le récepteur accepte la connexion et transmet à son tour un segment contenant les *flags* SYN et ACK pour signifier son accord. La connexion est considérée comme étant établie lorsque l'émetteur accuse réception de ce dernier

segment.

Lorsqu'un paquet est perdu durant cette phase, il ne peut être retransmis qu'après la longue expiration d'un *timer*. En effet, au démarrage, les connexions n'ont pas d'estimation du temps d'aller-retour entre le client et le serveur. Elles utilisent donc le *Retransmission timeout* (RTO) dont les valeurs initiales sont fixées entre 1 et 3 secondes. Dans le cas des connexions les plus courtes, ces valeurs peuvent être plusieurs ordres de grandeur au-dessus du temps de transmission des données.



(a) La poignée de main en trois temps



(b) Récupération d'une perte de SYN ou de SYN/ACK

FIGURE 2-7: Établissement de connexion

Ces mécanismes sont illustrés Figure 2-7.

Les délais causés par la perte d'un segment lors de l'établissement de la connexion sont souvent passés sous silence dans la littérature y compris dans des articles récents [60, 61]. Certains auteurs observent que la probabilité de perte des paquets contenant le *flag* SYN peut être supérieure à celle des paquets de données [62], toutefois sans explorer l'impact de cette différence. Ciullo *et al.* [63] ont réalisé que plus de 70% des paquets perdus étaient récupérés après expiration du RTO suite à une étude de la distribution des temps de réponse des connexions. Ils proposent alors deux contributions pour éviter les longs délais de récupérations suite à la perte du dernier paquet d'un flux.

Anelli *et al.* ont activement étudié la protection des SYN pour améliorer les temps de réponse des connexions. Pour ce faire, ils introduisent *REDFavor*, un mécanisme basé sur RED avec un traitement particulier des SYN [64].

Une dernière solution proposée dans la littérature consiste en la retransmission agressive de SYNs pour anéantir l'impact des *timeout*s lors de leur perte [65, 66, 67].

En général, à la suite de l'établissement de la connexion, la phase de *slow start* est caractérisée par l'envoi de rafales de paquets. Ces rafales croissent de manière exponentielles, et entraînent des trains

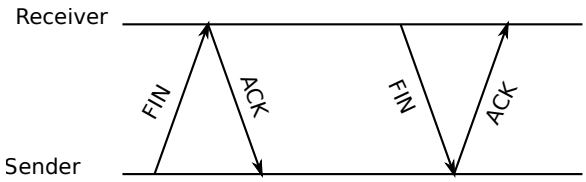


FIGURE 2-8: Fin de connexion

de pertes lorsqu'un tampon déborde en cœur de réseau. Ce comportement est à l'origine de nombreux problèmes parmi lesquels des épisodes de congestions brutaux, et des temps de retransmission élevés pour la connexion qui subit un train de pertes. Plusieurs solutions ont été proposées afin de garantir une sortie plus souple du *slow start* parmi lesquelles *Limited Slow-Start* [68] qui diminue progressivement l'intensité des rafales lorsque le *slow start threshold* est dépassé ou plus récemment *Initial Spreading* [69] qui consiste à étaler la transmission d'une rafale au cours d'un RTT.

2.3.2 Fin de connexion

Une connexion TCP se termine lorsqu'un des deux hôtes le décide. Il transmet alors un segment contenant le *flag* FIN. Le récepteur accuse réception de ce segment et la connexion est considérée à moitié fermée. En effet, les deux hôtes n'ont convenu d'interrompre le trafic que dans une seule direction. Dans l'autre sens, des données peuvent être émises jusqu'à l'envoi d'un paquet FIN par le deuxième hôte. Ce segment devra à son tour être acquitté pour que la connexion soit considérée comme fermée (*cf.* Figure 2-8).

De manière similaire à l'établissement de la connexion, un paquet perdu lors de cette phase résulte automatiquement en un *timeout*. Si, à ce stade, la valeur du *timer* de retransmission a en général été mise à jour avec les RTTs observés, elle reste dans la majorité des implantations limitée par une valeur minimale entre 200 ms (Linux) et 1 s (FreeBSD et Microsoft Windows) et est très dépendante de la variance des délais observés.

Deux mécanismes ont été proposés pour réduire l'impact des pertes en fin de connexion : *Early Retransmit* et *Tail Loss Probe*. Ces stratégies protègent également les derniers paquets envoyés dans le cas de connexions intermittentes. Des techniques à base de codes correcteurs d'erreurs peuvent également être employées dans le but de protéger l'intégralité de la connexion.

Early Retransmit [4]

Lorsque la fenêtre de congestion n'est pas assez grande (faible produit délai bande passante), ne peut être remplie (transmission intermittente) ou de manière plus générale, lorsqu'une connexion ne génère pas assez de données pour recevoir les 3 acquittements dupliqués nécessaires à la retransmission, les

techniques traditionnelles de récupération de pertes ne sont plus fonctionnelles. La retransmission ne peut plus alors qu'être due à l'expiration d'un *timer*. L'algorithme *Early Retransmit* diminue le nombre d'acquittements dupliqués nécessaires pour déclencher une retransmission et ainsi éviter les *timeout*s.

Le nombre d'acquittements dupliqués ER_{thresh} nécessaires pour la retransmission est déterminé par la formule suivante : $ER_{thresh} = oseg - 1$ (toutes les mesures données en paquets), où $oseg$ est le nombre de segments en vol (*i.e.* non acquittés) sur le réseau. Ainsi, un seul acquittement dupliqué est nécessaire pour récupérer de pertes en fin de connexions, ce qui permet de protéger efficacement jusqu'à l'avant-dernier segment d'un épisode de transmission.

Tail loss probe [5]

Early Retransmit permet de protéger essentiellement les connexions naissantes ou intermittentes. Toutefois, les pertes en queue sont toujours résolues via un *timeout*. *Tail Loss Probe* répond à ce problème en opérant uniquement au niveau de l'émetteur. TLP définit un *Probe timeout* (PTO) moins élevé que le RTO original (2 RTT). Lorsque le PTO expire, un nouveau segment est immédiatement transmis. Si aucun nouveau paquet n'est disponible, le dernier segment est réexpédié. Ce mécanisme permet soit de générer suffisamment de trafic pour permettre l'arrivée des 3 accusés de réception dupliqués et ainsi retransmettre les données perdues en fin de connexion, soit de récupérer directement la perte lorsqu'elle concerne le dernier paquet d'une rafale.

Systèmes à codes correcteurs d'erreurs

L'utilisation de codes correcteurs d'erreur de type Reed-Solomon permet de représenter chaque paquet comme une combinaison linéaire des paquets précédents. Ainsi, même si le dernier segment d'une transmission est perdu, il est possible de le récupérer en utilisant les paquets précédents.

TCP-NC[70] se base sur cette idée. Au lieu d'utiliser les pertes afin de réduire la fenêtre de congestion, cet algorithme se base sur TCP *Vegas*, et adapte le rythme de transmission en fonction des latences mesurées.

2.3.3 Autres protocoles de transport

TCP coexiste avec d'autres protocoles de transport conçus pour des applications spécifiques parmi lesquelles : applications contraintes en délai, transmission parallèle, sécurité, ou encore réservation de ressources. Dans le cas des connexions de taille finie, des protocoles ont également été développés pour favoriser les plus petits transferts face aux connexions longues.

Micro Transport Protocol – μTP [6]

μTP a été pensé comme un protocole de transfert de données en arrière-plan, conçu pour ralentir automatiquement le rythme de la transmission lorsque celle-ci interfère avec d'autres applications interactives, ce qui le rend particulièrement adapté aux transferts de large volumes de données pour lesquels l'utilisateur n'a pas d'exigences en termes de délais.

Tout comme TCP *Vegas*, μTP se base sur le délai pour déterminer la manière de partager un tampon avec d'autres connexions. L'algorithme vise un délai maximal de 100 ms, et pondère donc l'augmentation de la fenêtre en fonction des valeurs mesurées. Ainsi, une fenêtre de valeur *max_ window*, avec *off_ data* de données en vol augmente d'une quantité *scaled_ gain* pour un délai mesuré *off_ target* :

```
delay_factor = off_target / CCONTROL_TARGET;
window_factor = outstanding_packet / max_window;
scaled_gain = MAX_CWND_INCREASE_PACKETS_PER_RTT * delay_factor * window_factor;
```

Ce protocole a été acquis par Bit Torrent *Inc.* en 2006, et intégré au sein du protocole BitTorrent car particulièrement adapté aux transferts de gros fichiers sans impact sur le trafic interactif.

Aspera Fast Performance Transport Software – FASP [7]

FASP est une initiative industrielle qui vise à procurer une alternative à TCP pour transférer des fichiers sur des réseaux IP. Son objectif principal est de décorréler le débit de la latence réseau et d'être extrêmement résistant aux pertes de paquets.

Selon sa présentation commerciale, ce produit présente des fonctionnalités additionnelles, telles que la découverte de la bande passante disponible, l'optimisation du transfert de petits fichiers, et l'agrégation de connexion, en plus de l'équité procurée face à d'autres protocoles comme TCP. Il est actuellement proposé comme service pour Amazon Cloud.

2.3.4 Flow Queuing

Si les algorithmes de *Fair Queuing* décrits en section 2.2.3 procurent plus d'équité pour les petites connexions en comparaison des tampons FIFO, d'autres stratégies de files d'attente par flux (*Flow Queuing*) ont été développées dans le but de favoriser l'efficacité, parfois au détriment de l'équité. Ces algorithmes favorisent donc les connexions courtes, plus fragiles, par rapport aux transferts volumineux. Nous pouvons notamment citer *Least Attained Service (LAS)* et certains de ses dérivés LARS [71], Run2C [72] ou EFD [73].

Ces politiques de gestion de files d'attente présentent des caractéristiques intéressantes : non seulement elles protègent l'établissement de la connexion en assignant une nouvelle file pour chaque nouveau flux de données mais elles favorisent aussi les connexions qui n'ont pas quitté le *slow start*, et n'ont donc pas encore atteint la capacité du lien.

2.4 Combiner débits élevés à une faible latence

Dans les réseaux à commutation de paquets, les segments qui arrivent sur un lien sont stockés dans des tampons (qui suivent en général une politique *First in - First out*), en attente d'être traités. Dans ce contexte, le problème principal reste le dimensionnement du tampon déjà évoqué section 2.2.3.

Si le tampon est trop petit, de nombreux paquets seront perdus, et le débit des connexions risque de s'effondrer (ce phénomène arrive lorsque la probabilité de pertes de paquet dépasse 5 à 10%). La tentation devient alors forte de surdimensionner les tampons pour empêcher ces incidents. En pratique, ce comportement est au mieux inutile, mais se révèle souvent contre-productif.

En effet, TCP se base sur les pertes de paquets pour adapter le rythme de la transmission : ce dernier accélère jusqu'à une perte de paquet, à la suite de laquelle l'envoi de donnée est significativement ralenti.

Lorsqu'un tampon est surdimensionné, les paquets passent un temps important dans la file d'attente, ce qui cause des latences importantes et une forte variance des délais (*jitter*) qui peuvent atteindre plusieurs secondes. De plus, les pertes de paquets sont moins fréquentes, ce qui empêche TCP de ralentir lorsque le lien est saturé. TCP continue ainsi de nourrir la congestion, avec de graves répercussions sur toute application interactive (jeu en ligne, voix sur IP, requête DNS ou encore simple trafic web).

Dans cette situation, le tampon est dit *bloated* (ballonné), et le phénomène appelé *bufferbloat*.

Ce problème a été évoqué très tôt dans l'histoire des réseaux : la RFC 970 [54], publiée en 1985 (soit seulement 2 ans après le déploiement de la pile TCP/IP) exprime les premières inquiétudes à propos de tailles de tampons excessives, bien avant que le problème ne devienne significatif.

Dans un article intitulé *It's the Latency, Stupid* [74], publié en 1996, Chesire évoque la différence entre la bande passante – appelée à grandir grâce aux évolutions technologiques – et la latence, beaucoup plus complexe à éviter. Un certain nombre de pistes sont proposées pour réduire la latence, sachant qu'"une fois que vous avez une forte latence, vous devez faire avec" [5]. Toutefois, les solutions présentées ne se focalisent que sur des règles de bonne conduite à adopter lors de développement d'application

5. "once you have bad latency, you're stuck with it"

réseau (compression, cache, réduire la quantité de données envoyées etc.) sans s'attaquer au cœur du problème.

En ce qui concerne la recherche en réseaux informatiques, les premières interrogations ont commencé à se poser en 2009 [75] et un groupe de travail sur le sujet a été fondé en 2011, à la suite de l'article fondateur *bufferbloat : Dark Buffers on the Internet* [76].

2.4.1 Solutions en cœur de réseau

Le *bufferbloat* étant causé par la présence de tampons surdimensionnés dans le réseau, sa solution immédiate réside dans un dimensionnement correct des files d'attente des routeurs, qui peut impliquer de changer non seulement leur taille, mais aussi leur structure : par exemple, un tampon sur le lien montant de 3 paquets entraînera de faibles débits, car un mélange de données TCP et d'accusés de réceptions le fera déborder prématurément. En revanche, limiter la taille des tampons à un nombre d'octets plutôt que de paquets (pour reprendre l'exemple précédent, 3 paquets représentent approximativement 5000 B) réduit la latence tout en améliorant l'utilisation respective des liens montants et descendants. De plus, dimensionner un tampon nécessite de connaître à tout instant la capacité du lien, ce qui exclut d'emblée les liens dont la bande passante varie au cours du temps comme les liens cellulaires.

CoDel (*Controlled Delay*), développé par Kathleen Nichols et Van Jacobson utilise le minimum local des temps de séjour d'un paquet dans une file sur un intervalle de temps donné. Si le temps de séjour dépasse une valeur cible durant cet intervalle, le paquet est supprimé, et l'intervalle réduit. Cet algorithme vise à être déployé sans configuration préalable du fait de sa nature adaptative, et élimine ainsi l'étape de dimensionnement inhérente aux files FIFOs ou les paramètres additionnels de RED. Sa variante FQ CoDel combine les avantage des algorithmes de *Fair Queuing* aux faibles délais présentés par CoDel.

Proportional Integral Controler (PIE) est de conception plus légère que CoDel, dans le but de réduire le coût de l'algorithme tant en termes d'implantation que d'utilisation des ressources. Cet algorithme calcule la probabilité de supprimer un paquet en utilisant le délai instantané de la file. Tout comme CoDel, PIE est conçu pour être déployé sans configuration.

Grâce à ces deux stratégies de gestion des tampons, les paquets subissent des latences moins importantes, tout en permettant à de courtes rafales de paquets de traverser le réseau sans rencontrer de pertes. En effet, seuls les flux qui remplissent la file sur le long terme devraient être pénalisés, tandis que des épisodes ponctuels de remplissage de tampon sont tolérés.

2.4.2 Protocoles de transport

Si la majorité des latences excessives trouvent leur origine dans les routeurs en cœur de réseau, les algorithmes de contrôle de congestion de bout-en-bout peuvent participer à une amélioration des conditions en régulant la manière dont les données sont envoyées.

Modifications de la pile TCP

TCP *Vegas* (*cf.* section 2.2.2) essaie de par sa conception de garder l'occupation de la file aussi basse que possible en surveillant constamment le RTT d'une connexion et en s'adaptant en conséquence.

Le noyau Linux intègre plusieurs modifications pour contrer le *bufferbloat* [77] en proposant une combinaison de *Fair Queuing*, de modifications de la pile TCP et de configurations recommandées pour limiter la quantité de données stockées dans le réseau.

En parallèle de ces modifications de la pile TCP, d'autres protocoles de transport ont été développés dans le but de limiter les délais.

Datagram Congestion Control Protocol – DCCP [8]

DCCP est un protocole qui organise les données sous forme de datagrammes de manière similaire à UDP. Tout comme TCP, il apporte de nombreuses fonctionnalités telles qu'un établissement et une fin de connexion fiables, des notifications de congestion explicites (via ECN), un contrôle de congestion, et la négociation de fonctionnalités supplémentaires. En revanche, conçu pour des applications contraintes en délai et/ou en temps réel, DCCP n'implante pas de mécanisme de récupération de pertes, la réception de nouvelles données étant préférée à la retransmission d'ancien paquets. En conséquence, la fiabilité du transfert de données a été sacrifiée pour une amélioration des délais.

Quick UDP Internet Connexions - QUIC [9]

QUIC vise à remplacer TCP pour des applications web orientées connexion. Il procure les mêmes propriétés que TCP, avec certaines fonctionnalités additionnelles. Tout d'abord, les connexions peuvent être multiplexées pour profiter de la bande passante de plusieurs liens tout en procurant une fiabilité renforcée.

Le contrôle de congestion de TCP peut être nocif aux connexions multiplexées, c'est pourquoi QUIC utilise de nombreuses techniques, parmi lesquelles l'espacement de paquets selon une estimation de bande passante, une redondance des messages importants, ou encore l'utilisation de codes correcteurs d'erreurs. Une couche de sécurité similaire à SSL/TLS a été intégrée au protocole.

QUIC est implanté sous Google Chrome et Opéra, mais n'est activé par défaut que sous Google Chrome (63.3% des navigateurs en juillet 2015 [78]).

SPDY [10] et HTTP /2 [11]

Bien que conçus comme des protocoles applicatifs, au-dessus de TCP, SPDY et HTTP /2 luttent contre les forts délais rencontrés au cours de la navigation web à travers de multiples techniques telles que la compression, la priorisation, et le multiplexage.

SPDY est désormais obsolète, et la plupart de ses fonctionnalités ont été intégrées dans HTTP /2. Ces deux protocoles démontrent que si la latence est un problème de cœur de réseau, qui touche majoritairement les couches réseau et transport, les optimisations ne sont efficaces qu'en combinaison avec une gestion soigneuse des ressources au niveau des couches supérieures (*cf. It's the Latency, Stupid* [74]).

2.5 Exploiter de multiples chemins

Une nouvelle tendance dans les protocoles de transports réside dans l'exploitation de multiples chemins dans le but de procurer des débits plus élevés, de la redondance, et même résoudre les problèmes de *handover* dans les scénarios sans fil.

2.5.1 Multipath TCP [12]

Multipath TCP est conçu comme une couche additionnelle entre TCP et l'application. Il permet de regrouper de multiples connexions TCP dans une méta-connexion et améliore l'utilisation des ressources, tout en procurant de la redondance à la transmission. Bien qu'encore fortement expérimental, le protocole est déployé dans des applications à forte audience [6].

2.5.2 Stream Control Transmission Protocol – SCTP [13]

SCTP a été pensé pour procurer la fiabilité de UDP sur un protocole orienté message comme TCP. De manière similaire à *Multipath TCP*, il permet d'agréger plusieurs connexions au sein d'une même association SCTP. Cet algorithme fait partie de la famille de protocoles SGITRAN et est actuellement implanté sur une grande variété de systèmes [80], bien que peu utilisé.

6. Comme a pu le faire Apple pour son application Siri [79]

2.6 Conclusion et problématique

Au cours de ce chapitre, nous avons pu entrevoir les évolutions et défis que présente le travail sur un mélange de technologies anciennes (à l'échelle de l'informatique) et récentes dans des applications en constante évolution, et toujours plus exigeantes en ressources.

De nos jours, la principale problématique reste la maximisation des débits tout en procurant des délais réduits. Au cours de ce manuscrit, nous nous concentrerons sur cette problématique appliquée aux liens asymétriques, fortement présents aux niveaux des raccordements finaux (lignes ADSL, réseaux cellulaires). L'analyse se portera plus particulièrement sur la compréhension des phénomènes à l'œuvre et l'apport de solutions simples et légères aux problématiques rencontrées.

Chapitre 3

Simulations et Expériences TCP : réalisme et reproductibilité

Of course, it would all require split second robot timing. That's where I come in. You see, I own a watch.

Bender – Futurama

Introduction

La reproductibilité des expériences est la pierre angulaire de la recherche scientifique. En physique, biologie ou autres, cette reproductibilité peut être difficile à cause de besoins en équipements spécifiques, la variabilité des données selon les populations, etc. En revanche, la reproduction des résultats est plus accessible en informatique de par les outils employés. Dans un grand nombre de cas (en particulier les expériences sur simulateurs), la procédure à suivre ne devrait pas dépasser plus de deux opérations : récupérer l'expérience sur un dépôt distant, puis lancer ladite expérience. Les expériences sur machines réelles (en opposition aux simulations), quant à elles, soulèvent un autre

problème : il doit être possible de réunir les conditions matérielles et logicielles de la procédure initiale avant de pouvoir la répéter. À l'exception de cette contrainte, réitérer une expérience devrait être aussi simple que dans le cas d'un simulateur.

Il est pourtant très rare d'obtenir suffisamment d'informations dans un article afin de pouvoir vérifier les résultats obtenus. Les données sont très souvent incomplètes, parfois au point de ne pas indiquer la topologie exacte ou le modèle de trafic utilisé. Une étude [81] effectuée sur 601 articles en informatique montre que seulement 21% des articles présentent des expériences aisément reproductibles (code disponible et moins de 30 minutes nécessaires à la mise en place).

Au-delà de la possibilité de répéter facilement les expériences, se pose la question de la validité des résultats : si un simulateur permet de tester rapidement un grand nombre de configurations de manière reproductible, il ne fait qu'imiter une partie de la réalité, et les résultats obtenus peuvent être fortement influencés par les choix d'implantation, en particulier le soin apporté au réalisme des environnements simulés.

Ce chapitre se divise en deux parties : tout d'abord, les différentes solutions sont présentées et comparées les unes aux autres. La deuxième partie traite des choix et compromis effectués dans le cadre de la réalisation ainsi que la distribution d'un banc d'essai. Notons que nous ne nous intéresserons qu'au cas d'expériences réalisées sur les couches réseau et transport du modèle OSI, dans le cas de connexions filaires (Ethernet).

3.1 Expérimentations TCP et choix techniques

TCP est un des plus anciens protocoles réseaux encore utilisé. De ce fait, un grand nombres d'outils et méthodes — parfois diamétralement opposés — ont été développés au fil des années. L'expérimentateur se retrouve donc confronté à de nombreux choix techniques. Cette variété soulève de nombreuses interrogations : quels outils employer ? Dans quel but ? Quelles topologies et quels modèle de trafic utiliser ?

À ces interrogations s'ajoutent en trame de fond le questionnement suivant : *comment réaliser de manière exhaustive des expériences aisément reproductibles ?*

Cette section s'efforce de répondre à toutes ces questions, en pesant les différentes possibilités offertes par chaque solution.

3.1.1 Simulateur, Machine réelle ou virtuelle ?

Un simulateur permet de tester un grand nombre de topologies complexes, dans un temps potentiellement raisonnable. Les expériences sur simulateur présentent deux avantages majeurs : tout d'abord, une seule machine est nécessaire pour effectuer l'intégralité des expériences. De plus, le simulateur procure un environnement fixe, ce qui facilite la reproductibilité des procédures. Ainsi, une expérience n'est constituée que d'un ensemble de fichiers à ajouter à une installation existante du simulateur. La distribution peut donc se faire via un dépôt quelconque, voire une simple archive à décompresser, et peut inclure le simulateur lui-même dans le cas où la licence le permet.

Deux simulateurs génériques se détachent :

— **ns (Network Simulator)** est un des premiers simulateurs réseau, développé dès 1995. À l'heure actuelle, deux versions développées par des équipes indépendantes cohabitent. Ns 3 est une réécriture complète du simulateur, non compatible avec le ns 2. Toutefois, l'intégralité des protocoles supportés par ns 2 n'ayant pas été portés sur la dernière version, les deux simulateurs continuent d'être utilisés en parallèle. Ce simulateur peut s'interfacer avec la pile réseau du système d'exploitation, pour des expériences proches de la réalité. Toutefois, cette fonctionnalité est complexe à mettre en œuvre, et peut donner des résultats différents selon la version du noyau (voir partie suivante).

— **QualNnet** est un simulateur commercial, présenté comme spécialisé dans la modélisation de la couche physique. Les imperfections des liens, en particulier sans fils sont ainsi finement décrites. Le code de la couche réseau de ce simulateur dérive de la pile TCP/IP de FreeBSD, ce qui donne l'assurance de résultats proches du standard. De plus, son architecture permet d'exploiter plusieurs processeurs, voire un ensemble de machines pour simuler efficacement les topologies les plus complexes. Toutefois, son coût ainsi que ses conditions d'utilisations (serveur de licence) peuvent se révéler prohibitifs.

Ces simulateurs peuvent au premier abord sembler une solution idéale pour des expériences reproductible. Néanmoins, ils présentent un inconvénient majeur : malgré les efforts portés sur le réalisme du système, les topologies simulées restent souvent trop parfaites : liens parfaitement symétriques, traitement instantané des paquets par les routeurs, hôtes considérés comme possédant une mémoire et un CPU infinis, etc. Ainsi, si les expériences sur simulateur peuvent donner un bon aperçu des possibilités d'un système, elles ne sont pas (toujours) suffisantes pour valider un principe [1].

Il est donc nécessaire d'expérimenter sur des machines réelles. Celles-ci présentent en effet l'avantage de pouvoir effectuer des tests dans des conditions proches de celles de l'utilisateur final. Le fait

1. comme a pu amèrement le constater le rédacteur de ce manuscrit en début de thèse : une solution basée sur les mesures de RTT obtenait de très bons résultats dans les conditions de simulations. Toutefois, il s'est avéré que la solution exploitait la faible variance des RTT et des temps de traitements présentée par le simulateur, et les mécanismes ne fonctionnaient plus lors d'expérimentations sur machines réelles.

d'utiliser des machines réelles permet d'obtenir des résultats très différents, notamment dans le cas de solutions basées sur les délais. Ces dernières peuvent en effet souffrir des variations introduites par les machines, et présenter des performances considérablement amoindries par rapport à une expérience similaire sur simulateur.

Les machines réelles ont toutefois un inconvénient majeur : leur coût, qu'il soit financier, en espace de stockage ou en heures de travail. Le système doit en effet fonctionner dans un environnement isolé afin de minimiser les interférences produites par des éléments extérieurs, et ainsi garantir la reproductibilité. Ceci implique la mise en place de machines dédiées. De même, une expérience peut générer des traces réseaux de taille importante, ce qui implique de forts besoins en espace de stockage. À titre d'exemple, le banc d'essai décrit dans les parties suivantes immobilise 4 machines, pour des expériences générant des traces de l'ordre de la dizaine de Go. Enfin, permettre au plus grand nombre de reproduire les expériences implique d'écrire un moteur portable, robuste et extensible afin de nécessiter un minimum de manipulations. Cette tâche peut s'avérer laborieuse, d'autant plus que les limitations techniques peuvent empêcher la mise en place de certains raccourcis.

Toutefois, les résultats fournis par des machines réelles sont souvent plus réalistes que dans le cas de simulateurs, et représentent donc une étape indispensable à la validation de résultats.

Par rapport aux contraintes énoncées ci-dessus, les machines virtuelles deviennent attrayantes : celles-ci permettent en effet de déployer rapidement un environnement de tests, dans des conditions plus réalistes qu'un simulateur, tout en gardant un faible coût de mise en production. Cependant, de par leur mode de fonctionnement, ces machines sont dépendantes du système d'ordonnancement de l'hôte, ce qui fait que, par exemple, le trafic généré est transmis par rafales. Cette modification des dynamiques peut influencer les mesures, en particulier lors de l'étude des dynamiques des files d'attente dans le réseau. De plus, ajouter un deuxième système d'exploitation entre l'hôte et le réseau affecte les performances générales.

Ainsi, une validation sur banc d'essai est nécessaires, tandis que l'utilisation d'un simulateur est à restreindre à la validation de modèles mathématiques (pour lesquels les fluctuations introduites par les machines sont souvent négligées), ou pour des topologies plus complexes. L'utilisation de machines virtuelles est quant à elle à proscrire, car le trafic généré ne peut être considéré comme réaliste, et peut subir (ou causer) de sérieuses perturbations.

3.1.2 Banc d'essai

Si la mise en place d'une expérience en environnement simulé est relativement évidente, l'utilisation de machines réelles nécessite de faire certains choix et compromis selon les moyens disponibles, ainsi que l'utilisation qu'il en sera faite.

3.1.3 Choix d'une topologie

Les expériences en environnement réel ont un coût tout d'abord financier : chaque machine à intégrer dans le système en augmente le montant. C'est pourquoi la topologie choisie doit rester simple tout en permettant l'adjonction de nouvelles machines.

La topologie en haltère (*dumbbell*) se prête particulièrement à cet exercice. Un réseau minimal est en effet composé de trois machines : deux hôtes finaux et une passerelle. La passerelle peut, grâce à certains modules noyau (*dummynet* pour BSD, *netem* sous GNU/Linux, voir plus loin), émuler divers liens en fonction de la provenance des paquets. L'ajout de nouvelles machines à ce réseau se fait simplement, soit directement sur une des interfaces de la passerelle, soit à l'aide de commutateurs Ethernet. De plus, pour un nombre plus élevé de machines, la passerelle peut être remplacée par un routeur, ce qui accentue le réalisme de l'ensemble.

À ces trois machines de base peuvent être ajoutées deux autres composants : un serveur de stockage réseau afin de traiter toutes les traces directement, et une machine de contrôle pour superviser le système. Le système final est représenté Figure 3-1.

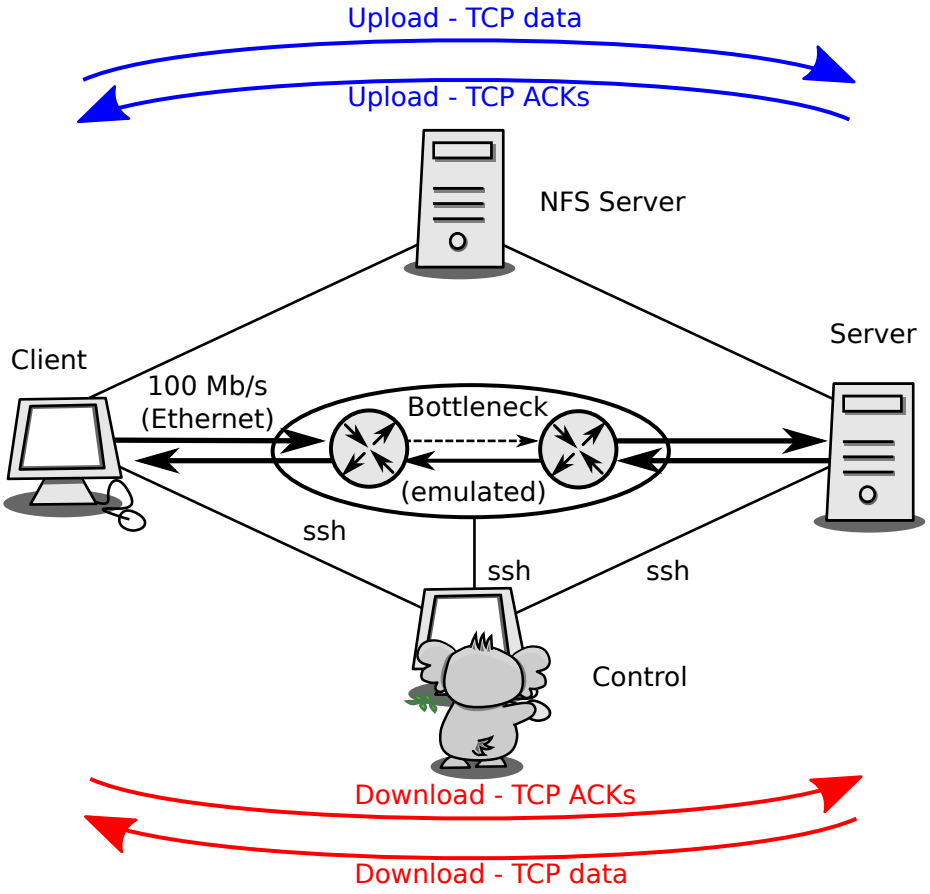


FIGURE 3-1: Topologie minimale d'expérimentations en conditions réelles

L'avantage principal de ce type de configuration est le fait de bénéficier d'un canal de contrôle séparé physiquement de l'environnement de test : les trafics ssh et nfs ne sont pas mélangés à celui de l'expérience. Le système procure donc la certitude que les phénomènes observés ne sont pas le fruit d'interférences avec du trafic extérieur, ce qui n'est pas le cas en utilisant de plus grandes plateformes d'expérimentations (type Grid5000 ou Planetlab). Ces dernières offrent souvent des machines interconnectées par un commutateur, à travers une unique interface. Trafic de test et trafic de contrôle sont donc mélangés, sans compter le fait que la plateforme est partagée entre de nombreux utilisateurs, qui émettent eux aussi leur propre trafic (ce qui peut être problématique lorsque toutes les machines de test ne sont pas connectées au même commutateur).

Enfin, si cette topologie peut paraitre simple — voire simpliste— il est toujours possible de valider les résultats par une expérience en conditions réelles, sur un réseau non contrôlé, comme c'est le cas pour les expériences présentées dans le chapitre 6. Il est toutefois à noter que ces dernières sont très complexes à reproduire, le choix même du routeur pouvant grandement influencer les mesures [82].

3.1.4 Choix d'un système d'exploitation

Une fois que la topologie est établie, un ou plusieurs systèmes d'exploitation doit être déployé sur le banc d'essai. Dans le cadre d'expériences sur la pile TCP/IP, ce choix est déterminant, car les détails de l'implantation influencent fortement les mesures.

Les hôtes

Pour les hôtes, trois systèmes d'exploitation se détachent : FreeBSD 9+, Linux 3.x et Windows 7+, chacun présentant ses avantages et inconvénients :

— **FreeBSD** propose une implantation directe du standard tel que décrit dans la RFC 5681 [33]. Chaque modification apportée à ce standard est aisément désactivable, ce qui en fait l'implantation de référence pour tester uniquement les algorithmes de contrôle de congestion. Depuis la version 9.0, la pile TCP a été réécrite afin de proposer plus de modularité, et les variantes *Cubic* et *Vegas* sont présentes

— **Windows** est le seul système proposant une implantation à jour de TCP *Compound*. De plus, cette version étant développée par Microsoft Research, leur implantation peut être considérée comme celle de référence de l'algorithme. À l'aide de logiciels supplémentaires tels que Cygwin, il est possible d'adapter des scripts Unix. Cependant, les licences auxquelles sont soumis les outils système rendent la redistribution des expériences plus complexe.

— **Linux** présente l'avantage de proposer de nombreuses variantes de TCP, des plus connues aux plus exotiques. En revanche, un certain nombre de modifications ne sont pas désactivables,

certaines étant encore à l'état de RFC "draft". Il est donc difficile de déterminer l'effet réel de l'algorithme de contrôle de congestion dans l'observation de certains phénomènes. De plus, l'implantation est en constante évolution, et une expérience est difficilement reproductible d'une version à l'autre du noyau.

FreeBSD semble être un choix optimal pour effectuer des tests sur des algorithmes standards. En effet, ce système d'exploitation procure l'assurance d'une implantation aussi proche que possible des RFC. Windows et Linux ne sont à utiliser que dans le but de tester une variante spécifique – et non disponible sous FreeBSD – ou pour effectuer une comparaison entre les implantations des systèmes.

La passerelle

La passerelle entre les deux hôtes a un rôle primordial dans le réalisme de l'expérience. Son rôle est en effet de simuler un ou plusieurs goulots d'étranglement, auquel les réactions des algorithmes sont directement liées.

Un paramètre crucial rentre en compte dans le choix du système d'exploitation : la fréquence de commutation de contexte. En effet, afin d'émuler de manière réaliste un goulot d'étranglement, celle-ci doit être la plus élevée possible. Pour les liens à forte bande passante, le gestionnaire d'ordonnancement peut être dépassé par la vitesse de départ des paquets sur le lien émulé, ce qui résulte en des rafales de paquets envoyées sur le réseau. Ainsi, la capacité maximale C d'un lien émulé (en b/s) avec une fréquence de commutation F (en Hz) et une taille maximale des paquets MTU (en bits) est donnée par la formule suivante :

$$C \approx MTU * F \tag{3.1}$$

Cette formule étant valable pour un système dédié à l'émulation du lien il est nécessaire de prendre une marge suffisamment grande afin de s'assurer de la fiabilité des résultats. Le choix a donc été fait de diviser par deux les résultats de la formule précédente pour obtenir la capacité maximale du lien émulé.

Ici, seuls deux systèmes sont utilisables : FreeBSD et Linux.

— **FreeBSD** et son module **dummynet** permettent de rapidement mettre en place un ou plusieurs goulots d'étranglement, voire de chaîner plusieurs liens virtuels avec différents débits, délais et tampons. La possibilité d'augmenter la fréquence de commutation à 20 KHz permet de simuler des liens jusqu'à 100 Mb/s. Toutefois, un nombre très limité de politiques de gestion des files est disponible, ce qui limite l'usage lors de l'étude de ces mécanismes.
— **Linux** propose quant à lui de nombreux types de files d'attente au sein de son module

netem, au prix d'une flexibilité légèrement amoindrie. Toutefois, il n'est possible de simuler de manière réaliste des liens que jusqu'à 5 Mb/s de par sa fréquence de commutation de 1KHz.

Ici, FreeBSD semble toujours être le candidat idéal, à l'exception des situations dans lesquelles d'autres algorithmes de gestion des files doivent être utilisés. Dans ce cas, l'utilisation de Linux est envisageable.

3.2 Modèle de trafic

Les expériences décrites dans les chapitres suivants sont pour la plupart réalisées sur le banc d'essai décrit dans la section précédente, et les systèmes d'exploitation adéquats sont employés en fonction des conditions. Le trafic généré doit à son tour pouvoir représenter des conditions réseaux proches du réel.

Cette tâche s'avère complexe : le trafic réseau évolue constamment, à la fois dans le temps et dans l'espace, et établir un modèle global s'avère au mieux utopique.

Toutefois, certains modèles existent, parmi lesquels les plus connus sont ceux décrits par 3GPP2 et 802.16. Leur complexité les rend toutefois peu aisés à l'utilisation. La plupart des articles qui utilisent un modèle de trafic emploient l'un des deux modèles suivants :

— Un simple modèle dans lequel les connexions sont générées par un processus de Poisson et la taille d'une connexion (ininterrompue entre son établissement et sa fin) est déterminée par une distribution à queue lourde (Zipf ou Pareto sont couramment utilisées). Ce modèle présente l'avantage de pouvoir être généré à partir de n'importe quel outil de tests réseaux comme `ipmt`.

— Un modèle de trafic http plus complexe, pour lequel un certain nombre d'objets sont requis par connexion, via un système de requête/réponse. Les intervalles entre les différents évènements sont généralement définis par un processus de Poisson, tandis que la taille des objets suit une distribution à queue lourde. Ce type de modèle demande l'implantation d'un générateur spécifique au modèle de trafic et est plus adapté aux expériences sur simulateurs.

Le premier modèle a été choisi pour sa simplicité de mise en place. Toutefois, un travail a été effectué afin d'utiliser le second modèle, et deux logiciels sont ainsi nés : `tcptest` et `tcpgen` [2]. Le premier permet de générer du trafic tel que spécifié par un fichier JSON. Le second génère les fichiers JSON selon le type de trafic choisi.

2. `https://gitlab.com/groups/tcptest`

3.3 Implantation et Distribution des sources

Comme nous avons pu le voir dans les précédentes sections, chaque OS est susceptible d'être utilisé, selon les tests effectués. Les logiciels employés doivent donc pouvoir être déployés avec un minimum de configuration.

Le choix a donc été fait d'employer des outils standards et/ou portables tels que `ssh`, `tcpdump` ou `ipmt` pour effectuer les expériences. Un environnement de tests modulaire a été écrit en bash [3]. Cet environnement permet de mettre en place un réseau de test décrit par un fichier de configuration tel que présenté dans le listing 3.1. Ce fichier permet de représenter la configuration du banc d'essai et de l'expérience : adresses IP, interfaces sur lesquelles faire les captures, nombre et type de simulations, variantes de TCP employées, lien et files d'attente émulées.

```
# Simulation duration
time=10
# Simulation type
simtype="single"
# Number of times an experiment is repeated
num="5"
# Logging directory
folderbsdcli="/mnt/shared"
folderbsdser="/mnt/shared"

# Client and Server interfaces
itfbsdcli="em0"
itfbsdser="em0"
# Ips of client, server and router
ipserver="192.168.42.1"
ipclient="192.168.43.3"
iprouterser="192.168.42.2"
iproutercli="192.168.43.2"
# TCP versions
tcpversionsbsd=("newreno" "cubic" "vegas")

# Uplink and downlink delays
di="1" #ms
do="6" #ms
# Uplink and downlink loss probabilities
probi="0.0"
probo="0"
# Uplink and downlink capacities
bwi="10000" #kbit/s
bwo="2000" #kbit/s
```

3. `https://gitlab.com/Tbraud/testbench`

```
31 # FreeBSD queues
   qibsd=("50KBytes")
33 qobsd=("50KBytes" "25KBytes" "35KBytes" "40KBytes" "12KBytes")
```

Listing 3.1: Fichier de configuration du banc d'essai pour trois machines sous FreeBSD

De même, l'architecture modulaire du banc d'essai permet d'écrire simplement des modules afin de créer de nouvelles expériences. Un exemple est donné dans le Listing 3.2. Le système d'exploitation et le rôle de la machine sont détectés automatiquement, ce qui se permet de se focaliser sur la description du comportement des hôtes. Ici, l'expérience se résume à mettre en place un client et un serveur de données et de les connecter dans le but d'échanger un flux continu de données. Les différents paramètres sont définis dans le fichier de configuration précédemment décrit, tandis que les opérations de configuration, de synchronisation et de consignation des données sont prises en charge par le moteur.

```
1  # Example experiment script
   if [ "$simtype" == "example" ]
3  then
     if [ "$side" == "server" ]
5    then
       # Do something server side
7      # For instance, set up a simple permanent server :
       tcptarget -P -p $port &
9    elif [ "$side" == "client" ]
     then
11     # Do something else
       # For instance wait for 5 seconds, then connect to the
13     # server and send data for $time seconds :
       sleep 5
15     tcpmt -p $port -d $time $ipserver &
     fi
17 fi
```

Listing 3.2: Exemple d'expérience — met en place un serveur et un client à l'aide d'`ipmt`

Ainsi, pour reproduire l'expérience, il suffit de récupérer le code du banc d'essai, le script approprié, et de relancer les tests avec les paramètres fournis dans l'article. Pour ce faire, une plateforme de redistribution d'expérience a été développée [4]. L'étape suivante serait de fournir une image permettant de déployer un système complet sur n'importe quelle machine et ainsi fournir une solution "clé en main".

4. `https://gitlab.com/django-personalsite/django-artnexp` ; une instance est disponible à `https://tristan.lyptus.fr/fr/publications/`

3.4 Conclusion

Au cours de ce chapitre ont été présentées les interrogations qui ont mené à la réalisation du banc d'essai sur lequel la quasi-totalité des expériences a été effectuée.

S'il reste difficile de pouvoir proposer des tests suffisamment exhaustifs, le choix de conditions réelles, dans des environnements suffisamment variés permet d'atteindre de bons niveaux de certitude dans les résultats, et de pouvoir comparer un grand nombre de solutions différentes.

Ainsi, les seuls inconvénients des solutions basées sur des machines réelles sont le coût financier, la quantité de travail, et la reproductibilité des expériences. Toutefois, nous avons pu voir qu'avec des moyens limités (et accessibles à un grand nombre de laboratoires d'informatiques), il était possible de réduire leur importance notamment au travers d'une solution "clé en main" qui permet de répliquer l'expérience avec un minimum de configuration.

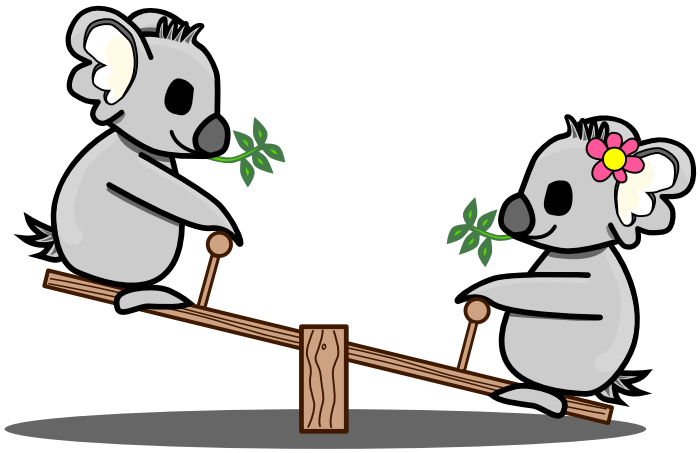
L'étape suivante vise à la mise en place d'images directement déployables, ainsi qu'une configuration réseau automatique. Ce faisant, les expériences pourront être reproduites avec une simplicité se rapprochant des simulateurs.

Chapitre 4

Dynamiques de connexions TCP antiparallèles sur un lien asymétrique

People assume that time is a strict progression of cause to effect, but *actually* from a non-linear, non-subjective viewpoint - it's more like a big ball of wibbly wobbly... time-y wimey... stuff.

The Doctor – Doctor Who



Introduction

Lorsque deux connexions TCP antiparallèles — deux connexions qui émettent des données dans des directions opposées en partageant les mêmes goulots d'étranglement dans chaque direction — partagent un goulot d'étranglement asymétrique (lien ADSL, 3/4G, etc.), leurs dynamiques sont profondément changées par rapport au cas de connexions symétriques. En effet, sur un lien chargé

ainsi dans les deux sens, on observe des oscillations parfois brutales de l'occupation des files d'attente à chacune de ses extrémités. Ces variations ne peuvent s'expliquer uniquement par les pertes par débordement et la diminution des fenêtres de congestion.

De plus, les deux tampons (montant et descendant) du goulot d'étranglement sont très rarement occupés simultanément. Les paquets de données et les accusés de réception oscillent donc entre les deux tampons, les remplissant alternativement, de manière similaire à un balancier. Ce phénomène est appelé *balancier de données*, et le déversement des données d'un tampon à l'autre *bascule*.

Plusieurs auteurs ont étudié les effets du trafic sur la voie de retour d'un lien asymétrique sur les performances de TCP et de nombreuses solutions ont été proposées, en cœur de réseau comme de bout-en-bout [83, 84, 85]. Toutefois, les phénomènes observés sont habituellement expliqués par la compression des ACKs (*ACK compression*) [86, 87]. D'autres études [88] évoquent le principe de bascule de manière incomplète.

Ce chapitre se focalise sur les causes de ces baisses de performances, et présente un modèle de l'occupation des files au cours du temps qui clarifie les interactions entre *upload* et *download*. Ce modèle procure une explication des phénomènes observés plus convaincante que la compression des accusés de réception. De plus, l'analyse apporte de nouveaux éléments aux études précédentes [88] qui affirment que la congestion reste du même côté du lien jusqu'au débordement du tampon, ce qui n'est pas toujours le cas, en particulier dans les réseaux asymétriques. Enfin, l'application du modèle permet de prédire avec une bonne précision les débits relatifs de l'*upload* et du *download* dans une configuration donnée.

Les trois contributions suivantes apparaissent dans ce chapitre :

Tout d'abord, ce chapitre raffine le modèle de Heusse *et al*, en élucidant pour la première fois ce qui gouverne de quel côté la file d'attente se remplit. Cette formule permet en outre de prédire avec exactitude les bascules indépendantes d'une perte.

Le résultat de ces analyses est ensuite injecté dans un modèle complet des dynamiques de flux TCP. Celui-ci capture le comportement de deux connexions à l'échelle du flux de données afin d'attester de l'impact du balancier sur les performances. L'étude des résultats remet notamment en cause l'influence de la compression des ACKs, qui était jusque-là soit négligée [89], soit considérée comme responsable des débits réduits observés pour le téléchargement.

Enfin, les files en cœur de réseau sont souvent dimensionnées en paquets et non en octets, ce qui diminue en partie l'effet nocif du balancier. La dernière section de ce chapitre traite des modifications à appliquer au modèle afin de pouvoir s'adapter à ce type de scénarios, plus proche de situations réelles.

4.1 De quel côté la file d'attente se remplit-elle ?

TABLE 4.1: Notations

q_x	Données dans la file (octets)
Q_x	Taille de la file (octets)
C_x	Capacité du lien au niveau TCP (octets/s)
X_x	Débit (octets/s)
w_x	Fenêtre de congestion (octets)
$w_x^\star$	Quantité de données non acquittées en attente dans les files du réseau. (octets)
τ_x	Temps d'aller-retour lorsque les tampons sont vides (s)
$RTT^\star$	Partie du délai due aux données dans les tampons, hors temps de propagation (s)

Lorsqu'une file commence à se construire d'un côté d'un lien asymétrique, le temps d'aller-retour augmente, ce qui affecte les flux de données dans les deux sens.

La file se remplit du côté de la connexion la plus exigeante en ressources. Par exemple, à la suite d'une perte sur le lien descendant, l'*upload* a une fenêtre de congestion plus grande que le *download*, et le tampon montant commence à se remplir. Le système reste dans cet état jusqu'à ce que le tampon déborde et qu'éventuellement la situation s'inverse.

Pour occuper un tampon en permanence, une connexion TCP doit atteindre une fenêtre de congestion w_x supérieure à $C_x \times \tau_x$, le produit délai bande passante du réseau entre l'émetteur et le récepteur. La part de w_x qui excède ce produit occupe inévitablement le tampon du goulot d'étranglement à moins que l'occupation de la file du chemin de retour n'augmente le RTT (et donc le produit délai bande passante effectif du système).

Soit $w^\star$ la quantité de données sur le réseau qui n'est pas en train de se propager sur les liens pour une connexion données (*cf.* notations en Table 4.1 – quantités en octets et capacités en octets/s) :

$$w_x^\star = w_x - C_x \times \tau_x,$$

où x correspond soit au *download* ou à l'*upload*, et τ_x est le temps d'aller-retour lorsque les files sont vides. $w_x^\star$ représente donc la quantité de donnée stockée dans les tampons montant et descendant du lien. Ainsi, si la file du lien descendant n'est pas vide, elle retarde les ACKs de l'*upload* d'une durée $\frac{q_\mathrm{d}}{C_\mathrm{d}}$ qui représente $\frac{q_\mathrm{d}}{C_\mathrm{d}} C_\mathrm{u}$ octets en vol. Inversement, si la file du lien montant se remplit, les ACKs du *download* sont retardés de $\frac{q_\mathrm{u}}{C_\mathrm{u}}$, soit $\frac{q_\mathrm{u}}{C_\mathrm{u}} C_\mathrm{d}$ octets en vol.

$$w_\mathrm{u}^\star = q_\mathrm{u} + \frac{q_\mathrm{d}}{C_\mathrm{d}} C_\mathrm{u}. \tag{4.1}$$

De même :

$$w_\mathrm{d}^\star = q_\mathrm{d} + \frac{q_\mathrm{u}}{C_\mathrm{u}} C_\mathrm{d}. \tag{4.2}$$

Enfin, la portion du RTT due au temps passé dans les files est égale à :

$$RTT^\star = \frac{q_\mathrm{u}}{C_\mathrm{u}} + \frac{q_\mathrm{d}}{C_\mathrm{d}} \tag{4.3}$$

Cette équation combinée aux équations 4.1 et 4.2 montre que soit :

1. Les deux files sont occupées et
$$\frac{w_\mathrm{u}^\star}{C_\mathrm{u}} = \frac{w_\mathrm{d}^\star}{C_\mathrm{d}}; \tag{4.4}$$
2. $q_\mathrm{d} = 0$, donc l'équation 4.2 n'est plus vérifiée car le téléchargement ne sature plus le lien. En conséquence, $w_\mathrm{u}^\star = q_\mathrm{u}$;
3. de même, $q_\mathrm{u} = 0$, donc $w_\mathrm{d}^\star = q_\mathrm{d}$.

En conclusion, un seul tampon peut être occupé pour une durée significative, sauf lorsque l'équation 4.4 est vérifiée. Ceci ne peut arriver que pour une durée limitée, ou dans le cas d'un lien parfaitement symétrique (ce qui est souvent la configuration par défaut des simulateurs). Dans d'autres cas — scénarios plus réalistes — **dès que l'équation 4.4 est vraie, les données en vol commencent à basculer d'un tampon à un autre**. De plus, lorsque $\frac{w_\mathrm{u}^\star}{C_\mathrm{u}} < \frac{w_\mathrm{d}^\star}{C_\mathrm{d}}$, le tampon descendant se remplit et *vice versa* (Notons que cette condition diffère de celle donnée dans l'article de Collange [88] qui n'est valide que pour un lien symétrique)

4.2 Bascule indépendante d'une perte

TABLE 4.2: Paramètres de l'expérience

$\tau_{download}$	42 ms
τ_{upload}	242 ms
C_d	288 kB/s (2.4 Mb/s at L2)
C_u	48 kB/s (400 kb/s at L2)
Tampon montant	20000 B
Tampon descendant	120000 B
Version de TCP	New Reno + SACK
Simulateur	Qualnet 6.1

Le balancier peut basculer en d'autres circonstances que lorsque l'émetteur arrête d'envoyer des données à la suite d'une perte. En effet, la quantité de données résiduelles normalisée $\frac{w_x^*}{C_x}$ peut excéder celle présente dans l'autre direction sans qu'une perte n'ait lieu. Dans ce cas, les données stockées dans un tampon basculent dans l'autre, ce qui arrive fréquemment avec des délais contrastés, car les fenêtres de congestion des connexions grandissent alors à des rythmes différents.

Pour visualiser ces effets, deux connexions TCP antiparallèles ont été démarrées sur le lien décrit Tableau 4.2. Le lien montant présente un délai plus de cinq fois supérieur au lien descendant afin de forcer l'apparition de bascules hors pertes. Les résultats de l'expérience sont présentés Figure 4-1, et ces épisodes sont visibles à $t \approx 111$s, $t \approx 135$s, $t \approx 151$s.

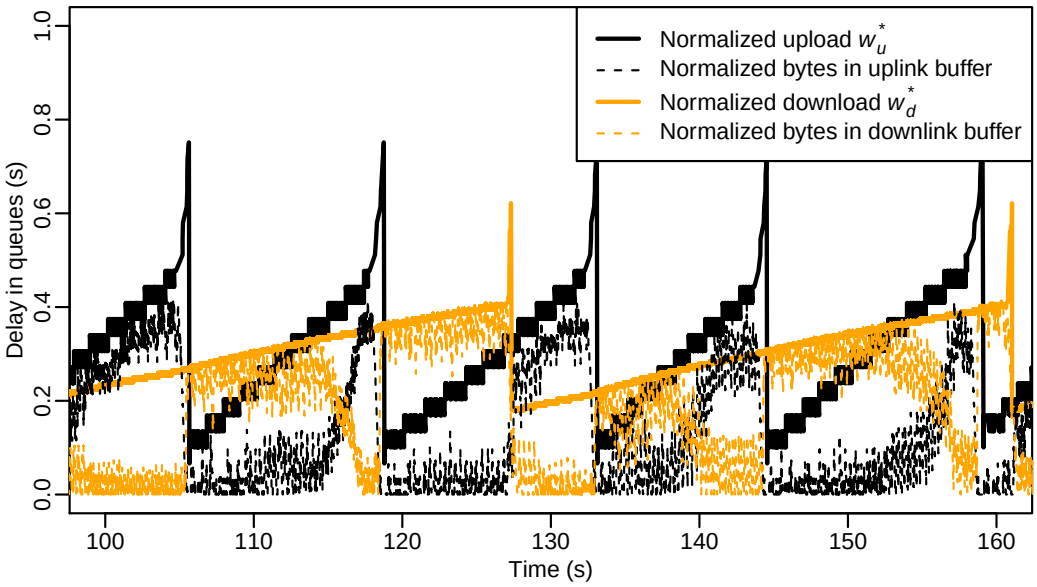


FIGURE 4-1: Épisodes de bascule décorrélés des pertes ; w_x^* représente la quantité de données qui réside dans les tampons montants et descendants, en prenant en compte les données en vol si le lien est occupé. Les données en vol résiduelles w_x^* et l'occupation de la file q_x sont normalisés par les capacités respectives du lien C_x

La Figure 4-1 illustre la manière dont les données en vol changent de côté lorsque l'équation 4.4 est vérifiée. Dans ce but, les données en vol de l'*upload* ainsi que le tampon montant ont été mis à l'échelle des valeurs du lien descendant en les multipliant par $\frac{C_\mathrm{d}}{C_\mathrm{u}}$ afin de faciliter la comparaison. Pour plus de clarté, seule la quantité de données présente dans le tampon (données résiduelles) $w_x^\star$ a été représentée, en omettant les données en cours de propagation sur les liens. Le débit sur le lien montant prend en compte la présence des ACKs du téléchargement qui consomment une part significative de la capacité — lorsque la bascule a lieu, les deux liens sont utilisés au maximum de leur capacités, et les ACKs du téléchargement utilisent une fraction $\frac{C_\mathrm{d}}{C_\mathrm{u}} \frac{60}{1508 \times 2}$ du lien montant (dans les conditions de l'expérience, l'option *delayed ACK* est activée et le simulateur ajoute 8 octets d'en-tête au niveau 2).

Commentaires

La Figure 4-1 confirme que la file se remplit du côté déterminé par l'équation 4.4. Lorsqu'une perte a lieu, l'émetteur arrête la transmission pour un RTT complet, et la bascule peut être considérée comme immédiate à l'échelle de la connexion. Dans le cas d'une bascule sans perte, cet épisode n'est plus instantané : une des files se vide progressivement tandis que l'autre se remplit. Le lien est alors utilisé à 100% dans les deux directions.

La bascule des données résiduelles d'un côté à l'autre du goulot d'étranglement a jusqu'ici été associée à celle d'un balancier. Dans le cas présent, le système est assimilable à une balance à deux bras de

longueur différente (C_u et C_d), portant chacun une charge $w_{\mathrm{u|d}}$.

Dans la suite de ce chapitre, les connections TCP seront considérées comme pouvant faire grandir leur fenêtre de congestion de manière infinie, ce qui n'est pas toujours le cas lorsque le système d'exploitation limite la quantité de données en vol. En général, les connexions descendantes sont plus susceptible d'atteindre cette limite, car le lien est plus large pour un RTT sensiblement semblable à l'*upload*. Dans ce cas, l'impact de l'asymétrie du lien sur le *download* est exacerbé.

Dans le cadre de cette analyse, les effets de la présence d'accusés de réception sur l'occupation du tampon et la capacité de la voie de retour ont été ignorés. En effet, pour des liens et tampons suffisamment grands, les ACKs ont un impact négligeable lorsque les tampons sont dimensionnés en octets. En pratique, les routeurs et systèmes d'exploitation standards ont des tampons dont la limite est fixée à un nombre fixe de paquets, quelle que soit leur taille. Ce type de dimensionnement a deux conséquences : tout d'abord, si le tampon du lien montant est trop petit, les ACKs peuvent rapidement le faire déborder. La fenêtre de congestion, qui est divisée par deux à chaque perte, ne peut atteindre le produit délai-bande passante du réseau, et empêche alors l'*upload* de remplir le lien. Ensuite, pour de plus grands tampons, la présence des ACKs limite la croissance de w_u, et en conséquence les délais causés par le tampon montant (*cf.* Chapitre 6)

4.3 Modéliser deux connexions antiparallèles dans un tampon dimensionné en octets.

Alors que la section précédente raffine la compréhension des interactions entre connexions antiparallèles à un instant donné, cette partie se focalise sur les dynamiques de la fenêtre de congestion à l'échelle de la durée des connexions. Les dynamiques, une fois modélisées, sont ensuite intégrées à un modèle complet du système qui permet de caractériser avec précision l'état du réseau à tout instant. Ce modèle est ensuite comparé à des simulations afin d'en vérifier la validité.

Le modèle décrit ci-dessous s'inspire de la machine à états décrite par Dennis Collange. Au contraire de cette dernière, les cas pour lesquels $w_i(t) = W_i$ ont été négligés (W_i étant la fenêtre annoncée par le récepteur). Le système est considéré comme idéal, avec une fenêtre annoncée W_i toujours supérieure à la fenêtre de congestion.

Le lien asymétrique entre les deux hôtes est représenté Figure 4-2.

L'augmentation de la fenêtre de congestion est dépendante du cadencement des ACKs, ainsi :

$$\frac{dw(t)}{dt} = \frac{dw}{da}\frac{da(t)}{dt} \tag{4.5}$$

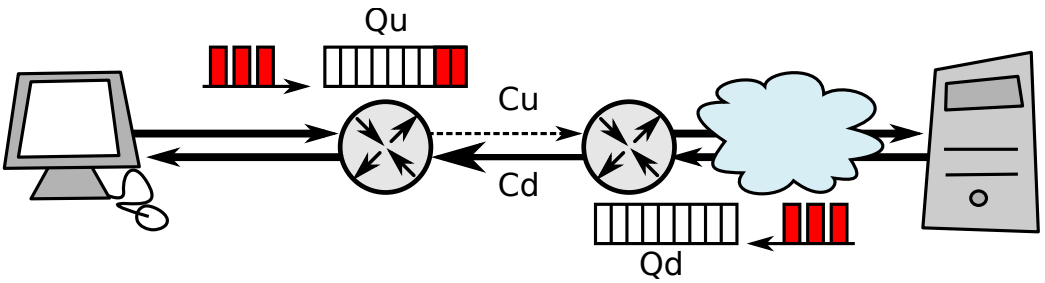


FIGURE 4-2: Deux connexions TCP antiparallèles

où $\frac{da(t)}{dt}$ est le rythme de réception des ACKs, égal au débit instantané divisé par le nombre b de paquets de données acquittés par un unique ACK. En conséquence,

$$\frac{da(t)}{dt} = \frac{X(t)}{b} = \begin{cases} \frac{w(t)}{b\tau} & \text{si q(t)} = 0 \\ \frac{C}{b\tau} & \text{si q(t)} > 0 \end{cases} \tag{4.6}$$

En *congestion avoidance*, lorsqu'une fenêtre de congestion complète a été acquittée, cette dernière grandit d'un paquet de taille m. L'équation suivante approxime la croissance de la fenêtre de congestion en fonction de la quantité de données acquittée (b paquets acquittés par ACK) :

$$\frac{dw}{da} = \frac{bm}{w(t)} \tag{4.7}$$

État initial : $q_i(t) = 0, \forall i$

Dans l'état initial, les deux tampons sont considérés vides, jusqu'à ce que l'une des deux connexions commence à remplir la file d'attente.

Les deux connexions se comportent alors de manière indépendante, et tentent chacune de remplir le lien. Chaque flux est régi par l'équation suivante :

$$w_i(t) = w_{i,0} + \frac{m}{\tau}(t - t_0), \tag{4.8}$$

où $w_{i,0}$ est la fenêtre de congestion à l'instant t_0, le début de la connexion.

De l'état initial, le système peut passer aux états **Tic** ou **Tac**. La première connexion dont la fenêtre de congestion $w_i(t)$ atteint $C_i\tau$ détermine de quel côté le tampon se remplit. Le système va donc quitter cet état après un temps t tel que :

$$t = min((C_i\tau - w_{i,0})\frac{\tau}{m}) + t_0 \quad \forall i, \tag{4.9}$$

t_0 étant défini comme l'instant auquel le système est rentré dans l'état actuel (ici, l'état initial).

États Tic ou Tac : $0 < q_i(t) < Q_i,\ q_j(t) = 0$

Le système est la plupart du temps dans les état Tic ou Tac. Dans ces deux états, une des deux connexions remplit son tampon, tandis que l'autre file reste vide. L'occupation d'un seul des deux liens se fait alors de manière optimale.

Le système est dans l'état **Tic** lorsque $\frac{w_u^*}{C_u} < \frac{w_d^*}{C_d}$, et **Tac** si $\frac{w_u^*}{C_u} > \frac{w_d^*}{C_d}$ comme vu dans la section précédente.

Les deux fenêtres de congestion grandissent proportionnellement à la capacité du lien C_i. En effet, $\frac{dw_j(t)}{dt} = \frac{m}{bw_i(t)}$, et

$$w_i(t) = \sqrt{w_{i,0}^2 + 2mC_i(t - t_0)} \tag{4.10}$$

$$\begin{aligned} w_j(t) &= w_i(t) - w_{i,0} + w_{j,0} \\ &= \sqrt{w_{i,0}^2 + 2mC_i(t - t_0)} - w_{i,0} + w_{j,0} \end{aligned} \tag{4.11}$$

Le système peut sortir de cet état par une **Bascule** ou une **Perte**, en fonction de la première condition de sortie rencontrée :

— **Bascule** après un temps :

$$\Delta t_b = \frac{1}{2mC_i}\left[\left(\frac{w_{j,0} - w_{i,0}}{C_j - C_i}\right)^2 - w_{i,0}^2\right] \tag{4.12}$$

— **Perte** après :

$$\Delta t_p = \frac{1}{2mC_i}\left[(Q_i + C_i\tau)^2 - w_{i,0}^2\right]. \tag{4.13}$$

Note : $C_d > C_u$, donc, selon l'équation 4.4 si les deux fenêtres de congestions grandissent au même rythme, le système ne peut subir une bascule que de **Tac** (la file montante se remplit) à **Tic** (la file descendante se remplit).

Bascule : $0 < q_i(t) < Q_i, \forall i$

La bascule a lieu lorsque les deux files se remplissent en même temps, ce qui n'arrive que pour $\frac{w_d^*}{C_d} = \frac{w_u^*}{C_u}$. Cet état ne peut survenir qu'entre les états **Tac** et **Tic**.

Les connexions se comportent alors comme deux flux isolés, avec pour chacun une file d'attente vide et un RTT égal à :

$$\tau' = \frac{w_u^*}{C_u} = \frac{w_d^*}{C_d}. \tag{4.14}$$

La fenêtre de congestion augmente linéairement durant cette phase, le RTT peut donc être approché par l'équation suivante :

$$\tau' = \frac{w_{i,0} + m/2}{C_i}. \tag{4.15}$$

Ainsi, les deux fenêtres grandissent à une vitesse :

$$w_i(t) = w_{i,0} + \frac{C_i m}{w_{i,0} + m/2}(t - t_0) \tag{4.16}$$

Cette phase dure un RTT après lequel le système passe à **Tic**, ce qui correspond au remplissage de la file du lien montant. Le système ne peut pas aller dans le sens inverse (**Tac** vers **Tic**), (voir état **Tic**/**Tac**).

Perte : $q_i(t) = Q_i, q_j(t) = 0$

Lorsqu'une perte a lieu quelque part sur le réseau, les fenêtres de congestions du lien montant et descendant continuent leur croissance pendant un RTT. À la suite de ce RTT, la perte est détectée et le système passe en **Récupération de perte**.

Récupération de perte : $q_i(t) = 0, 0 < q_j(t) \leq Q_j$

Lorsqu'un émetteur détecte une ou plusieurs perte(s), les paquets concernés sont retransmis. L'émetteur divise alors sa fenêtre de congestion par deux, ce qui a pour effet d'interrompre toute transmission pendant environ 1 RTT, et de vider le tampon congestionné. Au cours de cette phase, q_i, la quantité de données résiduelle présente dans le tampon qui déborde bascule intégralement dans la file d'attente opposée. Les ACKs deviennent alors des paquets de données et vice-versa. Le système peut quitter cette phase pour trois états :

- après 1 RTT, $\frac{w_i^*(t)}{C_i} > \frac{w_j^*(t)}{C_j}$, passage à l'état **Tic** correspondant,
- après 1 RTT, $\frac{w_i^*(t)}{C_i} < \frac{w_j^*(t)}{C_j}$, passage à l'état **Tac** correspondant,
- $w_j^*(t) > q_j$: une **perte** a lieu pour la connexion j. Ce cas de pertes simultanées n'est possible que lorsqu'on considère que le contenu d'une file d'attente se déverse instantanément dans le tampon du lien de retour.

TABLE 4.3: Paramètres de l'expérience

τ_d	42 ms	
C_d	10 Mb/s	20 Mb/s
Q_d	150 KB	300 KB
τ_u	42 ms	
C_u	1 Mb/s,2 Mb/s,5 Mb/s	4 Mb/s
Q_u	36 KB, 75 KB	72 KB,150 KB
	105 KB, 150 KB	210 KB, 300 KB

La Figure 4-3 représente l'automate qui caractérise ce modèle.

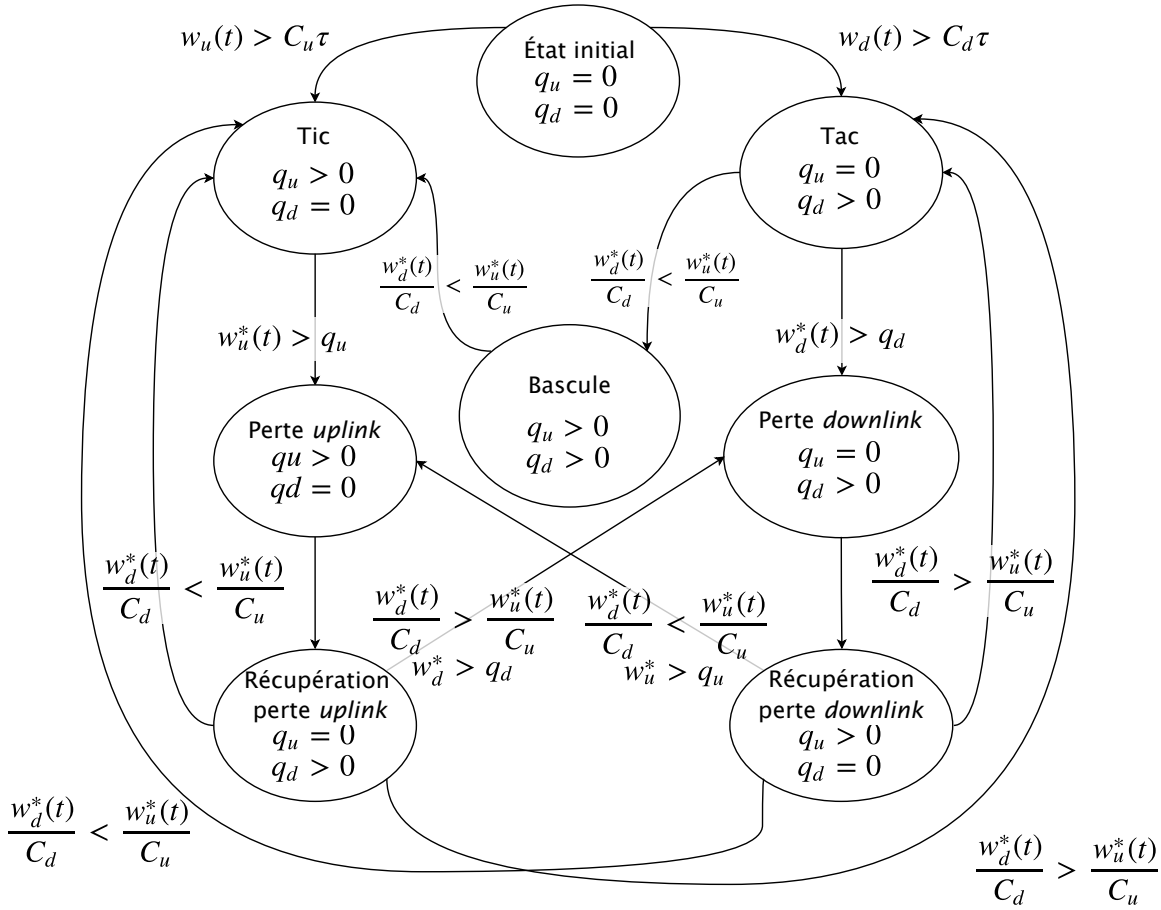


FIGURE 4-3: Dynamiques de deux connexions TCP antiparallèles

Mise en application du modèle

Afin de juger de la précision du modèle décrit dans la section précédente, plusieurs expériences ont été réalisées. Le Tableau 4.3 donne les conditions expérimentales choisies pour englober une grande variété de situations. Le goulot d'étranglement va de modérément à fortement asymétrique (5/10 Mbps à 1/10 Mbps), tandis que les files sont soit bien dimensionnées, soit surdimensionnées à divers degrés par rapport au lien montant. Afin de valider le passage à l'échelle du modèle, le même jeu d'expériences est également reproduit sur un lien à 20 Mbps/4 Mbps.

Le simulateur *Qualnet* est utilisé pour la totalité des expériences, en utilisant la topologie décrite Figure 4-2. Une connexion est lancée dans chaque direction pendant 2000 secondes. Les mesures de débit sont faites sur l'intervalle de temps $[300; 2000]s$ afin de garantir que les deux connexions ont atteint un état stable en *congestion avoidance*.

Les simplifications suivantes ont été apportées au modèle :

— La bascule est instantanée.

— La détection et la récupération de perte sont regroupées au sein d'un seul état de durée nulle.

— les ACKs sont considérés négligeables sur les liens et dans les tampons.

Le modèle est exécuté dans les mêmes conditions que la simulation (cf. Table 4.3). Ici, les valeurs moyennes des fenêtres de congestion sur les liens montants et descendants sont calculées entre 300 et 2000 s. En effet, le RTT est le même pour les deux connexions, donc $\frac{w_{avg,d}}{w_{avg,u}} \approx \frac{X_{avg,d}}{X_{avg,u}}$. Ainsi, le ratio des moyennes de fenêtres de congestion est une bonne approximation du ratio des débits mesurés lors des simulations.

Les résultats de ces expériences sont présentés en Figure 4-4. Comme prévu, $\frac{X_d}{X_u}$ suit de manière linéaire le ratio des files.

Le modèle donne des valeurs très proches de la réalité dans le cas de liens modérément asymétriques (10 Mbps/1 Mbps et 20 Mbps/4 Mbps) avec une erreur de quelques pourcents seulement.

Dans le cas d'un lien faiblement asymétrique (10 Mbps/5 Mbps), le modèle est légèrement moins précis (10% d'erreur), à l'exception du cas où $Q_u = 36$ kB (queue en *upload* sous-dimensionnée) pour lequel la valeur mesurée dérive de plus de 40% de la valeur prédite.

Pour comprendre cette différence, les données en vol sont présentées Figure 4-5. Au moins 2 pertes sont clairement décorrellées de l'état du balancier, aux instants $t \approx 694$s et $t \approx 702$s. Dans cette configuration, la fenêtre normalisée w_u^* est souvent plus basse que l'occupation des files d'attente. De fortes rafales de paquets arrivent de manière transitoires dans le tampon, alors que le réseau n'est pas saturé. Ce phénomène est connu sous le nom de *compression des ACKs*. La compression des ACKs peut ainsi entraîner des rafales de paquets plus grandes que la file montante, et causer son débordement. L'impact de ce mécanisme est d'autant plus marqué que le ratio entre les queues montantes et descendantes est grand. Contrairement à l'idée la plus répandue, nous sommes en présence d'un cas dans lequel la compression des ACKs s'avère bénéfique pour le *download* !

Sur un lien fortement asymétrique (10 Mbps/1 Mbps), le modèle se détache des simulations. Cette imprécision provient des simplifications du modèle. En effet, à ce niveau d'asymétrie, les accusés de réception sur le lien montant occupent une part non négligeable de la file d'attente et de la capacité du lien. De plus, on a supposé qu'une perte était instantanée, c'est-à-dire que la totalité du tampon bascule dans le tampon opposé en un temps nul. En pratique, pour un lien très asymétrique, ce

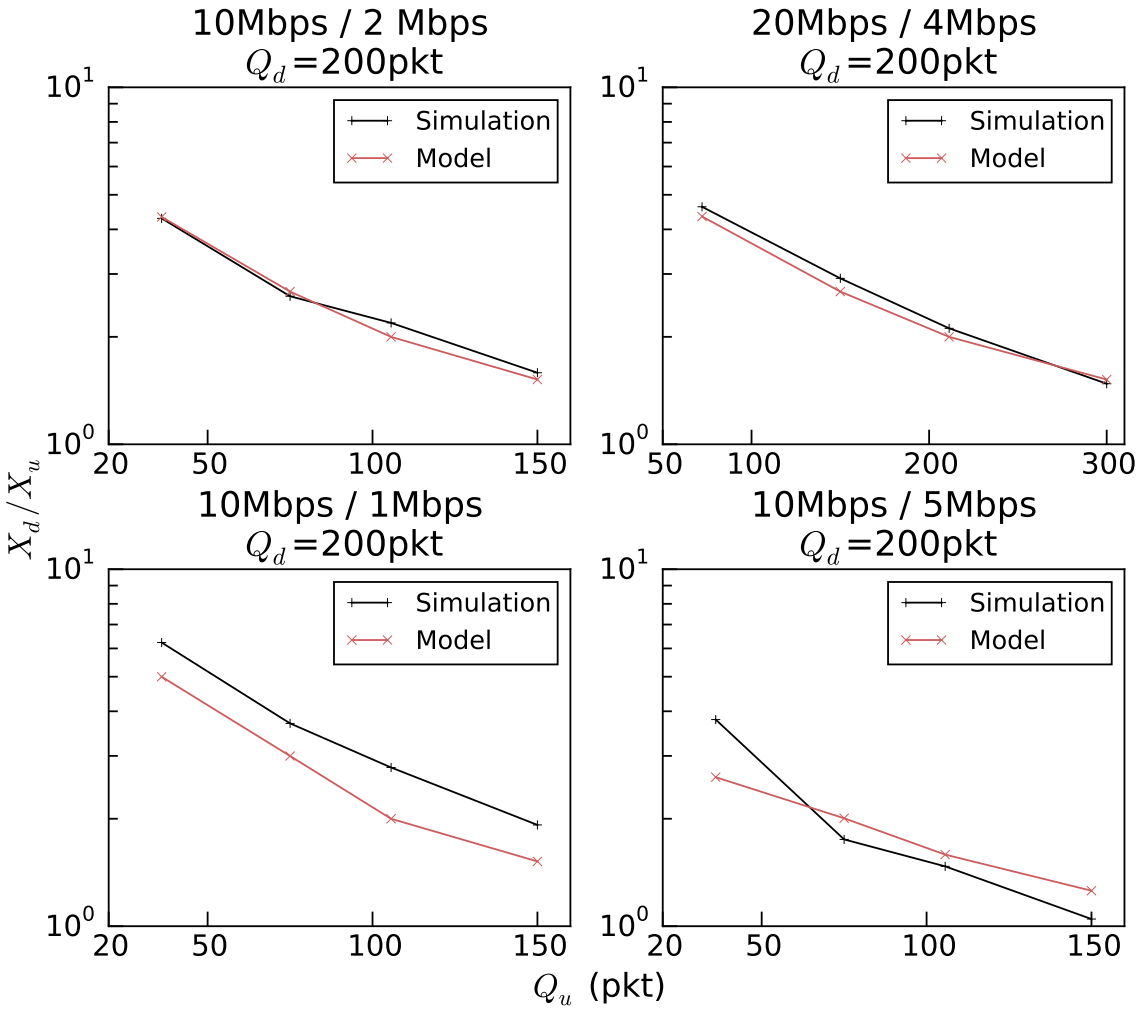


FIGURE 4-4: Comparaison entre le ratio des débits obtenu lors des simulations et calculé selon le modèle de la Section 4.3.

facteur dépend de la probabilité qu'une suite d'ACKs soit suffisamment longue dans une file pour générer une rafale de paquets plus grande que le tampon opposé.

Figure 4-6, le modèle a été modifié pour prendre en compte les cas pour lesquels $w_x^*(t) > Q_x \forall x$ n'est plus une condition suffisante de perte simultanée. Celle-ci n'est ainsi plus systématique, mais soumise à une probabilité p qu'il devient ainsi possible d'encadrer. Ici, une perte simultanée n'apparaît que dans 33 à 50% des cas. Cette probabilité devient plus faible avec le ratio des files : lorsque les deux files font la même taille, environ 50% des pertes déclenchent une perte dans l'autre direction, tandis que dans le cas d'une file montante plus petite que 36 KB, cette probabilité s'approche de 0.

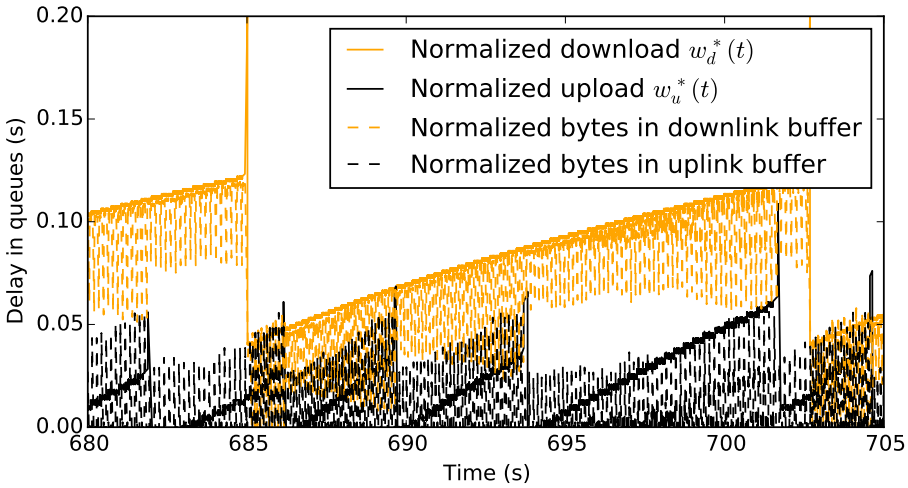


FIGURE 4-5: Fenêtres de congestion montantes et descendantes normalisées pour un lien asymmétrique – $C_d = 10$Mb/s, $C_u = 5$Mb/s, $Q_d = 150$KB, $Q_u = 36$KB – Les pertes en *upload* ont lieu indépendamment des bascules.

4.4 Modéliser deux connexions antiparallèles dans des tampons dimensionnés en paquets

Dans les sections précédentes, le comportement de connexions antiparallèles était modélisé pour des tampons en octets. Toutefois, pour des raisons de simplicité d'implantation, la plupart des tampons des routeurs sont dimensionnés en paquets, ce qui change les dynamiques des interactions entre connexions.

Les changements suivants sont nécessaires à la modélisation d'un système en paquets :

Créer une file en paquets à partir d'une file en octets

Une file en paquets est plus complexe à intégrer dans un modèle continu tel que décrit en Section 4.3. Toutefois, il est possible d'approcher le comportement d'une file en paquets en prenant pour base une file en octets. Dans l'hypothèse d'un environnement composé uniquement de paquets de données et d'accusés de réception, dans lequel tous les paquets de données ont la même taille, et tous les ACKs accusent réception de la même quantité de données, une file en paquets de taille q_p peut être représentée comme une file en octets de taille q_b, qui déborde lorsque la condition suivante sont vérifiée.

$$q_b < n_{data} \times m + n_{ack} \times m \times b \tag{4.17}$$

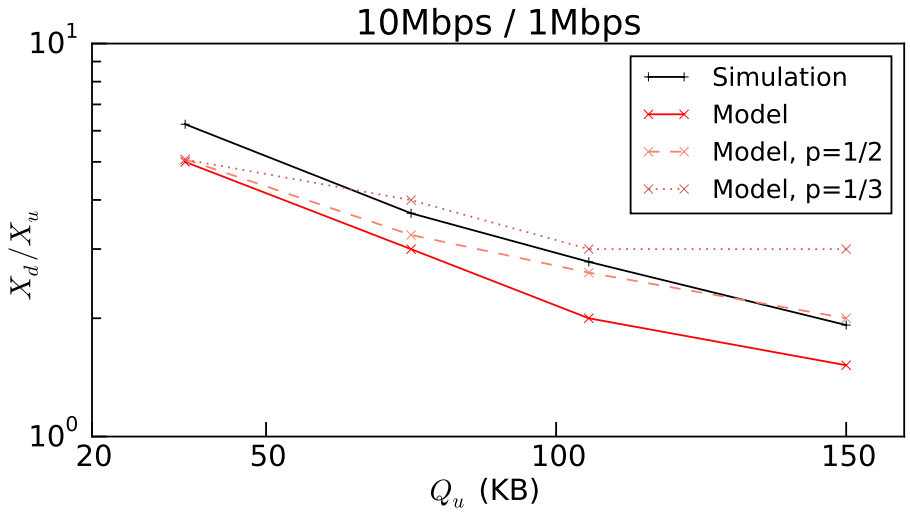


FIGURE 4-6: Comparaison entre les ratio des débits obtenus par les simulations et les débits calculés grâce au modèle présenté Section 4.3 avec diverses probabilités de pertes simultanées.

Tic/Tac : $0 < q_i(t) < Q_i, q_j(t) = 0$

La connexion quitte cet état pour une bascule ou une perte après un temps $t = min(t_c, t_l)$. Toutefois, les ACKs dans le tampon sont désormais pris en compte pour estimer l'instant de la perte.

Le nombre d'ACKs dans le tampon peut être approximé de la manière suivante :

$$n_{ack}(t) \times m \times b \approx w_j^*(t) = w_j(t) - C_j\tau = w_{j,0} - w_{i,0} + w_i(t) - C_j\tau \tag{4.18}$$

Ainsi, une perte a lieu à l'instant :

$$\Delta t = \frac{b}{2mC_i}\left[(Q_i + (C_i + C_j)\tau + w_{i,0} - w_{j,0})^2 - w_{i,0}^2\right] \tag{4.19}$$

Perte

La transition de l'état "récupération de perte sur le lien montant/descendant" vers l'état "récupération de perte sur le lien descendant/montant" n'est plus valide tant que le ratio des files (en paquets) est supérieur au ratio des capacités multiplié par le nombre de paquets acquittés par ACK.

Modèle et simulations

Le modèle a été confronté à des résultats de simulations obtenus à travers une série d'expériences similaire à celle présentée Section 4.3 avec les paramètres décrits Table 4.4. Les résultats sont présentés Figure 4-7.

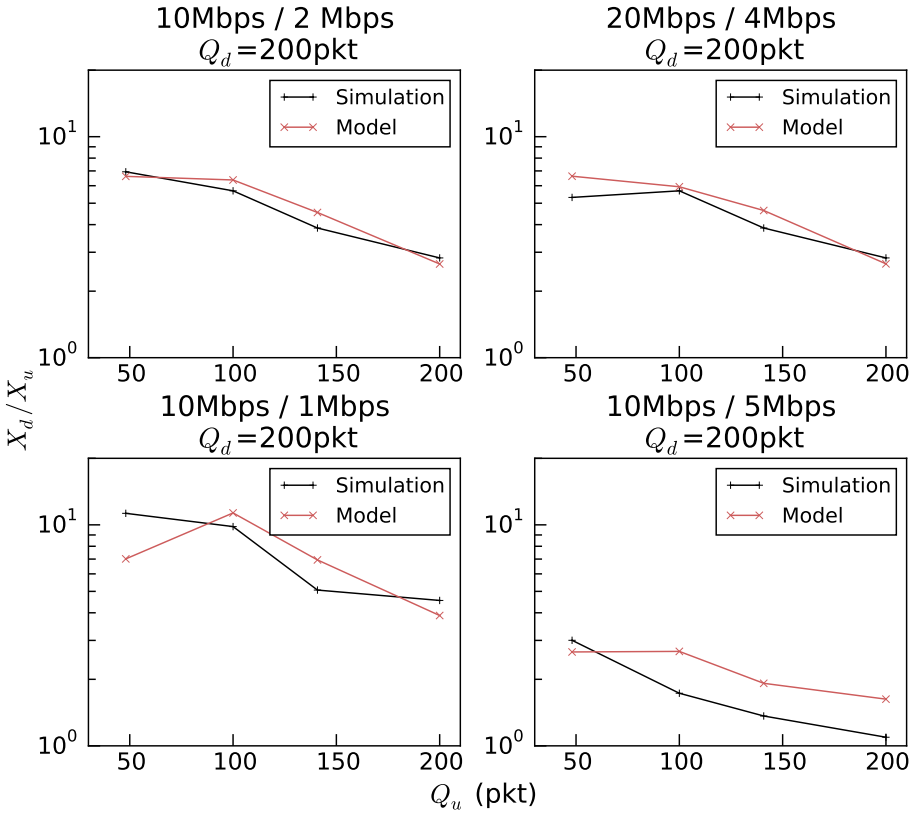


FIGURE 4-7: Comparaison entre les ratios de débits mesurés en simulation et calculés en utilisant le modèle de la Section 4.4, taille de la file en paquets.

TABLE 4.4: Paramètres de l'expérience

τ_d	42 ms	
C_d	10 Mb/s	20 Mb/s
Q_d	200 pkt	
τ_u	42 ms	
C_u	1 Mb/s, 2 Mb/s, 5 Mb/s	4 Mb/s
Q_u	48 pkt, 100 pkt, 140 pkt, 200 pkt	

De manière similaire à la Section 4.3, le modèle est très proche des valeurs mesurées pour des liens modérément à fortement asymétriques (10 Mbps/2 Mbps, 20 Mbps/4 Mbps and 10 Mbps/1 Mbps).

En ce qui concerne les liens faiblement asymétriques (10/5 Mb/s), les résultats, bien que surestimés, suivent la tendance générale de la courbe. La différence entre simulation et modèle peut s'expliquer par le fait que les liens ne sont pas en permanence occupés, contrairement aux hypothèses faites dans le modèle. De plus, les cas dans lesquels les pertes simultanées apparaissent n'ont pas été pris en compte. Néanmoins, lorsque les bandes passantes sont similaires, les pertes simultanées — ainsi que les pertes non corrélées avec des bascules — peuvent avoir lieu à cause de la compression des ACKs, qui de par sa nature aléatoire a été écartée du modèle.

4.5 Conclusion

Ce chapitre confirme qu'il est impossible de maintenir les deux liens continuellement actifs tant qu'une approche fondée sur le cadencement des ACKs régule la transmission dans des tampons de type FIFO. Toutefois le fait d'utiliser des files dimensionnées en paquets diminue légèrement l'impact des interactions entre connexions sur les performances.

Le modèle présenté rend compte de ces interactions et souligne dans certains cas l'influence des phénomènes laissés de côté, telle la compression des ACKs. En effet, nous avons vu que celle-ci peut de temps en temps favoriser le *download* au lieu de le ralentir.

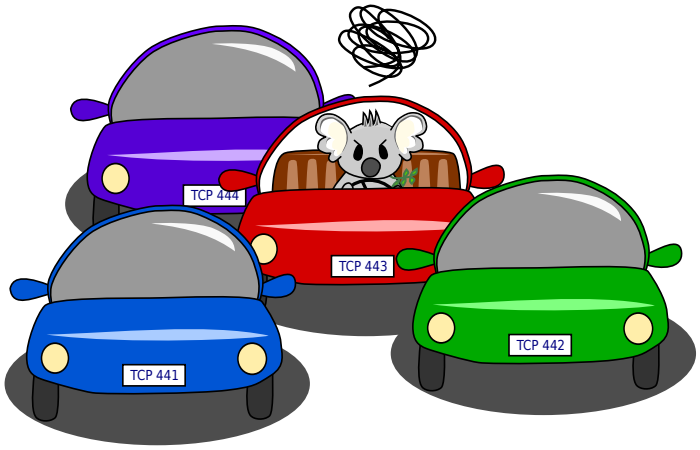
Dans les chapitres suivants, nous nous intéressons à l'étude de solutions pratiques à ces interférences. Bien que ces dernières puissent répondre aux problématiques traditionnelles de *bufferbloat*, l'étude se focalise sur le cas de tampons surdimensionnés sur la voie de retour, et plus particulièrement la réaction d'une ou plusieurs connexions sur le lien descendant, tant en termes de latences que de débits.

Chapitre 5

Étude des variantes de TCP dans le cas de connexions antiparallèles

It's obvious, once I stop to think.

Caleb – Ex Machina



Le chapitre précédent présente un modèle analytique de deux connexions TCP (*New Reno*) dans un lien asymétrique. Ce modèle permet d'attester de l'effet du mouvement de balancier des données en vol sur les débits, tant montant que descendant. En effet, en dehors de périodes transitoires, un seul des deux tampons peut être occupé continûment, et donc une seule des deux connexions peut bénéficier entièrement de la capacité disponible.

Ce phénomène et ses conséquences ont été mis en évidence à plusieurs reprises pour des connexions TCP *New Reno* [89, 88] alors que d'autres variantes sont largement déployées. Celles-ci diffèrent au niveau de l'algorithme de contrôle de congestion qui joue un rôle central dans la dynamique du

système.

Ce chapitre se focalise désormais sur le comportement de connexions TCP antiparallèles en situation réelle. Les variantes les plus populaires de l'algorithme (*New Reno*, *Vegas*, *Compound* et *Cubic*) sont comparées les unes aux autres afin d'évaluer leur réaction face à une connexion sur la voie de retour. En effet, chaque variante présente ses propres spécificités. TCP *Vegas* propose ainsi un algorithme de contrôle de congestion totalement différent de *New Reno* ; il vise à maintenir l'occupation du tampon la plus basse possible. Au contraire, TCP *Cubic* et *Compound* sont beaucoup plus agressifs durant l'expansion initiale de la fenêtre de congestion avant de ralentir lorsque la file d'attente commence à saturer.

De plus, compte tenu des fortes interactions entre connexions montantes et descendantes (*cf.* Chapitre 4), la combinaison de différentes variantes TCP en *upload* et *download* donne lieu à des dynamiques différentes selon le sens de la connexion. Les algorithmes de contrôle de congestion ont donc été testées les uns face aux autres, avant de les soumettre au scénario plus réaliste de connexions de tailles finies générées aléatoirement confrontées à un long *upload* sur le chemin de retour.

Les expériences se focalisent donc sur quatre versions courantes de TCP, mises en compétition l'une face à l'autre afin d'observer leur comportement sur le lien descendant lorsqu'une connexion est présente sur le lien montant.

La première variante testée, *New Reno*, peut être considérée comme le mètre étalon du contrôle de congestion : il s'agit en effet de la version originale de l'algorithme, encore largement déployée (par défaut sous FreeBSD et Windows), et sert de référence dans ces tests.

TCP *Cubic* est également une version populaire dans les serveurs (algorithme par défaut sous Linux), qui inclut des modes opératoires spécifiques afin de le distinguer des autres variantes, notamment une accélération plus agressive à la suite d'une perte afin de remplir rapidement les liens de fort produit délai-bande passante. La croissance de la fenêtre de congestion est ici déterminée comme une fonction cubique du temps centrée autour de sa dernière valeur maximale.

La boucle de contrôle basée sur le délai de TCP *Vegas* est représentative d'une approche originale, potentiellement avantageuse dans le cas de connexions antiparallèles. Comme démontré dans le chapitre précédent, les interactions entre *upload* et *download* sont fortement liées à des délais excessifs dans les files d'attente du réseau. TCP *Vegas* tente de garder l'occupation des tampons la plus faible possible, et est donc insensible donc les tampons surdimensionnés.

Enfin, TCP *Compound*, la version développée par Microsoft, propose un fonctionnement entre *Vegas* et *Cubic* : l'accélération est plus forte que pour *New Reno*, mais la vitesse de cette accélération est contrainte par les délais observés dans le réseau. *Compound* n'est activé par défaut que sur les versions serveur de Windows, et est disponible en option pour les clients

L'objectif de ce chapitre est de trouver quelle approche résulte dans une meilleure utilisation du réseau en répondant aux besoins de l'utilisateur (forts débits et faibles latences pour le *download*), bien que le chapitre précédent ait établi qu'il était impossible d'utiliser le lien à 100% dans les deux directions en même temps.

5.1 Conditions expérimentales

Les expériences ont été menées sur un banc d'essai composé de trois ordinateurs, comme décrit dans le Chapitre 3. Le routeur utilise FreeBSD 9.2, tandis que les hôtes alternent entre FreeBSD 9.2 pour les variantes *New Reno*, *Cubic* et *Vegas*, et Windows 7 pour TCP *Compound*. Sur le routeur, le lien asymétrique est émulé à l'aide du module *Dummynet*. Les hôtes utilisent une version *Vanilla* du noyau, tandis que le routeur fonctionne grâce à une version recompilée avec une fréquence d'interruption supérieure à la normale (Option HZ à 10000), ce qui permet théoriquement d'émuler des liens jusqu'à 100 Mb/s.

τ_d	42 ms	
τ_u	52 ms	
C_d	288 kB/s	1.2 MB/s
	2.4 Mb/s au L	10 Mb/s au L2
C_u	48 kB/s	240 kB/s
	400 kb/s au L2	2 Mb/s au L2
Q_u	10/60 packets	40/250 packets
Q_d	60 packets	250 packets
TCP	*New Reno, Vegas, Cubic, Compound*	

TABLE 5.1: Paramètres de l'expérience

Les conditions expérimentales sont décrites Table 6.3. Le lien descendant émulé a une capacité de 2.4 Mb/s, tandis que le lien montant est à 400 kb/s. Les tampons montants font respectivement soit 10 paquets (un tampon correctement dimensionné, pour une latence faible) soit 60 paquets (un tampon surdimensionné), tandis que le tampon descendant est à 60 paquets. En effet, utiliser des tampons montants et descendants de taille similaires est une pratique fréquente qui produit de fortes interférences entre *upload* et *download* [90]. Le lien ainsi émulé est très proche d'un lien ADSL typique.

Afin d'atteindre des produits délai-bande passante plus élevés (et donc déclencher les modes spécifiques de *Cubic* et *Compound*), un autre lien asymétrique est émulé avec des capacités montantes et descendantes de respectivement 10 Mb/s et 2 Mb/s. Le tampon montant est de 250 paquets lorsque surdimensionné, 40 paquets sinon. Le tampon descendant est de 250 paquets dans les deux cas.

Une connexion longue est générée dans le sens descendant, suivie par une autre, dans le sens montant 20 à 40 secondes plus tard (durée déterminée de manière aléatoire). Cette expérience s'arrête après 1000 s. Le trafic est généré à l'aide de `tcpmt` (suite `ipmt` [91]), observé à l'aide de `tcpdump` et analysé

avec `tcptrace`.

Les mesures sont effectuées lorsque les deux connexions sont en *congestion avoidance* et ont stabilisé leurs fenêtres de congestion. Dans ce but, seule la partie de la trace comprise entre 200 et 800 s est prise en compte.

Au cours de ces expérimentations, certaines variantes de TCP se sont stabilisées autour de débits qui peuvent varier fortement d'une expérience à l'autre Dans le but d'obtenir des statistiques fiables, chaque expérience a été lancée 10 fois. Les résultats présentés correspondent aux valeurs moyennes et écart types mesurés pour ces jeux de données.

5.2 Tampons correctement dimensionné

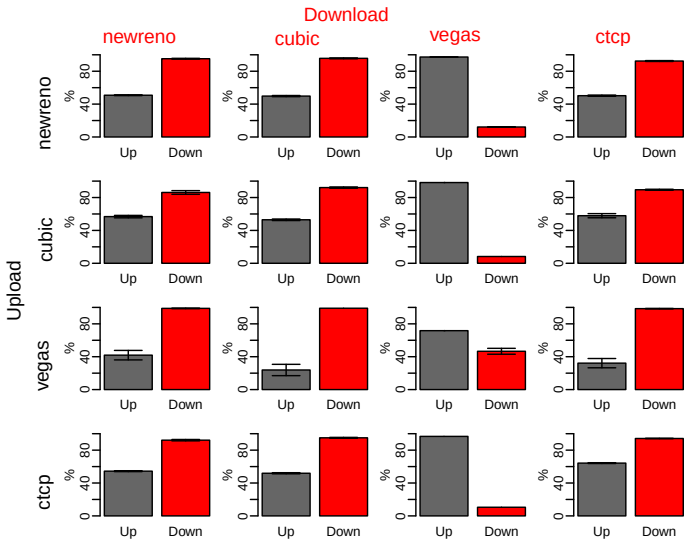


FIGURE 5-1: Utilisation du lien (%) et écart type pour un tampon correctement dimensionné — respectivement 60 et 10 paquets pour les liens descendant et montant — 2.4/.4 Mbps

La Figure 5-1 montre l'utilisation du lien en confrontant les variantes de TCP sur un lien un asymétrique dont les tampons sont correctement configurés (*i.e.* 10 *vs.* 60 paquets).

Lorsque TCP *New Reno* est utilisé face à lui-même, le *download* utilise de 90 à 100% du lien, tandis que l'*upload* peine à atteindre 60%. En effet, les tampons sont dimensionnés en paquets : les ACKs sur la voie de retour peuvent facilement causer un débordement du tampon. L'*upload* ne peut donc pas saturer le lien.

Les résultats observés pour TCP *Cubic* sont assez similaires à *New Reno* : le délai et la bande passante du lien sont relativement petits, et *Cubic* passe la majeure partie du temps en mode "*New Reno*"

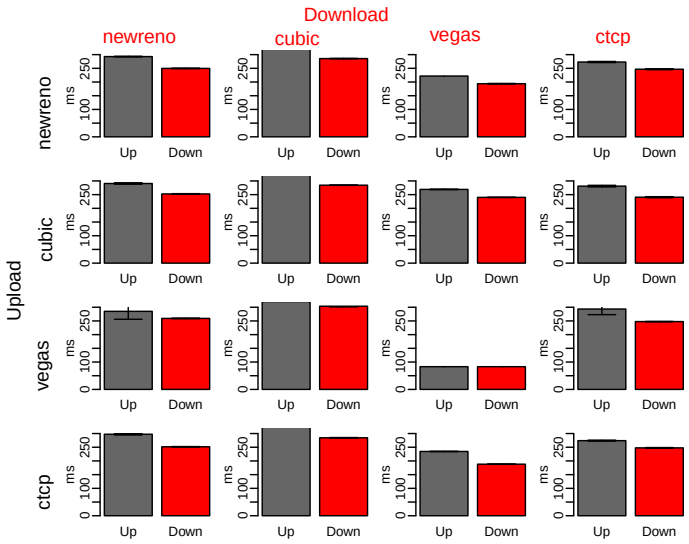


FIGURE 5-2: RTT observé par les connexions TCP et écart type pour un tampon correctement dimensionné — respectivement 60 et 10 paquets pour les liens descendant et montant — 2.4/.4 Mbps

TCP *Vegas* essaie de garder une occupation du tampon minimale, ce qui s'avère être désastreux en termes de performances : le partage d'un tampon entre TCP *Vegas* et *New Reno*/*Cubic*/*Compound* se fait au détriment de *Vegas*, qui s'efface devant les autres connexions. Ce phénomène est bien connu lorsque deux transferts de données vont dans la même direction [92]. Cette expérience étend ce résultat au trafic sur la voie de retour. *Vegas* est la seule variante pour laquelle, lorsque utilisée sur le lien descendant, l'*upload* atteint presque 100%. Toutefois, utiliser *Vegas* du côté serveur s'avère catastrophique pour le *download*.

En ce qui concerne TCP *Compound*, la fenêtre de congestion est ajustée en fonction du délai et des pertes. Pourtant, les résultats sont similaires à TCP *New Reno*. De manière identique à TCP *Cubic*, le produit délai-bande passante est trop bas pour que *Compound* puisse sortir du "mode *New Reno*".

La Figure 5-2 présente les RTTs de l'*upload* et du *download* pour un tampon correctement configuré sur le lien montant. Dans chaque situation, les RTT sont approximativement identiques pour les connexions sur les liens montant et descendant, ce qui confirme les résultats du chapitre précédent : la portion du RTT due au temps passé dans les files d'attente est égale pour les deux côtés du goulot d'étranglement. De plus, les RTT observés sont inférieurs de 400 ms, ce qui reste raisonnable pour la majorité du trafic.

TCP *Compound* et *New Reno* sont ici encore très similaires, tandis que *Cubic* obtient des RTT légèrement plus élevés (d'environs 10%).

Réciproquement, *Vegas* tente de garder l'occupation de la file la plus basse possible, ce qui correspond à garder $\frac{q_u}{C_u}$ le plus proche de 0 possible. Ainsi, les RTT observés sont bien moins élevés à la fois

pour l'*upload* et le *download*.

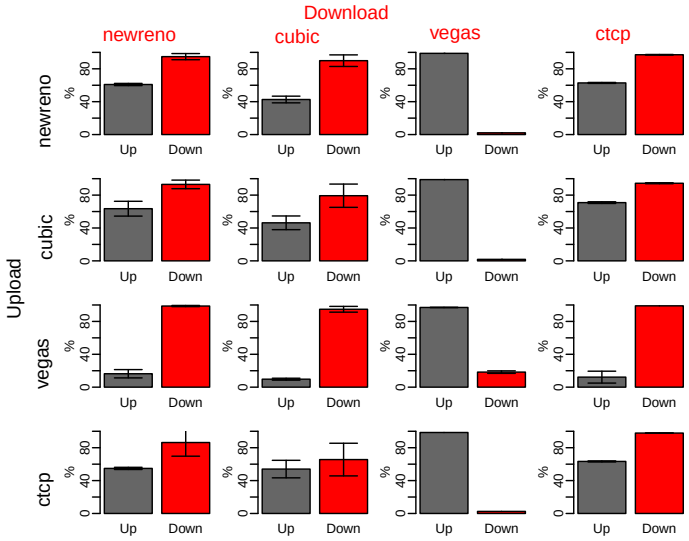


FIGURE 5-3: Utilisation du lien (%) et écart type pour un tampon correctement dimensionné — respectivement 250 et 40 paquets pour les liens descendant et montant — 10/2 Mbps

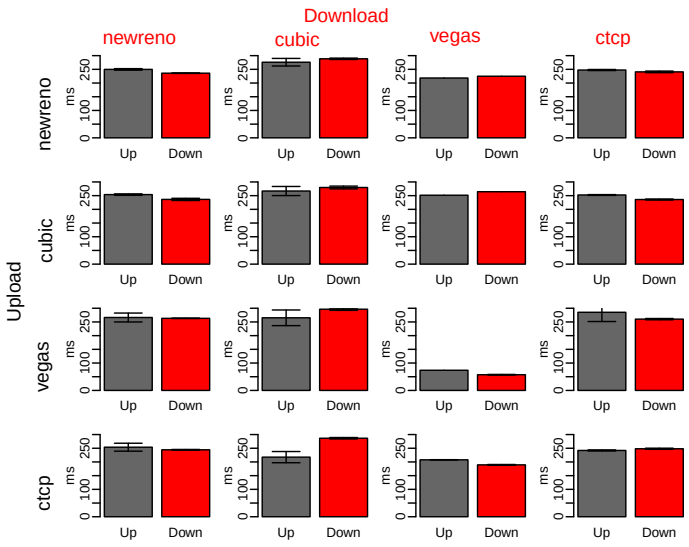


FIGURE 5-4: RTT observé par les connexions TCP et écart type pour un tampon correctement dimensionné — respectivement 250 et 40 paquets pour les liens descendant et montant — 10/2 Mbps

Les Figures 5-3 et 5-4 présentent respectivement le débit et le RTT de l'*upload* et du *download* pour un second lien asymétrique de capacité plus élevée (10 Mb/s - 250 pkt down *vs* 2 Mb/s - 40 pkt up). Le produit délai-bande passante est ainsi multiplié par 4, les différences devraient être plus marquées entre TCP *New Reno*, *Cubic* et *Compound*.

Toutes les variantes à l'exception de *Vegas* atteignent la même utilisation du lien descendant, autour de 100%. En effet, un dimensionnement correct des tampons est le facteur clé dans le bon fonc-

tionnement des algorithmes basés sur le cadencement des ACKs. La différence principale entre les versions de TCP apparaît lorsque TCP *Cubic* est utilisé côté serveur, et *Compound* pour le client, c'est-à-dire dans le cas le plus fréquemment rencontré ! Cette combinaison entraîne un débit plus faible sur le lien descendant, avec une variance élevée.

Des différences mineures apparaissent sur le lien montant. En effet, *Compound* utilisé sur le lien montant permet d'obtenir des débits plus élevés en *upload* que *New Reno* (entre 5 et 10%), tandis que *Cubic* sur le lien descendant a tendance à écraser la connexion montante (débits entre 30 et 40% plus faibles). En revanche, lorsque *Cubic* est utilisé sur le lien montant, l'*upload* présente des débits entre 5 et 10% plus élevés.

Ces différences proviennent du produit délai-bande passante, désormais suffisamment élevé pour déclencher le "mode *Cubic*" ou "*Compound*". *Cubic* devient ainsi plus agressif afin d'atteindre rapidement la capacité du lien. Ainsi, les débits observés ont tendance à être plus élevés. De même, la composante basée sur le délai de *Compound* joue sur l'accélération de la croissance de la fenêtre. La connexion est donc plus agressive jusqu'à commencer à remplir le tampon, moment à partir duquel l'algorithme se comporte comme TCP *New Reno*. À l'instar de *Cubic*, cette agressivité ne bénéficie pas à *Compound*, qui présente des débits en *download* plus bas que *New Reno*.

TCP *Vegas* présente des performances plus faibles que dans l'expérience précédente. En effet, *Vegas* envoie des données en fonction des délais observés. Ici, la même quantité de données est envoyée pour une capacité multipliée par 4 (les débits observés sont exactement les mêmes que dans l'expérience précédente dans le cas du *download*, pour une amélioration mineure des performances de l'*upload*).

En ce qui concerne les RTT présentés Figure 5-4, aucune différence significative n'est constatée, les files étant dimensionnées selon les mêmes ratios.

5.3 Tampon montant surdimensionné

La Figure 5-5 représente le pourcentage d'utilisation du lien obtenu en confrontant les variantes de TCP sur un lien asymétrique avec le même tampon aux deux extrémités du goulot d'étranglement. Dans cette configuration, l'effet de bascule est très marqué avec TCP *New Reno* : l'*upload* atteint 90% de la capacité du lien, tandis que le *download* tombe aux alentours de 50%. Alors que les délais augmentent dans le lien, TCP *Cubic* passe plus de temps en "mode *Cubic*", et la différence entre *Cubic* et *New Reno* devient significative. En effet, *Cubic* procure une meilleure équité en termes de débits, qu'il soit utilisé sur le lien montant ou descendant, au prix d'une plus grande variabilité dans les débits et de performances réduites pour le *download*. Un autre point à noter est la variation importante du débit moyen d'une expérience à l'autre.

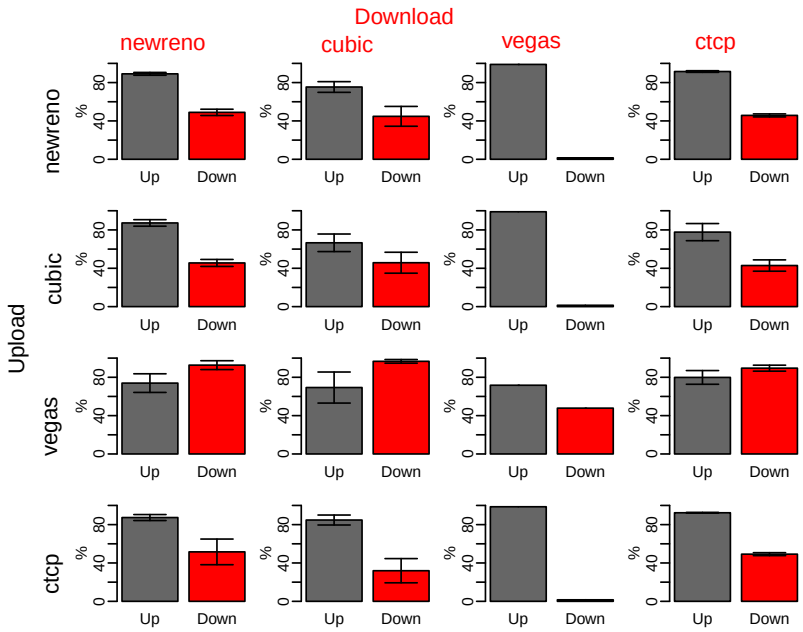


FIGURE 5-5: Utilisation du lien (%) et écart type pour un tampon montant surdimensionné — 60 paquets montants et descendants — 2.4/.4 Mbps

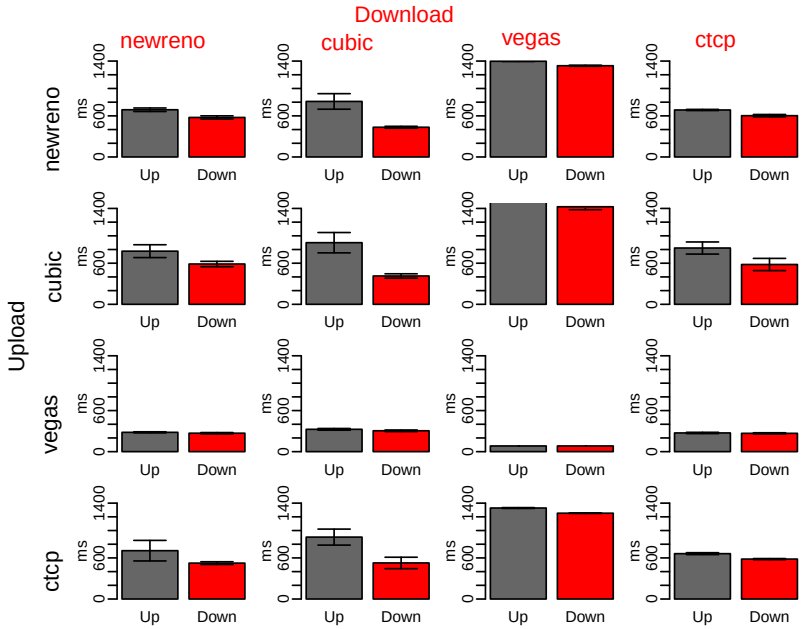


FIGURE 5-6: RTT observé par les connexions TCP et écart type pour un tampon montant surdimensionné — 60 paquets montants et descendants — 2.4/.4 Mbps

TCP *New Reno* et *Cubic* ne sont pas les solutions optimales pour ce type de configuration étant donné qu'elles tendent à remplir les files d'attente, au contraire des solutions basées sur le délai qui maintiennent l'occupation du tampon basse. Cet effet a été observé récemment dans une étude sur les effets de l'utilisation de tampons excessivement grands dans les réseaux mobiles. Dans

cette étude, TCP *Vegas* garde les files d'attente faiblement remplies, en contraste complet avec les autres variantes[90]. De même, ici, TCP *Vegas* permet d'atteindre des débits descendants supérieurs (proches de 100% du lien) aux autres variantes lorsque utilisé sur le lien montant.

En effet, si TCP *Vegas* garde les tampons vides, le balancier de données ne peut avoir lieu. Toutefois, sa fenêtre de congestion est ajustée uniquement en fonction du RTT, ce qui a pour conséquence de faibles RTT, mais aussi des débits extrêmement réduits.

Dans cette situation, TCP *Vegas* est inadapté à une utilisation sur le lien descendant, mais lorsqu'il est utilisé pour l'*upload*, le *download* en bénéficie fortement. En conclusion, *Vegas* est un très mauvais choix côté serveur, mais reste un bon candidat pour le client car il favorise la réactivité du téléchargement.

Bien que la fenêtre de congestion de TCP *Compound* contienne une composante basée sur le délai, les résultats sont similaires à ceux obtenus avec *Cubic* ou *New Reno*. Outre le fait que le produit délai-bande passante est probablement trop faible pour que *Compound* sorte du "mode *New Reno*", cette composante joue sur l'accélération de la croissance de la fenêtre. La connexion est donc plus agressive jusqu'à commencer à remplir le tampon, moment à partir duquel l'algorithme se comporte comme TCP *New Reno*.

La Figure 5-6 montre les RTT des connexions lorsque deux tampons identiques sont utilisés avec un lien asymétrique. Lorsque *New Reno* ou *Compound* sont utilisés face à *New Reno* ou *Compound*, le ratio entre les RTT des connexions est similaire au cas de tampons correctement configurés. Lorsque TCP *Cubic* est utilisé face TCP *New Reno* ou *Compound*, le RTT du *download* augmente légèrement, tandis que celui de l'*upload* explose. Cette différence de RTT est encore plus notable lorsque *Cubic* est utilisé sur les liens montant et descendant. Ces différences proviennent du fait que les délais atteints sont de l'ordre de la seconde et peuvent bloquer la transmission de paquets en *upload* lors des épisodes de congestion (c'est-à-dire lorsque les délais sont les plus élevés).

TCP *Vegas* présente des RTT identiques sur le lien montant à ceux observés lors de l'expérience précédente. En effet, même si le tampon montant est plus grand, *Vegas* n'essaie pas de le remplir, et les bascules ne peuvent avoir lieu. Lorsque cette variante est utilisée sur le lien descendant les résultats sont désastreux : les autres algorithmes tentent (et réussissent) en effet de remplir le tampon montant, causant de longs délais qui impactent particulièrement *Vegas*.

Sur la Figure 5-7, les débits sont représentés pour un lien asymétrique avec 10 Mb/s - 250 paquets down et 2 Mb/s - 250 paquets up. Les résultats sont identiques bien qu'accentués en comparaison de l'expérience précédente. En effet, lorsque *New Reno* ou *Compound* sont utilisés sur le lien montant, l'*upload* atteint une forte utilisation du lien tandis que le *download* n'occupe qu'entre 20 et 40% du lien descendant.

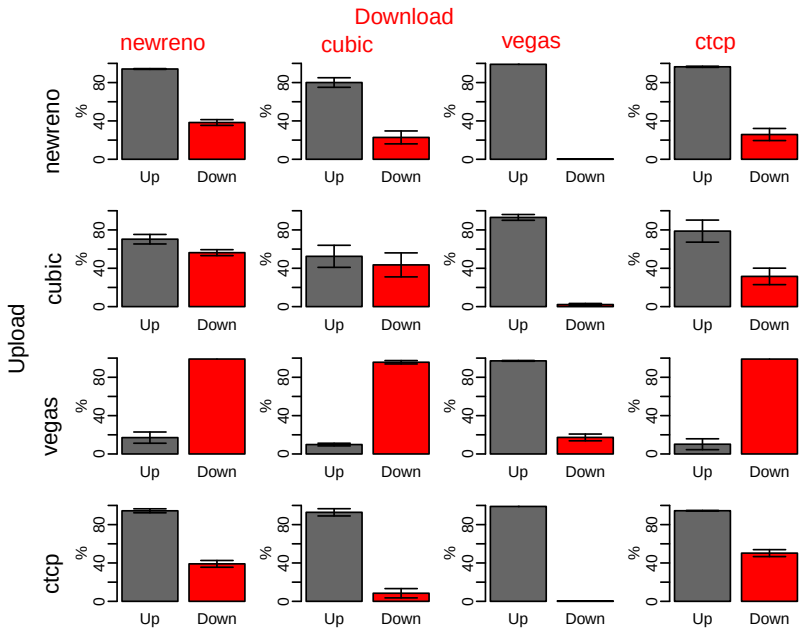


FIGURE 5-7: Utilisation du lien (%) et écart type pour un tampon montant surdimensionné — 250 paquets montants et descendants — 10/2 Mbps

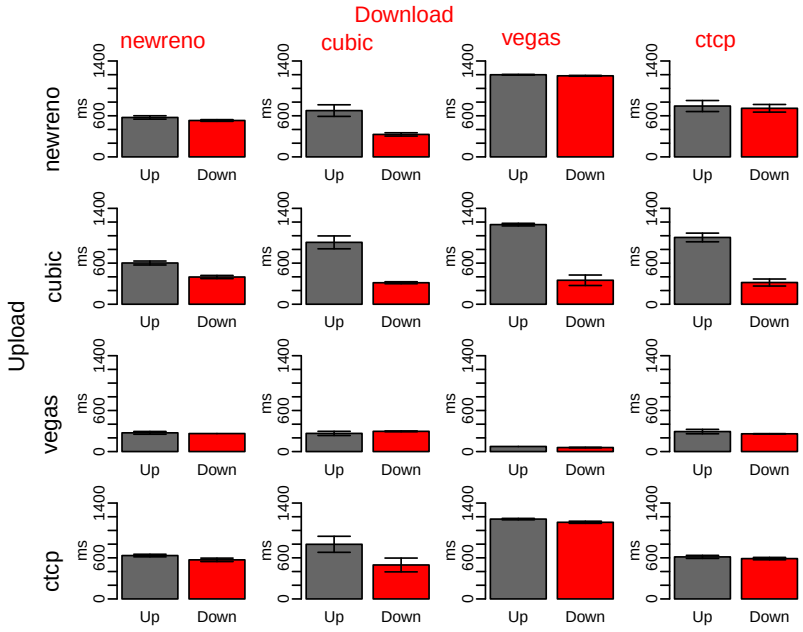


FIGURE 5-8: RTT observé par les connexions TCP et écart type pour un tampon montant surdimensionné — 250 paquets montants et descendants — 10/2 Mbps

Au contraire, si TCP *Vegas* est utilisé sur le lien montant, l'*upload* est sévèrement écrasé par le *download*, qui atteint presque 100% du lien descendant. De manière similaire à la section précédente, TCP *Vegas* atteint les mêmes débits absolus quelle que soit la taille du lien.

TCP *Cubic* donne des résultats assez surprenants lorsqu'utilisé sur le lien montant. Dans ce scénario, l'*upload* et le *download* souffrent du tampon surdimensionné sur le lien montant. Ici, le produit délai-bande passante est beaucoup plus élevé, et *Cubic* passe donc plus de temps dans le "mode *Cubic*", qui est beaucoup plus agressif. Le tampon se remplit donc plus rapidement, ce qui donne une occupation moyenne plus élevée, mais également des pertes plus nombreuses, qui empêchent la connexion d'atteindre une occupation du lien satisfaisante.

En ce qui concerne les RTT présentés Figure 5-8, d'autres différences apparaissent. Tout d'abord, les RTT sont similaires à l'expérience précédente pour *New Reno*, *Vegas* ou *Compound* sur les liens montants et descendants. En revanche, la différence de RTT entre l'*upload* et le *download* lors de l'utilisation de *Cubic* est accentuée.

5.4 Temps de réponse des téléchargements

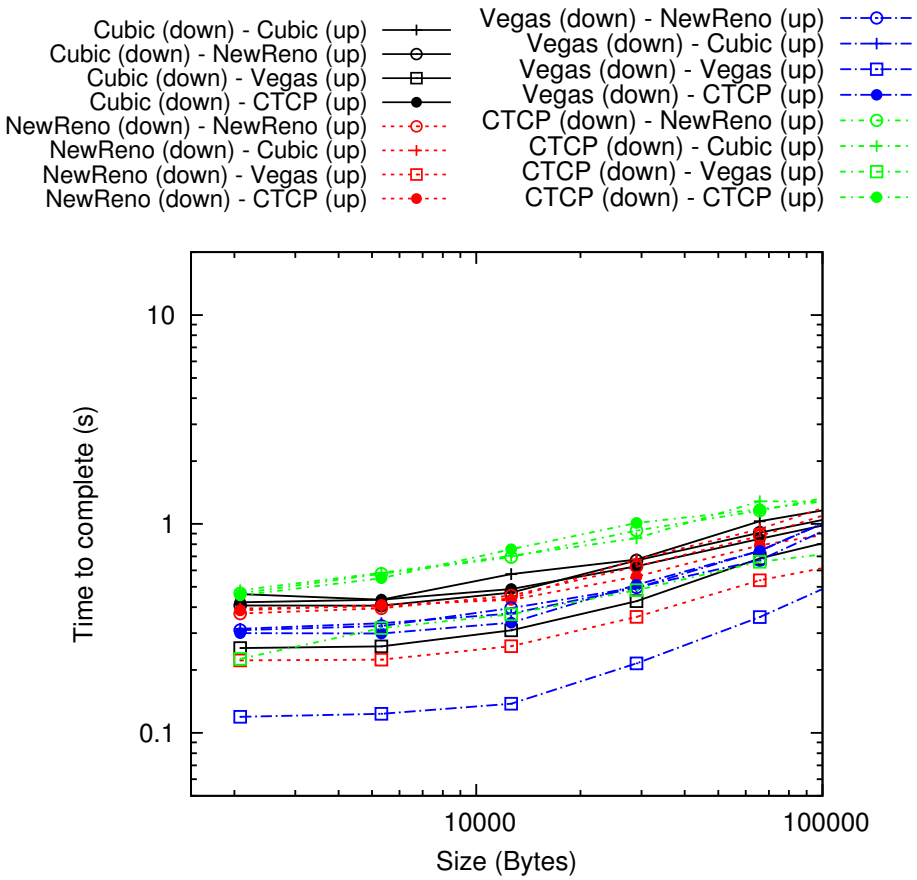


FIGURE 5-9: Temps de réponse conditionnels des connexions pour un trafic poissonien descendant et une connexion montante. Charge à 50%. Tampon correctement dimensionné : 10 Mb/s - 250 pkts down 2 Mb/s - 40 pkts up.

Afin de tester les diverses variantes de TCP dans un environnement plus réaliste, les interactions

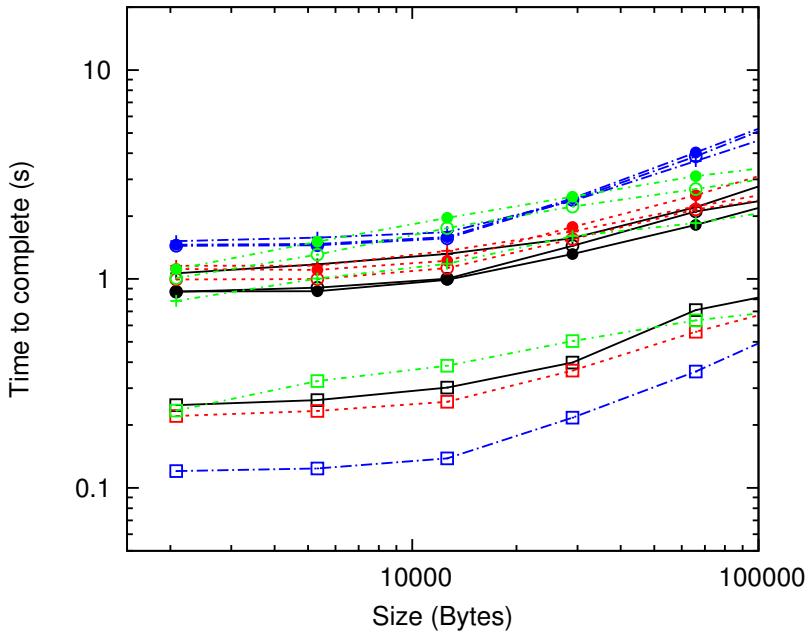


FIGURE 5-10: Temps de réponse conditionnels des connexions pour un trafic poissonien descendant et une connexion montante. Charge à 50%. Tampon surdimensionné : 10 Mb/s - 250 pkts down 2 Mb/s - 250 pkts up.

entre un *upload* long (1000 s) et une charge descendante complexe ont été étudiées. Ce scénario correspond par exemple à un utilisateur qui transmet des fichiers *dans le cloud* tout en surfant sur Internet. La charge descendante a été composée d'une manière simple : les connexions arrivent selon un processus de Poisson, tandis que leur taille suit une distribution à queue lourde. Le processus de Poisson a une intensité $\lambda = 0.15\, s^{-1}$, tandis que les tailles des transferts suivent une distribution de Zipf de paramètre $\alpha = 1.7$.

Cette charge synthétique correspond à une occupation descendante d'environs 50% avec 6662 connexions de taille moyenne 59 paquets, à cause de la présence de quelques flux longs. 4981 connexions font 5 paquets ou moins. Cette charge a été générée sur le deuxième lien décrit Table 6.3 (10 Mb/s - 2 Mb/s).

L'objectif de cette expérience est de capturer le ralentissement qu'un utilisateur peut rencontrer lorsqu'il navigue sur Internet alors qu'un *upload* (*cloud*, *torrent*, *ftp*) est actif. Il est important de noter que les *uploads* ont tendance à durer plus longtemps que les *downloads* en raison de l'asymétrie du lien, ce qui accentue encore le problème.

En général, les temps de réponse présentés dans les Figures 5-9 et 5-10 reflètent les résultats obtenus pour des connexions longues, avec des temps de réponse qui peuvent doubler pour des tampons surdimensionnés.

Lorsque *Vegas* est utilisé pour l'*upload* (points "carrés"), les temps de réponse en *download* sont bas,

ce qui est cohérent avec la section précédente, car les *downloads* se retrouvent face à des tampons essentiellement vides sur la voie de retour, et se terminent donc bien plus tôt. De plus, la Figure 5-9 amène à des conclusions bien différentes de la Figure 5-1 : pour des transferts courts à moyens, c'est-à-dire lorsque le *slow start* influence encore les performances de manière significative, les temps de réponse atteints par *Vegas* sont sensiblement similaires à ceux des autres variantes, à l'exception du cas dans lequel *Vegas* est utilisé à la fois sur les liens montant et descendant, qui présente les temps de réponse les plus bas mesurés. Le fait de simplement regarder les flux longs comme dans la section précédente aurait totalement occulté ce phénomène, là où dans un contexte plus proche de la réalité, *Vegas* présente des performances supérieures à ses concurrents. Enfin, lorsque *Vegas* est utilisé sur le lien montant, la taille du tampon a peu, voire aucun impact. En effet, comme *Vegas* cherche à minimiser l'utilisation du tampon, la file d'attente montante voit principalement des ACKs, *i.e.* des petits paquets qui sont rapidement envoyés sur le réseau, et quelques paquets de données. Cette file d'attente a donc plus de mal à se remplir. De plus, si cette situation arrive dans le cas d'un tampon surdimensionné, un délai négligeable est généré en comparaison avec celui obtenu dans le cas d'un tampon bien dimensionné.

Curieusement, *Cubic* et *Compound* utilisés sur le lien descendant donnent des temps de réponse plus bas que *New Reno* (20%) et *Vegas* (25%) lorsque le tampon montant est surdimensionné, avec de très bons résultats lorsque *Compound* est utilisé sur le lien descendant et *Cubic* sur le lien montant (nous en profiterons pour savourer l'ironie de la chose, les statistiques montrent que la situation inverse est beaucoup plus plausible [93]). Au contraire, lorsque ce tampon est bien dimensionné, TCP *Vegas* donne de très bons temps de réponse (50% meilleurs que *Compound*, et 20% inférieurs à *New Reno* et *Cubic*).

Ces deux expériences ont été retentées avec une charge plus élevée (80%), avec des résultats sensiblement similaires.

Conclusion

L'étude des variantes les plus populaires de TCP confirme les conclusions établies de manière théorique au chapitre précédent : l'effet de balancier n'apparait que lorsque la transmission est régulée par des algorithmes basés sur le cadencement des ACKs. Les solutions qui ne saturent pas le tampon montant telles que TCP *Vegas* permettent quant à elles d'éviter les délais excessifs dans la file. Ainsi l'utilisation de *Vegas* atténue les effets que peuvent avoir des tampons surdimensionnés, au prix d'un débit en *upload* réduit.

Une autre solution réside dans la modification des mécanismes de gestion des files d'attente dans le réseau, afin de réduire les interférences entre *upload* et *download*, que ce soit par une simple politique

de *fair queuing* par connexion, ou, lorsque cette solution n'est pas disponible, en dimensionnant correctement les tampons. Enfin, certaines politiques ont été spécialement développées dans le but de minimiser les délais dus aux tampons, la plus connue étant CoDel. Ces deux dernières solutions ont été adoptées par le fournisseur d'accès à Internet *Free* : l'algorithme FQ_CoDel, qui combine *fair queuing* et faibles latences est ainsi déployé sur les tampons montants de ses routeurs ADSL [94].

Ces deux derniers chapitres ont également démontré que, dans le cas de tampons montant surdimensionnés sur des liens asymétrique, le cadencement des ACKs reste l'élément limitant, loin devant la variante utilisée : sur toutes les expériences effectuées, les différences entre TCP *New Reno*, *Compound* et *Cubic* sont le plus souvent de l'ordre de quelques pourcents, parfois au bénéfice de TCP *New Reno*.

Chapitre 6

Le paradoxe des petits paquets ou comment améliorer la latence en ajoutant du trafic ?

You let one ant stand up to us, then they all might stand up ! Those puny little ants outnumber us a hundred to one and if they ever figure that out there goes our way of life !

Hopper – A Bug's Life

Introduction

Les deux chapitres précédents portaient sur l'analyse de l'effet de balancier qui apparaît lorsque deux connexions antiparallèles partagent un lien. Si le fait d'employer un algorithme de contrôle de congestion basé sur le cadencement des ACKs dans des tampons FIFO suffit à déclencher cet effet, il ne devient problématique que lorsque le tampon est surdimensionné.

Or, lorsque de grands tampons sont présents dans les routeurs, les connexions peuvent subir de fortes latences, phénomène connu sous le nom de *bufferbloat* [95, 96]. Bien qu'il n'existe pas à notre connaissance d'étude établissant avec précision l'étendue du phénomène au sein des réseaux actuels [97], sa présence peut avoir des conséquences dévastatrices sur l'expérience utilisateur, notamment dans le cas de trafic "temps réel". Au cours de ce chapitre, nous étudierons les solutions que l'utilisateur final a à sa disposition afin de pallier ces effets.

Prenons pour exemple un lien montant résidentiel à 400 kb/s, avec un tampon montant de 50 paquets : transmettre 50 paquets (soit 75 KB) prend 1.5 s. À titre de comparaison, le *timeout* par défaut pour les SYN TCP (Linux) et des requêtes DNS est d'une seconde. Ainsi, un hôte peut retransmettre des segments avant même que le paquet original n'ait quitté le tampon. De plus, comme nous avons pu le voir dans les deux chapitres précédents, un tampon montant surdimensionné affecte fortement les débits descendants.

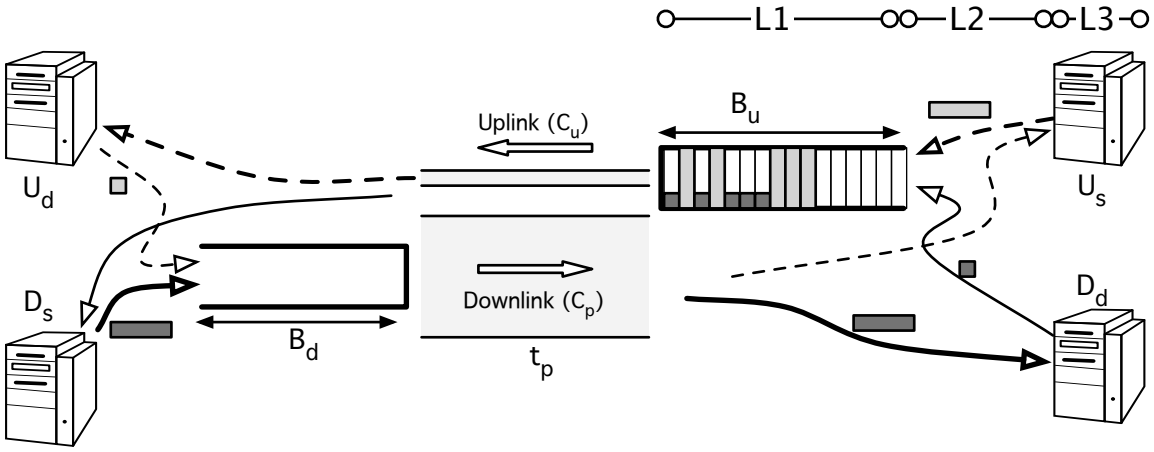


FIGURE 6-1: Un tampon de taille excessive (B_u) sur le lien montant mène à de fortes latences lorsque le tampon contient des grands paquets. Lorsque le trafic est bidirectionnel, petits et grand paquets se mélangent dans le tampon.

De nombreuses solutions ont été étudiées afin de résoudre ce problème. Néanmoins, ces dernières sont au mieux complexes à déployer, et le plus souvent inaccessibles à l'utilisateur final.

La plus efficace d'entre elles consiste à changer la politique de file d'attente à l'entrée du lien montant, par exemple en donnant plus de priorité aux petits paquets (messages de contrôle), au niveau **L1** de

la Figure 6-1. Toutefois, cette pratique peut entraîner des déséquencements dans le flux de données ; de plus, cette solution demande un accès aux routeurs, accès souvent dépendant de l'opérateur réseau, et donc fermé à l'utilisateur final.

Une solution consiste à rapprocher le goulot d'étranglement de la source de l'*upload*, où il est possible de configurer la politique de file d'attente (représenté au niveau **L2** Figure 6-1). Cette méthode reste toutefois contraignante car elle requiert d'estimer la bande passante du goulot d'étranglement en temps réel, tout en sacrifiant une part non négligeable du débit maximum. Le trafic montant doit en effet être transmis à un rythme inférieur à la capacité du goulot d'étranglement, afin de garder le tampon vide. Pour ce faire, le goulot d'étranglement nouvellement créé doit être configuré pour avoir un tampon plus petit, ou être régi par une politique autre que FIFO.

Il est aussi possible d'agir directement au niveau de la source (**L3**) en évitant de saturer le lien montant. Ce chapitre se focalise sur ce type de solutions qui, contrairement aux autres approches, sont sous le contrôle de l'utilisateur final.

Une des solutions possibles à ce niveau réside dans l'utilisation d'un algorithme de contrôle de congestion comme TCP *Vegas* pour limiter l'occupation de la file d'attente. Comme *Vegas* tente de minimiser la quantité de données présente dans les files d'attente, il n'est pas sensible à la taille du tampon, toutefois au prix de débits sévèrement réduits.

Une autre approche peut être envisagée : il est possible de réduire la portion des délais due aux tampons en s'assurant qu'une fraction significative des emplacements contient des petits paquets. En effet, les files d'attente sont en majorité dimensionnées en paquets. À occupation égale, le temps passé dans la file est fonction de la taille des paquets ; la présence de petits paquets réduit donc les latences observées à position égale dans la file.

Ces petits paquets peuvent apparaître comme un effet de bord d'autres types de trafic, par exemple les ACKs TCP générés par une connexion descendante. L'influence bénéfique des ACKs d'une connexion descendante sur le délai peut expliquer pourquoi le *bufferbloat* n'est pas nécessairement aussi important qu'imaginé [97]. Les tampons surdimensionnés sont en effet chose commune, mais la manière dont le réseau est utilisé peut diminuer leur impact.

Ce chapitre propose une solution pour réduire les délais du réseau, par l'envoi d'une charge synthétique de petits paquets dans le tampon montant. Un tel mécanisme peut apparaître paradoxal, mais sa validité est démontrée à travers l'analyse du phénomène ainsi qu'un jeu d'expériences significatif. Les résultats observés montrent une amélioration significative des délais, au prix d'une baisse de débit négligeable.

6.1 Ajouter du trafic pour diminuer les délais ?

La plupart du temps, les tampons sont structurés comme des tableaux de paquets, quelle que soit leur taille, c'est-à-dire que le tampon déborde lorsqu'un certain nombre de paquets, et non d'octets, est présent dans la file. Le temps passé dans la file par un paquet dépend donc fortement de la taille des segments présents, comme illustré Figure 6-2.

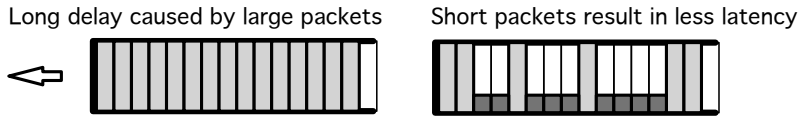


FIGURE 6-2: Les petits paquets améliorent la latence réseau

6.1.1 Expérience préliminaire

Considérons un modem ADSL standard (BeWAN 700G) avec un tampon montant de 30 paquets, connecté à un lien asymétrique de 3815 kb/s descendants et 854 kb/s montants au niveau ATM. La Table 6.1 présente les performances mesurées avec et sans trafic additionnel.

TABLE 6.1: Expérience préliminaire

	Sans UDP	Avec UDP
Délai dans la file	346±84 ms	124±50 ms
Débit TCP montant	728 kbit/s	660 kbit/s
Débit d'un flux TCP unique	3.26 Mb/s	
Débit d'un *download* TCP en concurence avec un *upload* TCP	2.80 Mb/s	3.06 Mb/s

La latence causée par les files d'attente est tout d'abord mesurée pour un seul *upload* TCP. L'envoi de paquets UDP au rythme de 125 paquets par seconde (1 octet de charge) fait légèrement chuter le débit de l'*upload* (660 Kb/s contre 728 Kb/s). En revanche, le délai dans la file est divisé par deux (124 ms contre 346 ms).

Ensuite un *download* TCP est introduit. Ce flux obtient 3.26 Mb/s lorsque seul sur le lien. Face à un *upload*, ce débit chute à 2.8 Mb/s, et atteint 3.06 Mb/s lorsque du trafic UDP est ajouté sur le lien montant, soit un gain de près de 10%. Si le gain en débit pour le *download* est faible, la latence quant à elle est considérablement réduite.

L'ajout d'un flux de petits paquets UDP permet donc de diminuer et stabiliser le délai sans pour autant altérer de manière significative le débit. Les petits paquets permettent également de réduire l'impact de l'*upload* sur le *download* constaté au cours des deux chapitres précédents. Ces deux effets combinés améliorent fortement la responsivité globale du réseau.

6.1.2 Hypothèses et notations

Les hypothèses suivantes sont faites :

Le tampon montant est suffisamment grand pour que les algorithmes de contrôle de congestion saturent le lien (le cas inverse entraîne un autre type de perturbations qui ne seront pas traitées dans ce chapitre).

De même, on considèrera que les tampons TCP montants et descendants sont infinis, et donc que les fenêtres de congestion peuvent atteindre des valeurs suffisamment grandes pour remplir le lien.

Enfin, les notations suivantes seront considérées :

- Une connexion TCP envoie des données à un rythme D_{T}, avec une taille de paquets S_{T}, et un rythme d'envoi en paquet $R_{\mathrm{T}} = \dfrac{D_{\mathrm{T}}}{S_{\mathrm{T}}}$,
- Un flux UDP a pour débit d'émission D_{U}, une taille de paquets S_{U}, et un débit en paquets $R_{\mathrm{U}} = \dfrac{D_{\mathrm{U}}}{S_{\mathrm{U}}}$,
- $S_{\mathrm{U}} \ll S_{\mathrm{T}}$ et $D_{\mathrm{U}} \ll D_{\mathrm{T}}$,
- Les connexions TCP sont en *congestion avoidance*, et donc R_{T} est constant sur un RTT.
- C_u, la capacité du lien montant du goulot d'étranglement.
- $Q(t)$, l'occupation instantanée du tampon en paquets à un instant t.

6.1.3 Une seule connexion TCP

La croissance de la file sur une durée Δt petite devant un RTT est égale à la différence entre le nombre de paquets arrivés et la transmission des données sur le lien pendant cet intervalle :

$$\Delta Q_{\mathrm{T}} = R_{\mathrm{T}}\Delta t - \frac{C_u}{S_{\mathrm{T}}}\Delta t \tag{6.1}$$

6.1.4 Flux TCP et UDP en parallèle

Pour les deux flux, la croissance de la file est régie par l'expression suivante :

$$\Delta Q_{\mathrm{T,U}} = Q_{\mathrm{T,U}}(t+\Delta t) - Q_{\mathrm{T,U}}(t) = (R_{\mathrm{T}} + R_{\mathrm{U}})\Delta t - \frac{C_u}{S}\Delta t, \tag{6.2}$$

où $\dfrac{C_u}{S}$ est le débit auquel les paquets quittent la file (en paquets par seconde), et S est la taille moyenne des paquets.

$$S = \frac{R_{\mathrm{U}}S_{\mathrm{U}}}{R_{\mathrm{U}} + R_{\mathrm{T}}} + \frac{R_{\mathrm{T}}S_{\mathrm{T}}}{R_{\mathrm{U}} + R_{\mathrm{T}}} \tag{6.3}$$

Comme $D_{\mathrm{U}} = R_{\mathrm{U}} S_{\mathrm{U}} \ll D_{\mathrm{T}} = R_{\mathrm{T}} S_{\mathrm{T}}$, $S \approx \dfrac{D_{\mathrm{T}}}{R_{\mathrm{U}} + R_{\mathrm{T}}}$ et

$$\Delta Q_{\mathrm{T,U}} = (R_{\mathrm{U}} + R_{\mathrm{T}}) \Delta t \ (1 - \frac{C_u}{D_{\mathrm{T}}}) \tag{6.4}$$

Donc,

$$\Delta Q_{\mathrm{T,U}} = \frac{(R_{\mathrm{U}} + R_{\mathrm{T}})}{R_{\mathrm{T}}} \Delta Q_{\mathrm{T}} \tag{6.5}$$

Cette expression indique que grâce au trafic UDP additionnel, l'occupation du tampon augmente d'une proportion $\dfrac{R_{\mathrm{U}}}{R_{\mathrm{T}}}$ par rapport au cas d'un unique flux TCP. Si le tampon déborde plus tôt, la fenêtre de congestion TCP atteint des valeurs maximales plus faibles, et le temps passé par un paquet dans la file diminue en conséquence.

Afin de diviser les délais dans la file par un facteur $\alpha = \dfrac{\Delta Q_{\mathrm{T,U}}}{\Delta Q_{\mathrm{T}}}$, le nombre de paquets UDP envoyés par seconde doit être égal à :

$$R_{\mathrm{U}} = R_{\mathrm{T}}(\alpha - 1), \tag{6.6}$$

Sachant que dans tous les cas, le rythme d'envoi de paquets TCP est égal à : $R_{\mathrm{T}} \approx \dfrac{C_{\mathrm{u}}}{S_{\mathrm{T}}}$.

La surcharge causée par la couche MAC a été négligée dans les calculs ci-dessus. Sur un lien Ethernet, par exemple, la trame la plus petite possible est de 64 octets auxquels sont rajoutés le préambule et l'espace inter-trame (20 octets). Le ratio entre des segments TCP de 1500 octets et des petits paquets UDP est ainsi de 18 pour 1.

Dans un cas idéal, le paquet UDP le plus petit possible embarque 1 octet de données, pour un total de 29 octets en incluant les en-têtes IP et UDP. Le ratio des tailles est ainsi de 52 pour 1. Il est donc possible de générer de nombreux paquets UDP par segment TCP pour un coût négligeable.

L'*overhead* généré par le trafic UDP additionnel en fonction de la latence visée est donc égal à :

$$\mathrm{OH}_{\mathrm{U}} = \frac{1}{1 + \frac{S_{\mathrm{T}}}{S_{\mathrm{U}}} \frac{1}{\alpha - 1}}. \tag{6.7}$$

Ainsi, le délai peut théoriquement être divisé par 10 pour une perte de 15% de la capacité. [1].

En général, le débit UDP requis est estimé en fonction du débit en paquets de TCP au goulot d'étranglement. Envoyer des paquets UDP au même rythme permet ainsi de diviser la latence par deux. Si le rythme des paquets TCP et/ou la taille de la file d'attente ne sont pas accessibles, il est possible de prendre pour approximation un tampon de 50 paquets et d'augmenter ou diminuer le débit UDP en observant la latence, de manière manuelle ou automatique.

1. Ceci est le cas pour des liens ADSL qui n'utilisent pas PPPoE par exemple.

Dans les sections suivantes, cette dernière problématique sera négligée au profit de la validation et de l'analyse des effets du trafic UDP additionnel.

6.1.5 Envoyer des rafales de paquets UDP

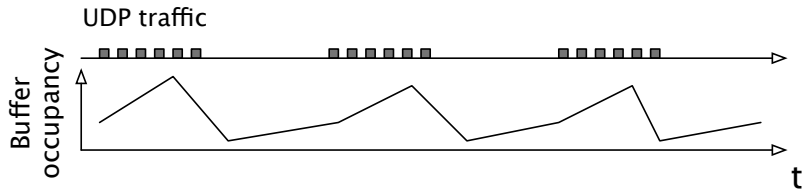


FIGURE 6-3: Envoyer des rafales de paquets UDP est suffisant pour garder l'occupation du tampon à un niveau bas.

De par le fonctionnement de la phase de *congestion avoidance*, les connexions TCP ne saturent un tampon qu'après plusieurs RTT. Lorsque le tampon déborde, le trafic UDP qui a contribué à cette perte perd alors sa pertinence pour plusieurs RTT (soit plusieurs centaines de millisecondes).

Il est donc suffisant d'envoyer le trafic additionnel par rafales cadencées de paquets UDP. Typiquement, les paquets au sein d'une rafale ont le même débit que ceux envoyés dans le cas de trafic continu, pour un effet similaire, mais sont envoyés avec des pauses de l'ordre de la seconde (cf. Figure 6-3).

6.2 Trafic sur la voie de retour et tampons excessifs

Alors que l'ajout de trafic UDP est un moyen d'améliorer la latence, les flux TCP peuvent avoir un effet similaire. Par exemple, un *download* TCP continu génère un flux d'acquittements dans le tampon montant, ce qui réduit ainsi sa taille effective de manière virtuelle.

Bien que nous ayons vu dans les chapitres précédents que l'*upload* peut nuire au *download*, il a aussi été montré que les interférences ont moins de chances d'apparaître lorsque les tampons sont structurés comme des tableaux de paquets et que l'option *delayed ACK* est désactivée, ce qui augmente la fraction de petits paquets dans les tampons du goulot d'étranglement. Ainsi, on conseillera dans ce type de situations de désactiver l'option *delayed ACK*, car elle peut se révéler contre productive.

De manière similaire, un trafic intense de petites connexions TCP peut diminuer les latences causées par la file d'attente. Pour analyser cet effet, considérons une connexion TCP composée d'une courte requête et d'un seul segment de données (voir Figure 6-4). Cette connexion a un paquet dans le tampon à chaque instant pendant 3 à 4 RTT si les autres délais sont négligés.

Pour un usage de type navigation web, un modèle de trafic simple comme décrit dans [98] considère

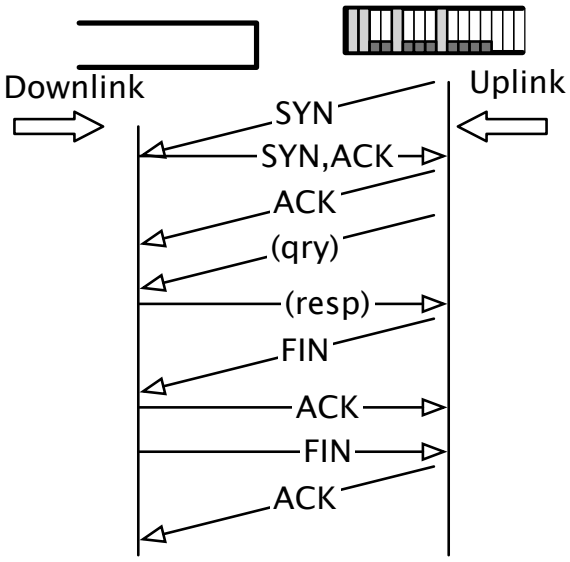


FIGURE 6-4: Chaque connexion TCP courte génère de petits paquets qui occupent le tampon de retour pendant 3 à 4 RTT

que l'établissement de la connexion suit un processus de Poisson d'intensité $\lambda = 2\,s^{-1}$. Le RTT peut être estimé aux alentours de 500 ms (ce qui peut rapidement arriver lorsque le tampon montant est surdimensionné). Ainsi, $3\lambda RTT = 3$ connexions sont établies pendant 3 RTTs, et chacune aura un paquet dans le tampon pendant 3 à 4 RTT.

Dans le scénario le plus simple pour lequel toutes les connexions sont initiées par un des deux hôtes et sont constituées d'un seul paquet de données sans réponse de la part de l'autre hôte, il y a en permanence un nombre moyen de 3 accusés de réception dans le tampon montant.

De plus, un nombre non négligeable de connexions sont de taille supérieure à 1 paquet. Un nombre supérieur d'emplacement dans la file sont donc occupés par des petits paquets. La taille de la file d'attente en est d'autant plus virtuellement réduite, et 3 emplacements occupés par des petits paquets représente la limite basse du modèle.

Cet effet peut être exacerbé si le lien ADSL est partagé, par exemple, au sein d'une famille. L'arrivée des connexions dans le goulot d'étranglement est alors déterminée par une superposition de plusieurs processus de Poisson. Cette superposition est elle-même un processus de Poisson de paramètre $\lambda = \sum_n \lambda_n$, ce qui signifie plus d'ACKs dans la file et donc un tampon plus petit du point de vue des connexions.

En conséquence, le trafic descendant cause souvent des rafales de petits paquets qui empêchent la fenêtre de congestion montante de grandir de manière excessive, pour des effets similaires à l'envoi de trafic UDP additionnel. Les expériences réalisées ci-dessous reflètent ce phénomène.

6.3 Expériences en conditions réelles

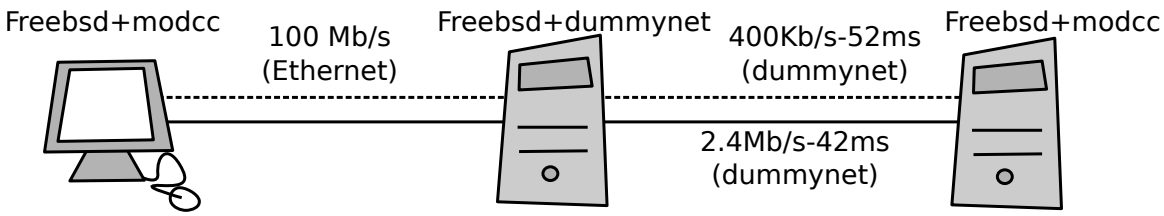


FIGURE 6-5: Banc d'essai

Une série d'expériences a été lancée sur un banc d'essai composé de trois ordinateurs (*cf.* Chapitre 3). Les conditions sont similaires à celles évoquées dans le Chapitre 5. Le routeur et les hôtes tournent sous FreeBSD 9.1 ce qui permet de tester les variantes les plus populaires de TCP sans avoir à changer de système d'exploitation. De plus ceci procure l'assurance d'utiliser la même pile TCP pour chaque expérience, et donc de n'observer que les variations dans les algorithmes de contrôle de congestion. Sur le routeur, le goulot d'étranglement est émulé à l'aide du module Dummynet. Les deux hôtes utilisent un noyau *vanilla* tandis que celui du routeur a été recompilé pour une commutation de contexte plus rapide (option HZ à 10000).

Tout d'abord, le modèle et le banc d'essai ont été validés au travers d'une simple expérience. Le routeur émule le lien asymétrique suivant : bande passante à 5 Mb/s, 42 ms de délai, et un tampon de 120 paquets pour le lien descendant, et 800 Kb/s, 52 ms de délai et 120 paquets pour le lien montant. Dans ce lien, un *upload* TCP est généré en parallèle d'un flux UDP à 250 pkts/s. Les résultats obtenus sont ensuite comparés aux débits et latences mesurés pour un flux TCP seul sur le réseau. Les résultats apparaissent dans la Table 6.2.

TABLE 6.2: Validation

	Sans UDP	Avec UDP
débit TCP	770 kbit/s	657 kbit/s
Utilisation du lien	100%	85%
RTT	280 ms	113.7 ms

Les résultats obtenus confirment les conclusions de l'expérience décrite Section 6.1.1 : pour une faible perte de débit, le RTT est divisé par presque 2.5.

La prochaine section se focalise sur un modèle de trafic plus réaliste : tout d'abord une charge composée de connexions longues et bidirectionnelles, puis une charge aléatoire, composée de plusieurs connexions de taille variable.

TABLE 6.3: Paramètres de la première expérience

$\tau_{download}$	42 ms	42 ms
τ_{upload}	52 ms	52 ms
C_p	288 kB/s	600 kB/s
	(2.4 Mb/s at L2)	(5 Mb/s at L2)
C_u	48 kB/s	96 kB/s
	(400 kb/s at L2)	(800 kb/s at L2)
Tampon montant	10 and 60 packets	20 and 120 packets
Tampon descendant	60 packets	12 packets
Variante TCP	*New Reno*	*New Reno*

6.3.1 Flux longs

Quatre expériences sont exécutées sur le banc d'essai configuré avec les deux liens Table 6.3.

Tout d'abord, un tampon surdimensionné est considéré. Selon la règle empirique donnée au Chapitre 2 le tampon doit pouvoir contenir une quantité de données égale au produit délai-bande passante. Ici, ce dernier est 6 fois plus grand que la taille recommandée.

Un *download* long est généré suivi par un *upload* après un temps aléatoire compris entre 20 et 40 secondes. Lors d'une deuxième expérience, un flux UDP de 250 paquets par seconde sur le lien montant est ajouté aux conditions précédentes. Par la suite, la simulation est réalisée avec un flux UDP intermittent sur le lien montant. Une rafale est composée de 100 paquets à 250 paquets/s et est envoyée chaque seconde. Enfin, la première expérience est relancée avec un tampon montant correctement dimensionné (10 paquets, ce qui permet d'atteindre un délai montant maximum de 300 ms). La suite `ipmt` [91] est utilisée pour générer les connexions, `tcpdump` et `tcptrace` servent respectivement à l'observation et l'analyse du trafic.

Le comportement de TCP est observé lorsque les deux connexions ont atteint la phase de *congestion avoidance* et que leur fenêtre de congestion se sont stabilisées, c'est-à-dire que les variations passagères dues au démarrage se sont dissipées et que les statistiques (moyenne, variance) sont stables. Pour ce faire, les analyses sont effectuées sur les données échangées entre 50 et 100 s.

Afin d'éviter les variations parasites dues au fait que ces expériences sont réalisées en conditions "proches du réel", chaque réalisation est lancée 10 fois et les valeurs moyennes sont calculées.

6.3.2 Effet du *bufferbloat*

Les Figures 6-6 et 6-7 présentent les résultats des deux expériences. Pour un tampon montant surdimensionné, les effets du *bufferbloat* sont clairement visibles : le RTT atteint 549/605 ms pour le lien le plus lent et 551/600 ms pour le lien le plus rapide. Alors que l'*upload* atteint entre 91 et 92% de la capacité montante, le *download* peine à atteindre 40% du lien descendant.

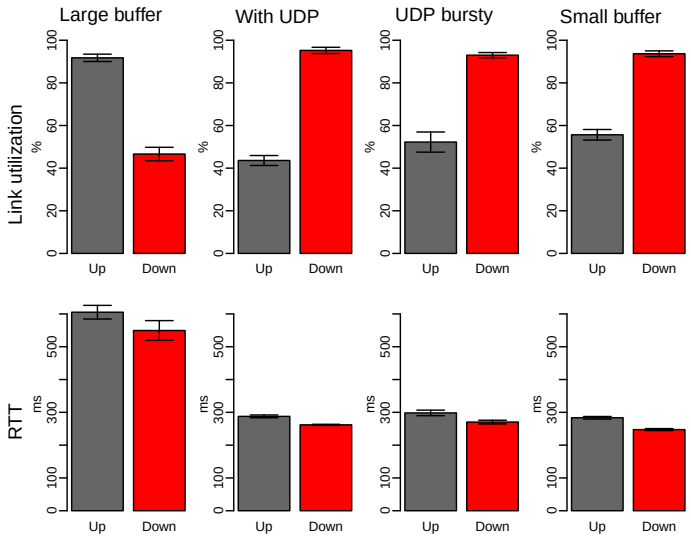


FIGURE 6-6: 1 *download* et 1 *upload* sur un lien asymétrique (2.4 Mb/s *vs.* 400 kb/s). Ajouter du trafic UDP à 250 pkt/s réduit la taille effective du tampon : l'utilisation du lien descendant augmente, et surtout, la latence chute.

Pour un tampon correctement configuré, le RTT est divisé par 2.5 et le *download* atteint 95% de la capacité descendante. L'*upload* quant à lui est retombé autour de 55%.

L'ajout de trafic UDP réduit la taille effective de la file d'attente, ce qui permet d'obtenir des résultats similaires au cas d'un tampon bien dimensionné. En effet, le RTT est divisé par 2.5, et bien que le *download* n'atteigne pas 95% de la capacité du lien, les performances sont nettement améliorées, sans intervention en cœur de réseau.

Le trafic UDP en rafales donne des résultats similaires au flux UDP continu, mais avec 4 fois moins de paquets envoyés sur le réseau.

Il est impossible d'atteindre un RTT inférieur à 300 ms, quelle que soit la configuration étant donné que le tampon descendant est dimensionné pour provoquer un délai maximal de 300 ms.

6.3.3 Temps de réponse du *download*

Les expériences précédentes ont validé le modèle pour du trafic simple. Afin de s'approcher au maximum de conditions réelles, le *download* a été représenté par une charge synthétique aléatoire. L'expérience suivante est réalisée en confrontant un *upload* long (1000 s) à une charge composée de connexions avec des arrivées poissoniennes et une distribution des tailles à queue lourde. Les nouvelles connexions arrivent selon un processus de poisson d'intensité $\lambda = 1\,s^{-1}$ et la taille des transferts suit une distribution de Zipf de paramètre $\alpha = 1.7$. Cette charge synthétique correspond à une occupation du lien descendant d'environ 50% pour le premier lien et 17% pour le deuxième

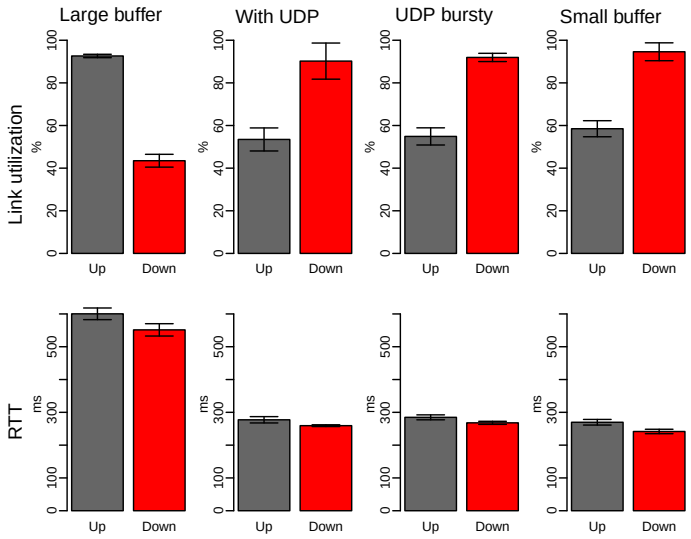


FIGURE 6-7: 1 *download* et 1 *upload* sur un lien asymétrique (5 Mb/s *vs.* 800 kb/s). Effet similaire de l'ajout de trafic UDP à 250 pkt/s : l'utilisation du lien descendant augmente et la latence chute.

avec 1012 connexions de 972 paquets en moyenne (quelques connexions très longues apparaissent).

Bien que similaires à celles présentées dans le chapitre précédent, ces expériences diffèrent dans leur but : il s'agit ici de mesurer l'impact du trafic UDP additionnel sur des connexions de taille finie.

Les expériences sont tout d'abord menées sans trafic UDP. Lors d'une deuxième épreuve, un flux UDP à 125 pkt/s est rajouté à la charge montante.

Les figures 6-8 et 6-10 présentent les temps de réponse conditionnels (en fonction de la taille de la connexion) pour des charges descendantes de 50 et 17% respectivement, tandis que les Figures 6-9 et 6-11 montrent les temps de réponses descendants pour les mêmes charges en présence de trafic UDP.

Les figures 6-8 et 6-10 présentent des résultats sensiblement similaires à ceux décrits au Chapitre 5 : les temps de réponse les plus faibles sont atteints lorsque *Vegas* est utilisé sur le lien montant, tandis que *CuBIC* et *New Reno* présentent des performances identiques.

L'adjonction de trafic UDP (*cf.* Figures 6-9 et 6-11) résulte en des temps de réponses sensiblement plus faibles, similaires aux résultats obtenus lors de l'utilisation de TCP *Vegas* sur le lien montant. Pour une charge de 17%, les temps de réponse sont même divisés par deux dans les cas où *CuBIC* ou *New Reno* sont utilisés sur le lien montant.

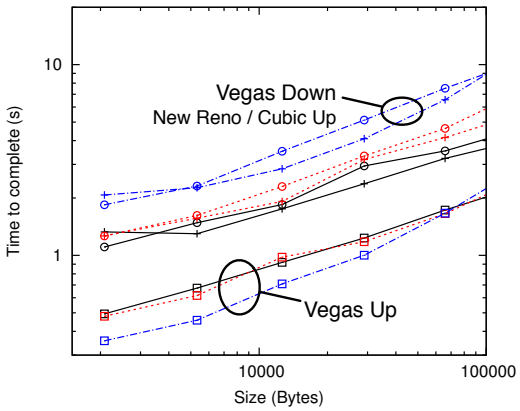


FIGURE 6-8: Temps de réponse conditionnel *en fonction* de la taille du transfert pour une charge descendante de 50% en présence d'un *upload* continu.

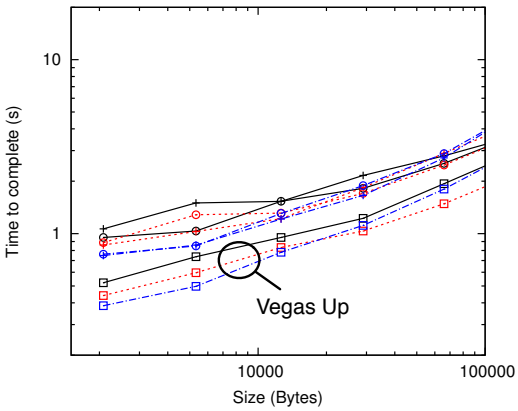


FIGURE 6-9: Temps de réponse conditionnel *en fonction* de la taille du transfert pour une charge descendante de 50% en présence d'un *upload* continu avec du trafic UDP additionnel.

6.4 Trafic réaliste sur ligne ADSL

Pour cette expérience, un *upload* a été effectué vers un serveur TCP distant situé derrière une ligne ADSL réelle. Le lien descendant est de 5440 kbit/s pour 352 kbit/s montants. Sur le serveur, une charge TCP aléatoire est générée selon le même principe que dans la section 6.3.3. Les connexions sont donc générées selon un processus de Poisson d'intensité $1\,s^{-1}$, et leur taille suit une distribution de Zipf de paramètre 1.7.

Cette configuration permet de générer 1061 connexions TCP descendantes, de taille moyenne 660 paquets, et 898 connexions de moins de 10 paquets.

L'expérience est réalisée à l'aide d'un routeur ADSL classique (voir Section 6.1.1). Le tampon du lien montant suit une politique FIFO, et est fixé à 30 paquets (sur le modem ADSL), tandis que la file d'attente du lien descendant est estimée à 60 paquets. Les délais peuvent ainsi atteindre des valeurs proches de la seconde sur le lien montant.

TCP *Vegas*, *New Reno* et *CuBIC* sont utilisés pour l'*upload* et le *download*. Un flux UDP à 125 pkt/s est ensuite rajouté.

L'expérience est réitérée avec une charge de 6 connexions par seconde. Une charge TCP descendante de 5851 connexions est générée, avec une taille moyenne de 413 paquets, et 5002 connexions de moins de 10 paquets.

Les résultats sont présentés sur les Figures 6-12, 6-13 et 6-14. Ces dernières confirment les résultats obtenus Section 6.3.3. Avec *Vegas*, les temps de réponse du *download* sont bien plus faibles que pour les algorithmes de contrôle de congestion. En effet, *Vegas* tente de garder une occupation faible des

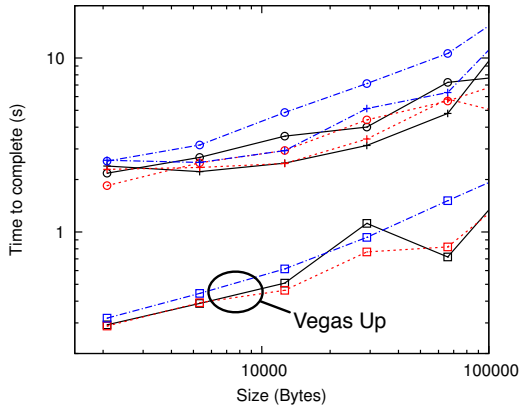


FIGURE 6-10: Temps de réponse conditionnel *en fonction* de la taille du transfert pour une charge descendante de 17% en présence d'un *upload* continu.

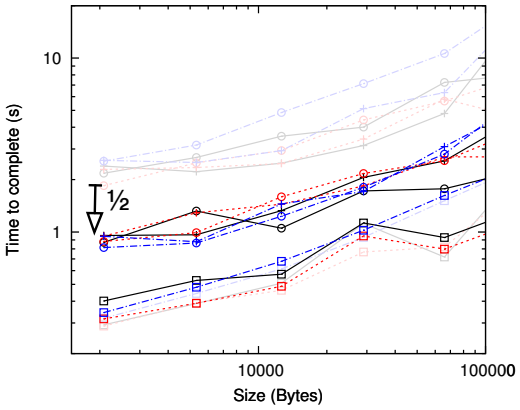


FIGURE 6-11: Temps de réponse conditionnel *en fonction* de la taille du transfert pour une charge descendante de 17% en présence d'un *upload* continu avec du trafic UDP additionnel.

tampons, la file d'attente contient donc principalement du trafic UDP et des ACKs. Ici, le trafic UDP augmente l'occupation de la file d'attente du lien montant alors que *Vegas* la maintient vide. L'impact du trafic UDP n'est ainsi plus négligeable sur l'*upload*. Toutefois, nous avons pu voir dans les chapitres précédents que TCP *Vegas* utilisé sur le lien montant résoud le problème du *bufferbloat*, et rajouter un flux UDP est donc absurde.

Pour un trafic TCP descendant plus intense, TCP *New Reno* et *CuBIC* obtiennent des performances contrastées : *New Reno* pour l'*upload* mène à des temps de réponse bien supérieurs aux autres algorithmes. Ici, l'utilisation de *Vegas* sur le lien montant est toujours bénéfique ; malheureusement, *New Reno* est bien plus fréquent que *Vegas* sur les hôtes finaux.

Sur ce lien ADSL, les performances ne sont pas aussi marquées que ce qui a été observé sur les expériences en banc d'essai, bien que présentant une amélioration de plus de 40% des temps de réponse. Ceci peut être expliqué par le fait que le tampon du modem est de 30 paquets (comparé aux 60 paquets simulés sur banc d'essai). De plus, le lien est bien plus asymétrique, avec un ratio entre lien montant et descendant proche de 15, tandis que le banc d'essai simulait un lien montant 6 fois plus petit que le lien descendant. Avec une telle asymétrie, dès que le lien descendant est saturé, des ACKs arrivent à une vitesse supérieure, et occupent donc le tampon montant.

Pour une charge supérieure (6 connexions par seconde), l'ajout de trafic UDP est moins significatif, car les ACKs présents dans le tampon montant ont déjà pour effet de réduire la taille effective de la file d'attente. Toutefois, son impact n'est jamais négatif pour les connexions descendantes.

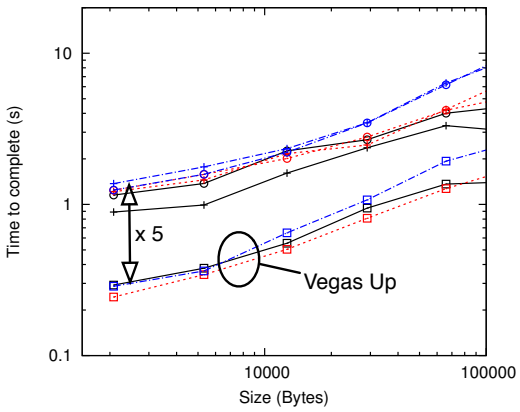


FIGURE 6-12: Temps de réponse conditionnel du *download* avec 1 connexion par seconde en présence d'un *upload* continu ; trafic TCP réaliste.

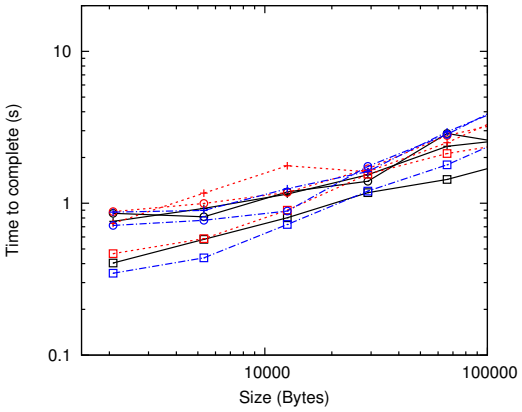


FIGURE 6-13: Temps de réponse conditionnel du *download* avec 1 connexion par seconde en présence d'un *upload* continu avec un flux UDP de 125 pkt/s ; trafic TCP réaliste.

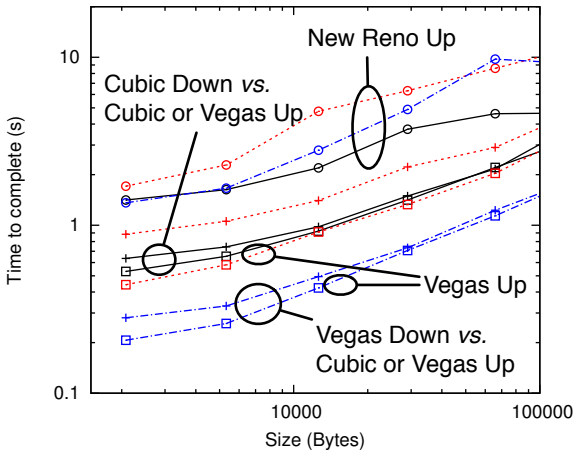


FIGURE 6-14: Temps de réponse conditionnel du *download* avec 6 connexions par seconde en présence d'un *upload* continu ; trafic TCP réaliste.

6.5 Conclusion

Ce chapitre explore une nouvelle solution pour parer aux effets négatifs du *bufferbloat* — les latences accrues par la présence de tampons surdimensionnés. Ce problème peut être résolu de manière efficace par des mécanismes existants (TCP *Vegas* sur le lien montant, CoDel ou *Fair queuing* en cœur de réseau), mais leur déploiement est au mieux complexe, parfois impossible, car demandant un accès aux routeurs, voire au code des routeurs ! Si les solutions existantes ne peuvent être utilisées, d'autres approches peuvent apporter un gain significatif en performances.

Ainsi, ajouter du trafic de faible intensité constitué de petits paquets améliore paradoxalement les latences, au prix d'une légère perte de débit. De même, le trafic descendant a une influence bénéfique

sur les performances. Ceci nous mène à un deuxième paradoxe : un lien sous forte charge peut subir moins de *bufferbloat* que le même lien sous une charge plus faible.

Dans ce chapitre, nous nous sommes concentrés sur les effets du trafic additionnel sur les données en transit entre le client et le serveur, en négligeant les potentiels problèmes qui peuvent apparaitre sur les liens subséquents. Une des améliorations de cette méthode serait l'envoi de paquets UDP à faible TTL, qui sortiraient du réseau immédiatement après le goulot d'étranglement. Dans ce cas, il serait bénéfique que les FAI limitent de débit des paquets ICMP TTL-exceeded, ce qui est souvent le cas afin de combattre les attaques se basant sur l'expiration des TTL.

Il faut également prendre en compte les liens sans fil, particulièrement dans le sens montant, car la capacité peut subir de forte variations, et des petits paquets peuvent introduire un *overhead* important (notamment pour les liens 802.11, les réseaux cellulaires pouvant concaténer les trames lors de la transmission)

Une première étape résiderait dans la réalisation d'un utilitaire qui ajuste automatiquement l'intensité de l'envoi de paquets UDP en fonction de la latence mesurée afin de s'adapter aux conditions du réseau. Cet outil permettrait ainsi de réguler le trafic dans le cas de liens à délai variable, et de supprimer le trafic UDP lorsque le lien subit une charge faible.

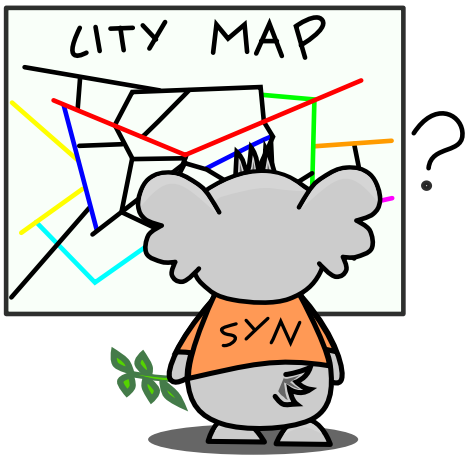
Chapitre 7

Priorisation des SYN et gestion active des files pour des temps de réponse courts

Never gonna give you up, Never gonna let you down, Never gonna run around and desert you

---

Rick Astley – Never gonna give you up



Introduction

Diminuer les temps de réponse des transferts TCP est primordial pour améliorer la réactivité de l'accès à l'information sur le réseau. Dans ce but, les trois chapitres précédents se focalisaient sur les délais de bout-en-bout et leur impact sur la transmission complète d'un flux de données. Minimiser

ces délais augmente en effet de manière notable la réactivité ressentie par l'utilisateur. Cependant, d'autres mécanismes peuvent être optimisés dans cette optique, parmi lesquels l'établissement de la connexion. Dans ce chapitre, nous nous concentrerons sur cette phase, qui peut vite devenir le facteur limitant en termes de temps de réponse, et dépasser de plusieurs ordres de grandeur la durée de transmission des données.

En effet, une nouvelle connexion n'a aucune vision du réseau, en particulier du temps d'aller-retour (RTT) entre le client et le serveur. En conséquence, en cas de perte, les premiers paquets d'une connexion, et en particulier les SYN, ne peuvent être retransmis qu'à l'expiration d'un long *timer* de retransmission (RTO), dont les valeurs par défaut sont souvent très élevées (entre 1 et 3 secondes). La réception d'un segment SYN/ACK met immédiatement àjour le RTO, ce qui permet de récupérer des pertes suivantes bien plus rapidement, même dans les cas pour lesquels le *fast retransmit* ne peut pas se déclencher. Ainsi, un client ne peut réagir à la perte d'un SYN ou du SYN/ACK correspondant qu'après une longue attente, dont la durée peut dépasser de plusieurs ordres de grandeur celle de récupération des pertes subséquentes.

Cet effet est particulièrement sensible pour les petites connexions et le trafic interactif. Seuls les transferts suffisamment longs peuvent diluer l'influence d'une perte de SYN sur toute la durée de la connexion.

En dépit de l'importance de la phase d'établissement de la connexion dans les performances, cette dernière a reçu beaucoup moins d'attention que l'échange de données en lui-même [64].

Dans un premier temps, nous nous intéresserons à l'impact des pertes de SYN ou de SYN/ACK sur les temps de réponse des connexions à travers l'analyse de plusieurs jeux de traces de trafic. Cette étude montre que les SYN et SYN/ACK sont en général perdus bien plus fréquemment que les autres segments TCP, ce qui donne une raison de plus de les considérer de manière particulière. Une approche intuitive consiste à es*timer* que les retransmissions de SYN sont deux fois plus fréquentes que les pertes de paquets, car déclenchées par la perte d'un SYN ou d'un SYN/ACK sur la voie de retour. Pourtant, les données observées montrent un taux de retransmission parfois jusqu'à 5 fois supérieur au taux de pertes moyen, ce qui exacerbe le problème pour les connexions courtes. Enfin, des mécanismes d'ordonnancement de données récents [99, 100] ont pour effet d'augmenter le taux de pertes dans le réseau indifféremment du type de paquets. Le taux de pertes de SYN croît ainsi proportionnellement à la probabilité de pertes totale, avec pour conséquence principale une perte de réactivité des connexions (à l'inverse des objectifs initiaux de ce type de politique).

En se basant sur ces observations, dans une deuxième partie de ce chapitre, un mécanisme de gestion des files nommé SPA (Syn Priority Active queue management) est proposé. Cet algorithme fonctionne grâce à deux files gérées par *CoDel* (*Controlled Delay Management*) [99] : les paquets SYN et FIN sont envoyés dans une file à haute priorité, tandis que l'autre file, à priorité plus basse,

reçoit les paquets de données. SPA est donc composé d'une paire de files avec des mécanismes d'AQM indépendants. Cette architecture est bien moins complexe que des mécanismes de *Fair Queuing*, et combine de faibles temps d'établissement de la connexion en combinaison avec une faible latence de transmission des données.

Trois contributions sont présentées dans ce chapitre. Tout d'abord, une modélisation des effets des retransmissions de SYN met en lumière l'impact des pertes de SYN sur le temps de réponse total de la connexion. Ensuite, les probabilités de pertes sont analysées à travers l'observation de traces réseau. Enfin, les performances de SPA sont étudiées et comparées aux algorithmes de gestion des files d'attente les plus connus, sur un banc d'essai.

7.1 Modélisation des temps de réponse TCP en fonction des pertes de SYN

Afin d'évaluer l'impact des pertes de SYN sur les transferts TCP de petite taille, le modèle bien connu de Mathis *et al.* [101] et Padhye *et al.* [102] est utilisé avec quelques modifications. Tout d'abord, l'évolution de la fenêtre de congestion est considérée discrète et non linéaire comme dans le modèle de Mathis. Ensuite, l'analyse est étendue à l'intégralité de la connexion : *slow start*, récupération de pertes et transmission des SYN et FIN ont été ajoutés au modèle.

La durée totale d'une connexion peut ainsi être exprimée de la manière suivante :

$$t_{\text{tot}} = t_{\text{syn}} + t_{slow\_start} + t_{congestion\ avoidance} + t_{recovery} + t_{\text{fin}}, \tag{7.1}$$

Tandis que l'impact de la phase d'établissement de la connexion est représenté par :

$$t_{\text{syn}}/t_{\text{tot}} \tag{7.2}$$

7.1.1 *Congestion avoidance*

Considérons tout d'abord la phase de *congestion avoidance* d'une connexion TCP avec un RTT constant et sujette à une probabilité de perte p_l.

Après la i^e perte, si $W_{\max}(i-1)$ est la fenêtre de congestion maximale (en segments) obtenue avant la perte précédente, alors,

$$W_{\max}(i) = \frac{W_{\max}(i-1)}{2} + n_i,$$

où n_i est le nombre de RTTs entre deux pertes. Donc, la valeur moyenne des fenêtres maximales

est :

$$\bar{W}_{\max} = 2 \times \bar{n}, \tag{7.3}$$

où $\bar{n} * \text{RTT}$ est le temps moyen entre les pertes.

Comme la fenêtre de congestion grandit d'un segment par RTT, il est possible de montrer par récurrence que le nombre de segments envoyés à la fin du $n^{\text{ème}}$ RTT depuis la dernière perte est :

$$\frac{\bar{W}_{\max}}{2} * n + \sum_{k=0}^{n-1} k = \frac{\bar{W}_{\max}}{2} * n + \frac{(n-1) * n}{2}. \tag{7.4}$$

Si la probabilité de perte est p_l et qu'une seule perte a lieu par épisode de congestion, alors le nombre moyen de paquets envoyés entre deux pertes est $1/p_l$, et la combinaison des équations 7.3 et 7.4 donne :

$$\frac{3 * \bar{W}_{\max}^2}{8} - \frac{\bar{W}_{\max}}{4} = \frac{1}{p_l} \tag{7.5}$$

donc,

$$\bar{W}_{\max} = 1/3 * (\sqrt{\frac{24}{p_l} + 1} + 1). \tag{7.6}$$

Comme la fenêtre moyenne estimée est égale à

$$\bar{W}_{\text{avg}} = 3/4 * \bar{W}_{\max}$$

, le débit estimé (en paquets/s) est égal à

$$\bar{X} = \frac{3}{4} * \frac{\bar{W}_{\max}}{RTT}.$$

et le temps moyen passé en *congestion avoidance* pour transférer n_{data} est

$$t_{congestion\ avoidance} = \frac{n_{data}}{\bar{X}}$$

Ce modèle permet de donner un bon aperçu de l'influence d'une perte de SYN sur une connexion. Ainsi, dans le cas d'une connexion présentant un RTT de 200ms, pour une probabilité de pertes de 2%, 1336 paquets doivent être transmis afin que le temps passé à récupérer de la perte de SYN soit inférieure à 10% de la durée totale de la connexion. Toutefois, cette estimation représente une limite basse, qui ne prend pas en compte le temps passé en *slow start* et en récupération des pertes, caractérisés par des débits réduits.

7.1.2 *Slow start*

On modélise la phase du *slow start* comme un état dans lequel la connexion commence avec une fenêtre de congestion d'un segment pour atteindre $\bar{W}_{\max}$, la fenêtre de congestion maximale calculée précédemment, ce qui marque la sortie de cette phase et le passage en *congestion avoidance*. Ce modèle, bien qu'imprécis, permet de prendre en compte l'accélération progressive de la connexion avant de remplir le lien.

Le nombre de RTT n_{ss} écoulés durant cette phase peut être estimé comme suit : $2^{n_{\text{ss}}-1} = \bar{W}_{\max}$. Ainsi,

$$n_{\text{ss}} = \frac{log(\bar{W}_{\max})}{log(2)} + 1.$$

Durant cette phase, la connexion envoie donc $2^{n_{\text{ss}}} - 1$ segments, pendant une durée $t_{\text{syn}} = RTT * n_{ss}$

7.1.3 Récupération des pertes

En considérant qu'une seule perte arrive par épisode de congestion, que la probabilité de perte totale est p_l et le temps de récupération R_t, une connexion de taille S MSS passe en moyenne $S * R * p_l$ secondes en récupération. Une approximation optimiste de R_t est d'un RTT (dans la pratique, on observe plutôt 1.5 à 2 RTT).

7.1.4 Pertes de SYN

Si la probabilité de perte de SYN est p_{Sl}, le temps passé dans la phase d'établissement de connexion est :

$$\begin{aligned} t_{\text{syn}} &= RTT + RTO_0 * \sum_{k=1}^{\infty} (p_{\text{Sl}})^k \\ &= RTT + \left(\frac{1}{1 - p_{\text{Sl}}} - 1 \right) * RTO_0 \end{aligned} \tag{7.7}$$

Où RTO_0 est le *timer* de retransmission initial (entre 1 et 3 secondes selon l'implantation). Pour obtenir des valeurs numériques des temps de réponse moyens des connexions, p_{Sl} peut être approché par $\approx 2\,p_l$ ou, pour plus de précision, par une valeur obtenue de manière expérimentale.

7.1.5 Temps de réponse total

Donc, pour une connexion de taille S, le temps de réponse total est :

$$t_{tot} = t_{\text{syn}} + t_{slow\_start} + t_{congestion\ avoidance} + t_{recovery} + t_{\text{fin}}, \tag{7.8}$$

où

$$t_{slow\_start} = \left(\frac{log(\bar{W}_{\max})}{log(2)} + 1 \right) * RTT, \tag{7.9}$$

$$t_{congestion\_avoidance} = \frac{S - (2^{n_{ss}} - 1)}{X_{\text{avg}}}, \tag{7.10}$$

$$t_{recovery} = S * R_t * p_l, \tag{7.11}$$

$$t_{\text{fin}} = RTT. \tag{7.12}$$

7.1.6 Application à des données réelles

Ces équations ont été appliquées aux mesures faites pour la trace CAIDA #1 (*cf* Table 7-2) présentée dans la section suivante. L'analyse de cette trace nous donne les valeurs suivantes pour les probabilités de perte et temps de retransmission :$p_l = 0.02$, $p_{Sl} = 0.12$, $R_t = 1\,s$. Étant donné que les traces ne fournissent pas d'information sur les RTT, ce dernier a été fixé à une valeur arbitraire de $200\,ms$. Si l'on considère que la page web est d'une taille moyenne de 55 kB selon httparchive [103], le tranfert HTTP moyen est de $S = 36$ segments.

Avec ces paramètres, $\bar{W}_{\text{avg}} = 10$, et $\bar{X} = 50$pkt/s. Ainsi,

$$\begin{aligned} t_{slow\_start} &= 0.86\,s, \\ t_{congestion\_avoidance} &= 0.32\,s, \\ t_{recovery} &= 0.50\,s, \\ t_{\text{fin}} &= 0.2\,s, \\ t_{\text{syn}} &= 0.2 + 0.4 = 0.6\,s, \end{aligned} \tag{7.13}$$

Ce qui correspond à un temps total moyen de $2.48\,s$, pour lesquels les retransmissions de SYN comptent pour 24%. Dans le cas d'une perte de SYN, la page mettrait $5.12\,s$ à se charger comparé à $2.12\,s$ sans pertes. Ce modèle confirme donc qu'une simple perte de SYN est suffisante pour doubler le temps de chargement d'une page web !

La Figure 7-1 représente les temps de réponse estimés pour les connexions avec est sans perte de SYN pour des RTO de 1 et 3 secondes, pour une connexion entre 25 et 10000 segments dans les conditions de la Trace #1 (Figure 7-2). Les ratios des temps de réponse des connexions avec et sans perte de SYN sont également représentés.

Les pertes de SYN représentent donc un problème majeur pour les connexions courtes, qui obtiennent des temps de réponse supérieurs à 200% du temps de réponse des connexions sans perte de SYN. L'impact devient négligeable ($i.e < 10\%$ du temps de réponse total) seulement pour des connexions de plus de 1000 segments pour un RTO d'1 s, et 2000 segments pour un RTO de 3 s. Les connexions

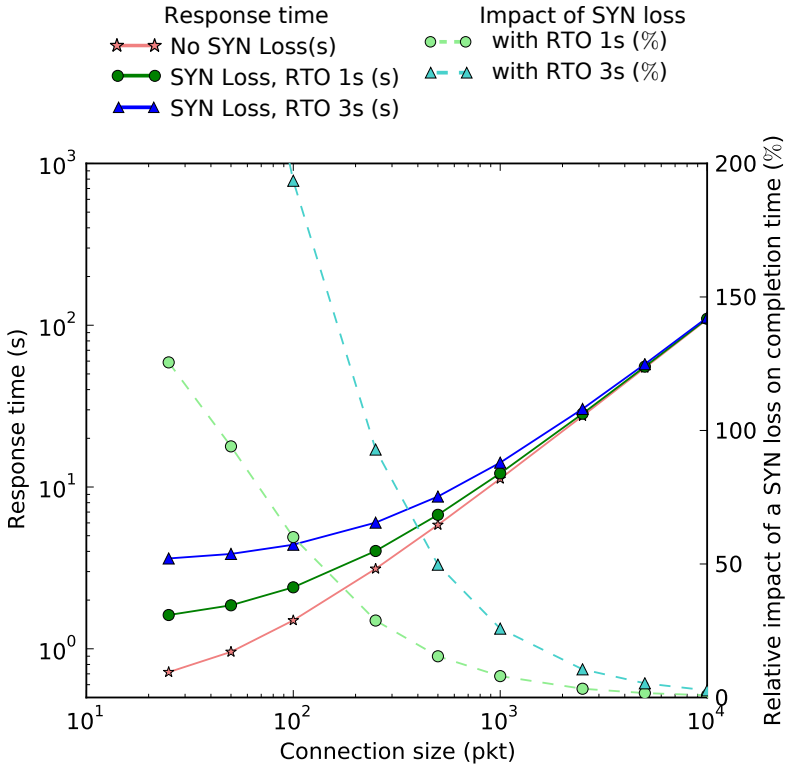


FIGURE 7-1: Temps de réponse des connexions avec et sans perte de SYN

courtes, qui sont déjà pénalisées par rapport aux connexions longues, souffrent énormément de ce phénomène, qui rend vaine toute tentative d'interactivité dans un environnement où les probabilités de pertes de SYN dépassent quelques pourcents.

7.2 Analyse de traces réelles.

TABLE 7.1: Traces CAIDA, dates et heures de capture

Trace #1	20/03/2014 13 :03 :00
Trace #2	18/09/2014 13 :07 :00

Afin d'évaluer l'impact des pertes de SYN ou SYN/ACK, un jeu de traces réseau accessible au public a été analysé : cinq ensembles de traces du jeu de données CAIDA [1]. Ces traces ont été enregistrées deux jours différents dans un *datacenter* d'Equinix situé à Chicago, connecté au lien *backbone* Ethernet 10 Gb d'un opérateur Tier 1.

Les traces enregistrées le 20 mars 2014 ont une utilisation du lien plus faible dans la direction A que celles enregistrées le 18 septembre 2014. Les résultats étant similaires pour des traces prises le même

1. The CAIDA UCSD Anonymized Internet Traces 2014
20/03/2014 12 :59 :11, 13 :03 :00, 13 :06 :00 and 18/09/2014 13 :07 :00, 13 :19 :00
`http://www.caida.org/data/passive/passive_2014_dataset.xml`

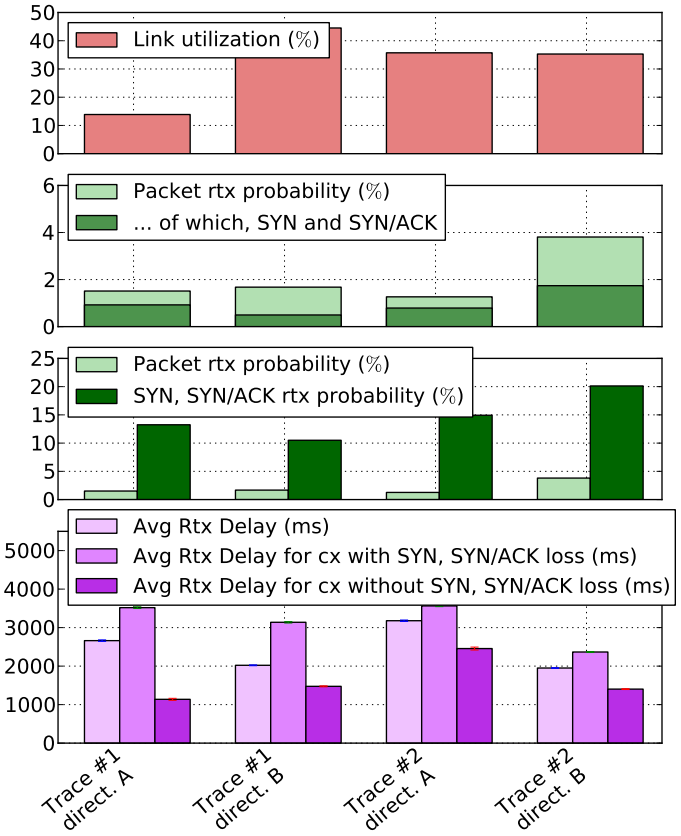


FIGURE 7-2: Analyse des traces CAIDA dans les deux directions, ordonnées par ordre croissant d'utilisation du lien dans la direction A. Les pertes de SYN représentent une large portion des pertes, et les SYN ont une probabilité de perte supérieure à celle des segments de données, avec un impact considérable sur les temps de retransmission.

jour, une seule paire de traces a été gardée par jour. Pour des questions de lisibilité, ces dernières seront référencées en suivant les notations de la Table 7.1.

Toutes les mesures à l'exception de l'utilisation du lien concernent des connexions avec au minimum un SYN, ce qui représente 180,000 des 650,000 connexions d'une trace. Les connexions qui présentent des temps de retransmissions supérieurs à 45 s ne sont pas considérées afin de retirer les données incohérentes de l'analyse (45 s correspond au délai total après 3 pertes de SYN au départ d'une connexion à cause de l'*exponential backoff*) [2].

En raison du nombre de mesures, l'intervalle de confiance à 95% est très petit et n'a pas pu être représenté pour les probabilités de pertes. Il est à peine visible pour les temps de retransmission.

La Figure 7-2 montre deux phénomènes marquants :

1. Les retransmissions de SYN et SYN/ACK représentent une grande partie de toutes les retransmissions (phénomène visible dans la seconde figure) ;

2. Comme indiqué dans le code source de FreeBSD *"the odds are that the user has given up attempting to connect by then."* [104].

2. La probabilité de retransmission de SYN ou de SYN/ACK est très élevée — entre 10 et 20% (3^{e} figure), et les connexions qui subissent au moins une perte de SYN ou SYN/ACK souffrent d'un temps de retransmission doublé par rapport aux autres connexions (4^{e} figure).

Notons que le RTO initial est de 3 secondes par défaut pour Windows, FreeBSD, ainsi que pour les versions du noyau Linux antérieures à 2011.

Le taux de retransmission plus élevé des SYN comparé aux autres segments est un effet sans surprise étant donné qu'une perte de SYN/ACK sur la voie de retour cause toujours un *timeout*, tandis que les acquittements cumulatifs rendent le reste de la connexion relativement insensible aux pertes dans la direction opposée [105]. Donc, comme les SYN et SYN/ACK ont la même probabilité de perte, le taux de retransmission double. Comme les files en cœur de réseau sont gérées selon une politique FIFO, dimensionnées en paquets, les petits segments tels que les SYN ont les mêmes chances d'être perdus que les paquets de données. Une file FIFO basée sur le nombre d'octets présents dans la file pourrait ainsi favoriser les petits paquets, qui peuvent, en général, être toujours acceptés dans la file d'attente alors qu'un grand paquet aurait été perdu.

Néanmoins, cela n'explique qu'une petite fraction de la différence observée ici. Une autre hypothèse qui peut expliquer la divergence entre les pertes de SYN et les pertes de données pourrait provenir des algorithmes de contrôle de congestion de TCP. Ceux-ci sont en effet conçus afin que les connexions perdent un nombre de paquets limité lors d'un épisode de congestion (1 paquet dans le cas de la phase de *congestion avoidance*). À la suite d'une perte, TCP divise la fenêtre de congestion par 2 pour réduire les débits de transmission et ainsi retarder la prochaine occurrence d'une congestion. Ainsi, les paquets transmis à la suite d'une perte ont une probabilité bien plus faible d'être perdus à leur tour. En contraste, les SYN arrivent à des instants aléatoires, et leur retransmission potentielle est en général tellement distante dans le temps que les deux événements peuvent être considérés comme décorrélés, ce qui peut mener à un taux de pertes supérieur aux paquets de données. De plus amples investigations sont requises pour expliquer cette différence.

Afin de confirmer ces mesures, des traces collectées par le groupe de travail MAWI (projet WIDE) ont été analysées. Ces traces ont été enregistrées sur le lien de transit entre WIDE et le fournisseur d'accès. De même, les connexions qui présentent des temps de retransmission supérieurs à 45 s ont été filtrés (cf. Table 7.2).

Une des deux directions présente moins de pertes, tandis que les connexions subissent beaucoup plus de *timeout*s (temps de retransmission plus élevés). Le même phénomène peut être observé dans les traces CAIDA enregistrées le 18 septembre 2014 dans la direction A : bien que les pertes soient moins fréquentes que dans la direction B, la plupart d'entre elles se termine par un *timeout*. Le *bufferbloat* dans un tampon en amont peut être à l'origine de ce problème, causant des délais supérieurs à la valeur du RTO.

TABLE 7.2: Analyse des traces MAWI

	Dir A	**Dir B**
Probabilité de retransmissions de paquets.	1.65%	2.73%
... parmi lesquelles SYN et SYN/ACK représentent	29.8%	79.5%
Probabilité de retransmission de SYN et SYN/ACK.	13.1%	29.9%
Temps de retransmission moyen (ms)	5276.96	2114.93
Temps de retransmission moyen pour des connexions avec pertes de SYN ou SYN/ACK (ms)	6324.93	2390.63
Temps de retransmission moyen pour des connexions sans pertes de SYN ou SYN/ACK (ms)	4707.11	1011.93

Les délais observés corroborent les résultats des traces CAIDA : les pertes de SYN et de SYN/ACK représentent la moitié des pertes. Leur retransmission double les temps de retransmission subis par les connexions.

De tels résultats soulèvent une question importante : ces traces étant relevées sur des liens d'interconnexion, il est très probable que des attaques par SYN *flood* soient présentes. Ces attaques peuvent-elles biaiser les résultats de notre analyse ? Afin d'analyser l'impact de ce type de pratiques, les potentiels SYN *floods* ont été filtrés, et les connexions impliquant des adresses IP qui montrent des signes de SYN *flood* ont été retirées.

Pour identifier ces adresses, les connexions sont séparées en deux groupes : celles qui transmettent des segments de données et celles ne présentant que des messages de contrôle (SYN, FIN, ACK, RST). Les hôtes impliqués dans 1000 fois plus de connexions sans paquets de données que de connexions avec données (avec un minimum de 1000 connexions sans données) ont été considérés comme participant au SYN flood et retirés de la trace. Notons que ceci n'est possible que pour les traces bidirectionnelles de MAWI.

Sur les 587,262 adresses IP présentes dans la trace, 69 ont été retirées. Ce filtrage correspond à un retrait de 30% des connexions, principalement dans la direction B. Les résultats ainsi corrigés sont présentés Table 7.3.

TABLE 7.3: Traces MAWI, SYN flood filtré

	Dir A	**Dir B**
Probabilité de retransmission de paquets	1.63%	2.72%
... parmi lesquelles SYN et SYN/ACK représentent	28.8%	78.9%
Probabilité de retransmission de SYN et SYN/ACK.	12.8%	40.6%
Temps de retransmission moyen (ms)	5312.9	2114.93
Temps de retransmission moyen pour les connexions avec pertes de SYN ou SYN/ACK (ms)	6469.4	2390.63
Temps de retransmission moyen pour les connexions sans pertes de SYN ou SYN/ACK (ms)	4710.5	1011.93

Peu de connexions ont été retirées dans la direction A, et les résultats sont donc similaires à l'analyse précédente. En revanche, filtrer une grande part des connexions dans la direction inverse augmente de manière spectaculaire les probabilités de retransmission de SYN, jusqu'à 40%, alors que les délais

et probabilités de retransmissions de paquets de données sont peu affectés. En effet, une attaque par SYN flood est caractérisée par l'envoi de nombreux SYN uniques, et l'absence de transmission ultérieure de données, car l'attaquant ne cherche pas à établir une connexion.

Les résultats obtenus, qu'ils proviennent des traces MAWI ou CAIDA, coïncident avec les conclusions de Damjanovic sur le projet LBNL/ICSI Enterprise Tracing [65]. Les mesures présentées dans cet article montre un taux total de 10% de retransmissions de SYN pour toutes les connexions dans un réseau local. L'analyse effectuée ci-dessus témoigne de pertes plus importantes, ce qui peut être expliqué par le fait que les traces MAWI ou CAIDA proviennent de liens de transit, où des connexions 3/4G, WLAN, ou encore satellite peuvent être présentes, avec pour conséquences des délais et débits variables et des pertes aléatoires. Une autre explication peut provenir de la présence de congestions sur le chemin, plus probable sur un lien de transit que dans un réseau local d'entreprise. Enfin, certaines applications peuvent tenter d'émettre des connexions vers des hôtes déconnectés, ce qui entraîne une surcharge de SYN importante sur le réseau.

7.3 SPA – SYN Priority AQM

Une solution récente aux problèmes de *bufferbloat* est CoDel [99], un algorithme de gestion des files basé sur le délai conçu pour être déployé avec un minimum de configuration. PIE [100] est un autre exemple de mécanisme développé afin de garantir de faibles latences dans le réseau. Toutefois, ces deux algorithmes (et en règle générale toutes les approches qui visent à minimiser le délai) se basent sur le postulat que des pertes fréquentes sont bénéfiques au réseau, car elles permettent de minimiser les délais. Ces mécanismes présentent donc des taux de pertes bien supérieurs aux autres types de files telles que FIFO ou RED. Toutefois, les pertes sont indifférenciées, bien que la perte de certains paquets ait été clairement identifiée comme résultant en un *timeout*. Ceci est tout particulièrement le cas pour les pertes en tête de connexion, lorsque le RTO est toujours à ses valeurs par défaut. En fin de connexion, les pertes peuvent aussi avoir pour conséquence des longs *timeout*s si les délais fluctuent au cours de la connexion.

FQ (*Fair Queuing*) CoDel [106] apporte une réponse à ce problème en donnant une priorité supérieure aux premiers paquets d'un flux, mais le gain en performance ne peut compenser la complexité apportée : par exemple, l'implantation par défaut sous Linux utilise 1024 files séparées, pour une empreinte mémoire bien supérieure.

Cette section présente une solution concurrente à ces algorithmes : SYN Priority AQM (SPA). L'approche suivie par SPA a pour but d'apporter un compromis entre CoDel et FQ CoDel : seules deux files sont gérées par CoDel — une file d'attente à haute priorité pour les paquets SYN et FIN, et une autre à basse priorité pour les paquets de données. En effet, quatre types de pertes ne peuvent

être résolues que par un *timeout* : SYN, FIN et pertes en fin de connexion ou en fin de transmission dans le cas d'un flux intermittent. Les pertes de paquets de données en fin de transmission sont plus difficilement détectables au niveau du routeur, et des solutions de bout-en-bout efficaces existent (Tail Loss Probe, Early Retransmit). En revanche, les SYN et les FIN peuvent être aisément isolés dans une file séparée, en appliquant un simple masque sur les *flags* des paquets. De cette manière, il est possible d'augmenter les performances globales en évitant la plupart des *timeouts*, tout en gardant une complexité bien inférieure à celle de FQ CoDel.

Le pseudo-code ci-dessous définit l'algorithme de manière plus formelle :

```
synfinqueue = CoDel()
packetqueue = CoDel()

def enqueue(p):
  if p.flag.SYN==set or p.flag.FIN==set:
    synfinqueue.enqueue(p)
  else:
    packetqueue.enqueue(p)

def dequeue(p):
  if synfinqueue.is_empty():
    packetqueue.dequeue(p)
  else:
    synfinqueue.dequeue(p)
```

Listing 7.1: Algorithme de SPA (SYN Priority AQM)

Notons que SPA peut être facilement mis en place en se basant sur une implantation existante de CoDel en coordination avec des outils système usuels. Ainsi, pour réaliser les expériences décrites ci-dessous, SPA a été conçu comme une file à priorité servant deux sous-files gérées par CoDel, au travers de l'utilisation de *tc* [107] et *iptables*.

Comme mentionné ci-dessus, une autre possibilité pour éviter les pertes de SYN et SYN/ACK est de dimensionner les tampons en octets plutôt qu'en paquets. Bien qu'efficace et relativement simple à implanter, cette solution présente un inconvénient majeur : il est nécessaire de connaître le produit délai-bande passante du lien pour dimensionner la file d'attente, ce qui peut s'avérer périlleux dans les liens sans fil (GSM, WiFi) pour lesquels ce dernier n'est pas fixe et présente une variance élevée. Si cette étape n'est pas effectuée, la file risque d'être surdimensionnée (et montrer des signes de *bufferbloat*) et de fortes latences seront observées dans le réseau. Au contraire, si la file est sous-dimensionnée, l'utilisation du lien ne sera pas optimale. De par sa conception, SPA ne souffre pas de ces limitations.

7.4 Expériences en conditions réelles et comparaison des différentes politiques de files d'attente

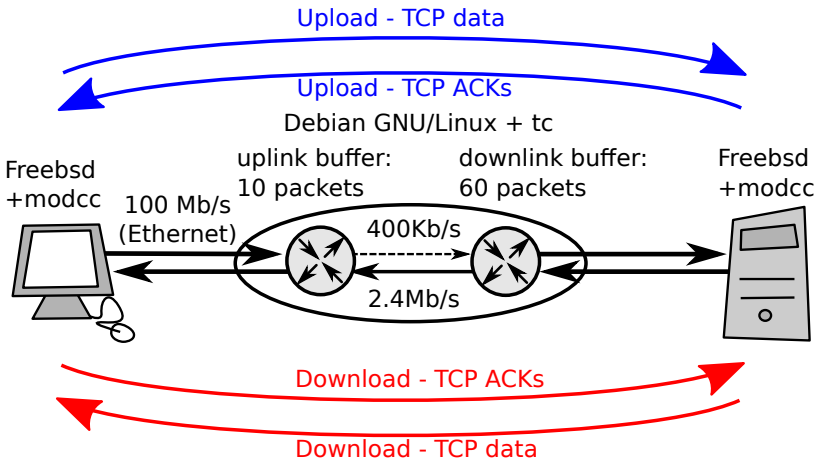


FIGURE 7-3: Testbed network for experiments.

Les expériences ont été lancées sur un réseau réel composé de trois machines (*cf.* Figure 7-3). Ce banc d'essai est similaire à celui décrit dans le chapitre précédent, à l'exception du routeur, transformé de la façon suivante : afin de pouvoir utiliser des utilitaires de gestion des files plus poussés, à l'instar de *netem* et *tc*, le routeur utilise désormais Debian GNU/Linux (wheezy), avec un noyau légèrement modifié. Au-dessus de ce système d'exploitation, *netem* et *tc* sont utilisés pour contrôler les files, tandis qu'*iptables* trie et marque le trafic avant de l'envoyer dans la file d'attente appropriée.

Comme avec FreeBSD, la fréquence de commutation de contexte est modifiée afin d'émuler un goulot d'étranglement de manière réaliste et d'éviter de créer des rafales de paquets supplémentaires dans le réseau. Dans le cas du noyau Linux, la valeur maximale autorisée est 1 KHz, ce qui permet d'émuler des liens jusqu'à 10 Mb/s de manière fluide.

Les connexions sont générées à l'aide de la suite `ipmt`, selon le modèle poissonien/zipf décrit lui-aussi dans les chapitres précédents. Le trafic est observé à l'aide de `tcpdump` et analysé grâce à `tcptrace`.

Les expériences sont lancées avec et sans trafic sur la voie de retour, selon les conditions décrites Table 7.4. Ces dernières correspondent toutes à une utilisation du lien aux alentours de 90%. En faisant varier les paramètres λ et μ des distributions, ainsi que la taille maximale des connexions, des conditions de trafic très différentes émergent.

Trafic Type #1 correspond à l'hypothèse réaliste de tailles de connexions suivant une distribution de Zipf à queue lourde. Dans ce cas, $1/\lambda = 0.12\,s$ et $\mu = 1.7$, ce qui permet de générer un trafic composé de connexions de 21 segments en moyenne, avec une médiane à 2 paquets et 80% des connexions de moins de 6 segments. Le lien émulé présente un débit descendant de 2.4 Mb/s pour

TABLE 7.4: Paramètres des expériences

Lien montant		
Capacité C_u	0.4 Mb/s	2 Mb/s
Délai D_u	52 ms	
Gestion des files Q_u	*Packet* FIFO	
Lien descendant		
Capacité C_d	2.4 Mb/s	10 Mb/s
Délai D_d	42 ms	
Gestion des files Q_d	*Packet* FIFO - *Byte* FIFO RED - ARED - REDFavor CoDel - FQ CoDel SFQ PIE SYN Prio - SYN/FIN Prio - SPA	
Paramètres du générateur de trafic		
trafic Type	**#1**	**#2**	**#3**
$1/\lambda$ (s)	0.12	0.017	0.032
μ	1.7	2	1.7
Nombre de connexions	8326	58803	31306
Taille moyenne	21 seg.	3 seg.	26 seg.

400 Kb/s montant.

Trafic Type #2 représente des conditions extrêmes afin de pousser les algorithmes de gestion de files d'attente dans leurs retranchements. Le lien utilisé dans le cas du Trafic Type #1 est soumis à la charge suivante : une connexion est générée toutes les $1/\lambda = 0.017\,s$, avec une taille moyenne de 3 segments ($\mu = 2$). La médiane est à 1 paquet, tandis que la taille maximale atteint 100 segments (80% des connexions génèrent 4 segments ou moins).

Trafic Type #3. Ce type de trafic permet d'analyser le comportement des différentes files pour un lien d'une capacité de 10 Mb/s, sous une charge de 95% composée de connexions arrivant toutes les 0.032 seconde, de taille moyenne 26 paquets et de médiane 2 segments (80% de moins de 7 segments).

Les politiques de gestion des files suivantes sont testées sur le lien descendant (Q_d) : FIFO (en paquets – *Packet* FIFO ou en octets – *Byte* FIFO), divers AQM (RED, ARED [52]), mécanismes basés sur le délai (CoDel et PIE), et *Fair Queuing* (FQ CoDel et SFQ – Stochastic Fair Queuing [108]). Ces algorithmes très répandus sont comparés à des files à priorité qui isolent les SYN ou SYN/FIN (SYN Prio, SYN/FIN Prio, RedFavor [64] et SPA). La file du lien montant reste inchangée pour toutes les expériences (*Packet* FIFO).

En ce qui concerne CoDel, FQ CoDel et SPA, le palier par défaut de 5 ms est utilisé. Toutefois, certains cas poussent à changer ce paramètre pour atteindre des performances acceptables, c'est pourquoi les expériences ont aussi été réalisées avec un palier de 20 ms.

Cette analyse se base sur les solutions en cœur de réseau, et les solutions de bout-en-bout comme le réajustement du *timer* de retransmission à une valeur plus agressive [65, 66, 67] n'ont pas été inclues. En effet, si l'effet de ces techniques sur les performances générales est similaire, seul le temps de retransmission est amélioré, tandis que la probabilité de pertes de SYN reste la même.

7.4.1 Trafic Type #1

Trafic unidirectionnel

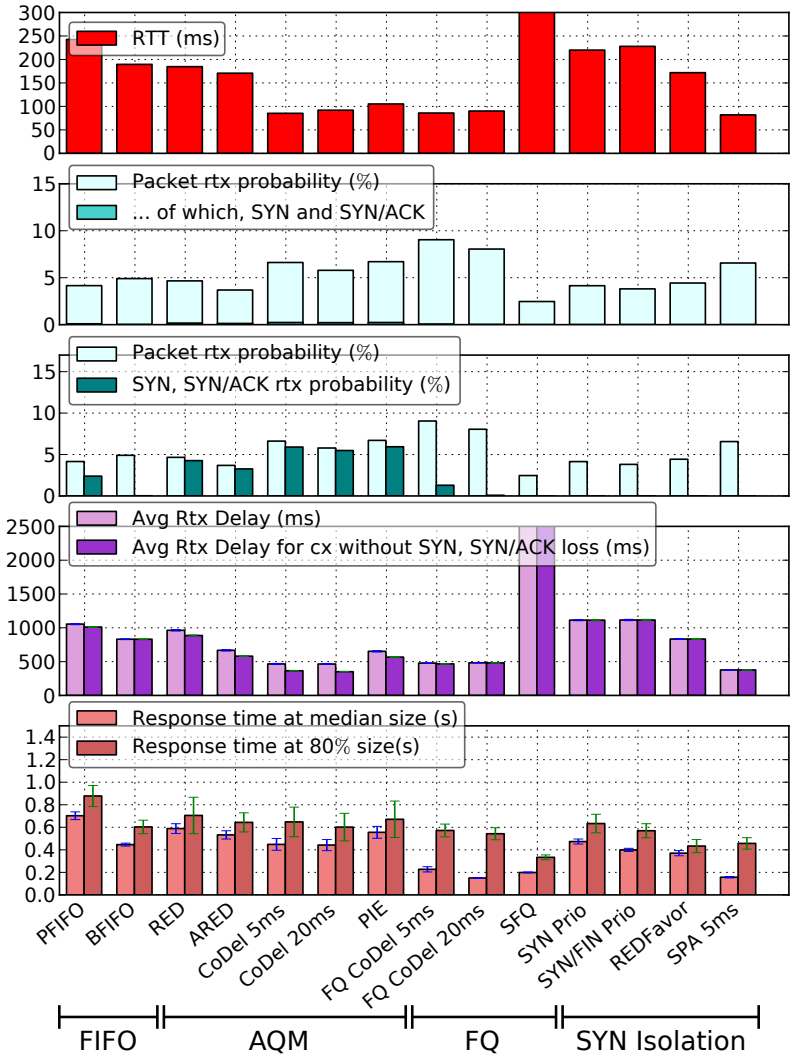


FIGURE 7-4: RTT moyen, pourcentage de perte, temps de retransmission et temps de réponse pour divers algorithmes de file d'attente sous le trafic Type #1. FQ CoDel, SFQ, RedFavor et SPA atteignent les temps de réponse les plus bas.

Figure 7-4 présente les résultats pour le trafic Type #1. De la même manière que pour la Figure 7-2, le nombre élevé de connexions sur lesquelles les mesures ont été effectuées (8326) donne un intervalle de confiance à 95% trop faible pour être représenté, à l'exception des temps de réponse.

Cette figure présente deux valeurs pour les temps de réponse : la première concerne les connexions de taille médiane (2 connexions), tandis que la deuxième correspond aux 80% des connexions de 6 MSS ou moins. Plusieurs phénomènes sont observables. Tout d'abord, n'importe quel mécanisme donne de meilleurs temps de réponse que *Packet* FIFO. De plus, certains algorithmes sortent clairement

du lot : FQ CoDel, SFQ, RedFavor et SPA. Pour les connexions les plus courtes, FQ CoDel 20 ms donne les temps de réponse les plus courts, suivi de près par SPA. SFQ atteint les meilleurs résultats pour des connexions plus longues, tandis que RedFavor et SPA maintiennent des temps de réponse bas.

En ce qui concerne les RTT, la différence entre les solutions basées sur le délai et les mécanismes standards est remarquable : CoDel et PIE divisent le RTT par deux par rapport à FIFO ou RED, tandis que FQ CoDel et SPA atteignent les RTT les plus faibles, aux alentours de 80 ms.

Le troisième histogramme compare les taux de pertes : PIE, RED et CoDel montrent des taux de retransmission de SYN entre 4 et 6% qui tendent vers plus d'équité que les autres en ce qui concerne les pertes de paquets. Un quart des temps de retransmissions de CoDel est dû aux pertes de SYN. Cet effet confirme qu'une petite fraction des connexions subit un délai de 3 secondes avant même d'avoir pu envoyer un seul paquet.

Une file FIFO configurée en Octets présente des temps de retransmission inférieurs (20%) à la version en paquets, pour des performances similaires à RED, sans pertes de SYN. Les RTT sont également inférieurs à ceux d'une file FIFO en paquets. En ce qui concerne les temps de réponse, *Byte* FIFO présente des performances supérieures à RED, ARED, CoDel ou PIE !

SFQ conduit à des délais de retransmission inhabituellement élevés. En effet, alors même que peu de pertes de SYN ont lieu, la plupart des pertes sont résolues par un *timeout* dans le meilleur des cas (et peuvent atteindre plusieurs *timeout*s pour la résolution d'une seule perte). De plus, les RTT sont beaucoup plus élevés (de l'ordre de la seconde) que pour les autres mécanismes : ils sont tronqués sur la figure afin de pouvoir observer les différences entre les autres solutions. Ce comportement peut être considéré comme un mélange entre des phénomènes de *starvation* (famine) et de *bufferbloat*. En effet, SFQ repose sur un certain nombre de files d'attente (1024 dans l'implantation par défaut) dans lesquelles chaque connexion est servie de manière périodique. Dans le cas d'une charge composée d'un grand nombre de connexions, la problématique de *bufferbloat* réapparait, démultiplié par le nombre de files. Une connexion n'est servie qu'une fois toutes les $\frac{n_{connexions} * \bar{S_{pkt}}}{C}$, où $\bar{S_{pkt}}$ est la taille moyenne des paquets et C la capacité du lien. De ce fait, pour une taille moyenne de 1000 octets sur un lien à 10Mb/s occupé par 100 connexions en parallèle, chaque connexion n'est servie que toutes les 80ms, avec des RTT atteignant la seconde si l'on considère des files d'attente de 12 paquets. Ainsi, les flots les plus longs sont condamnés à attendre pendant une durée pouvant dépasser celle du *timer* de retransmission. De plus, lorsqu'une perte a lieu, la file ne libère pas assez de paquets pour pouvoir générer un nombre suffisant d'ACKs dupliqués et déclencher l'algorithme de *fast retransmit*. Les pertes sont donc systématiquement résolues par un *timeout*. Par contraste, ce phénomène n'empêche pas SFQ d'atteindre en moyenne les meilleurs temps de réponse pour les connexions courtes.

Isoler les SYN et/ou FIN dans une file séparée (SYN et SYN/FIN Prio, Redfavor) ne procure pas une amélioration significative des RTT ou temps de retransmission en comparaison d'une simple file FIFO en paquets. Néanmoins, la probabilité de retransmission de SYN est réduite, ce qui améliore les temps de réponse, notamment pour les connexions courtes.

Enfin, SPA présente les temps de retransmission les plus courts, tout en gardant des RTT et temps de réponse parmi les plus faibles. FQ CoDel atteint des résultats similaires, bien que légèrement en deçà de SPA lorsque ses paramètres par défaut sont utilisés.

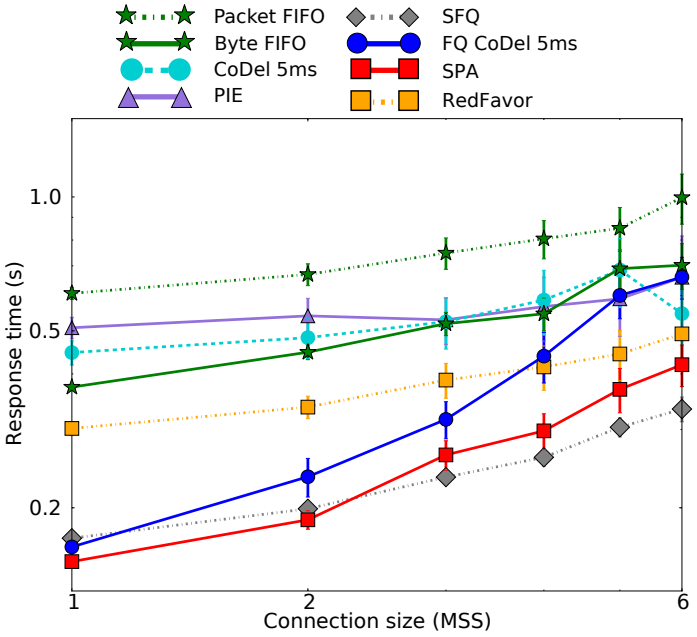


FIGURE 7-5: Temps de réponse des connexions pour divers algorithmes de gestion des files d'attente sous le trafic Type #1. SPA et les algorithmes de Fair Queuing présentent des temps de réponse clairement inférieurs aux autres mécanismes.

La Figure 7-5 présente les temps de réponse conditionnels des connexions pour plusieurs types de file d'attente. Seul les résultats significatifs sont tracés. Ainsi, RED ou ARED ne sont pas retracés car ils ne présentent pas d'amélioration significative. De même, SYN et SYN/FIN Prio ont un intérêt très limité par rapport à une simple file FIFO en octets. Comme CoDel et FQ CoDel ont été conçus pour ne nécessiter aucune configuration, le choix a été fait de ne pas montrer CoDel 20 ms et FQ CoDel 20 ms, qui constituent une modification de l'algorithme original.

La séparation en deux groupes est manifeste : PIE et CoDel ont des performances similaires, tandis que *Byte* FIFO améliore considérablement les performances des connexions en comparaison avec *Packet* FIFO (les flux les plus courts se terminent en un temps 40% plus court). Par ailleurs, SPA, SFQ et FQ CoDel présentent les temps de réponse les plus bas, avec une amélioration entre 100 et 300% pour les connexions courtes par rapport à *Packet* FIFO.

Dans le cas de solutions fondées sur le délai comme CoDel ou Pie, cette figure confirme que la philosophie de "pertes bénéfiques" est à double tranchant lorsque les segments de contrôle comme les SYN ne sont pas considérés de manière spécifique : ces mécanismes entraînent en effet de forts taux de retransmission, ce qui permet de réduire la congestion et les délais, au détriment de l'établissement de la connexion. En conséquence, ces algorithmes subissent des temps de retransmission et des délais supérieurs à ce qu'ils pourraient atteindre, bien que les performances soient meilleures qu'avec *Packet* FIFO.

REDFavor obtient des résultats bien supérieurs aux autres mécanismes AQM : avec une simple modification, les temps de réponse sont divisés par deux par rapport à *Packet* FIFO, ce qui est une amélioration notable en comparaison des autres algorithmes.

Enfin, SPA présente des temps de réponse encore plus bas que FQ CoDel et SFQ pour les connexions courtes. Pour des flux de plus de 3 MSS, la différence entre SPA et FQ CoDel s'estompe. SFQ continue d'obtenir des temps de réponse très faibles, similaires à ceux procurés par SPA, au prix de temps de retransmission inacceptables pour les connexions les plus longues, comme le montrait la Figure 7-4.

Trafic bidirectionnel

La première expérience portait sur du trafic unidirectionnel. Dans cette section, une connexion longue sur la voie de retour vient s'ajouter. Cette situation est plus à même de se produire, par exemple lorsqu'un utilisateur derrière une ligne ADSL surfe sur Internet tout en uploadant des fichiers chez un opérateur Cloud. Les résultats sont retracés sur les Figures 7-6 et 7-7.

Le premier résultat attendu est une hausse des pertes de SYN en présence de trafic sur la voie de retour. En effet, ce trafic entraîne des pertes de SYN/ACK, tandis que la présence des ACKs dans la file du lien descendant augmente virtuellement la taille de la file et mène à une hausse des pertes totales, comme nous avons pu le voir au Chapitre 6. En moyenne, on observe 25% de pertes supplémentaires, tandis que les retransmission de SYN doublent. En conséquence, les retransmissions de SYN représentent une plus grande partie des temps de retransmissions. En moyenne, les connexions sans pertes de SYN présentent des délais de retransmission inférieurs de 25% à la moyenne.

De même, les temps de réponse sont supérieurs. Ainsi, pour les connexions courtes, FQ CoDel et SPA ont les temps de réponses les plus courts pour les connexions courtes, tandis que SFQ et SPA sont idéaux pour des flux plus longs.

Les résultats relatifs sont similaires à ceux obtenus lors de l'expérience précédente : SPA, FQ CoDel et CoDel ont les RTT les plus faibles, tout en ayant une large part du temps de retransmission due aux pertes de SYN. RED et ARED sont similaires aux deux files FIFO, bien qu'ARED atteigne des RTT et temps de retransmission légèrement inférieurs.

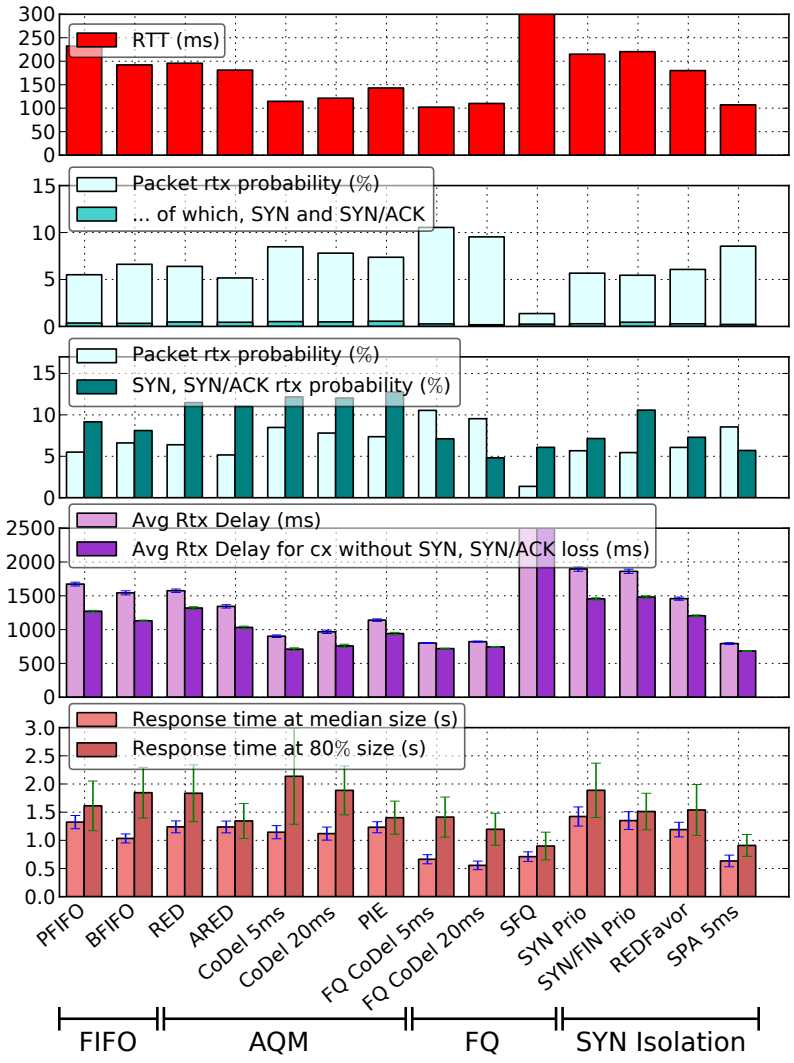


FIGURE 7-6: RTT moyen, pourcentage de perte, temps de retransmission et temps de réponse pour divers algorithmes de files d'attente sous le trafic Type #1 avec une connexion longue dans la direction opposée. Les taux de retransmission de SYN ainsi que les temps de retransmission augmentent fortement. FQ CoDel et SPA continuent d'obtenir les temps de réponse les plus faibles pour les connexions les plus courtes, tandis que SFQ et SPA présentent les meilleures performances pour 80% des connexions.

Les solutions fondées sur la protection des SYN (SYN Prio, SYN/FIN Prio et REDFavor) ne présentent pas d'amélioration significative en comparaison des files FIFO/RED standard, à l'exception d'un RTT légèrement moins élevé, ce qui est peu surprenant : la plupart des pertes se traduisant par de longs temps de retransmission sont dues au trafic sur la voie de retour.

Une fois encore, SPA présente des performances similaires à celles de FQ CoDel, tant en termes de RTT que de temps de retransmission, avec des temps de réponse divisés par deux en comparaison de la plupart des algorithmes AQM.

Les temps de réponse conditionnels sont présentés Figure 7-7. Les résultats sont globalement simi-

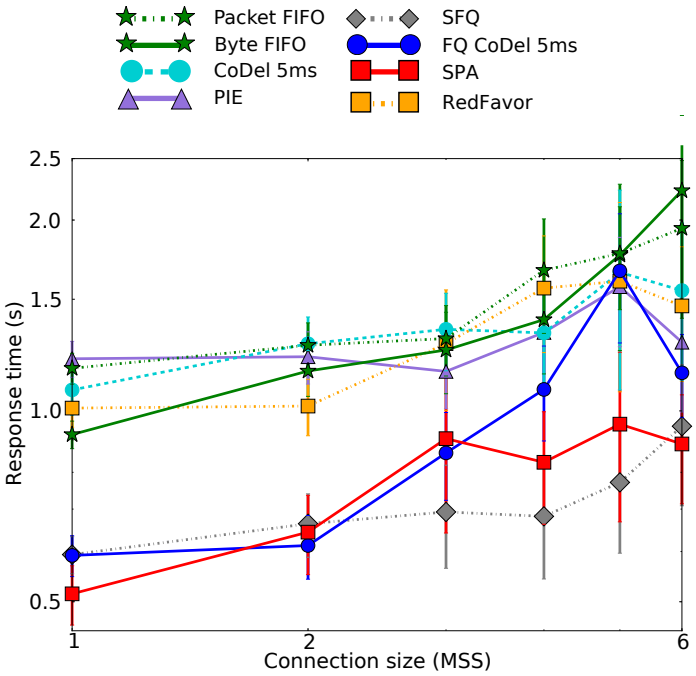


FIGURE 7-7: Temps de réponse des connexions pour divers algorithmes de gestion des files d'attente sous le trafic Type #1 avec une longue connexion sur la voie de retour. L'écart entre AQM et *Fair Queuing* est toujours important, mais de nouvelles différences émergent à l'intérieur de chaque groupe.

laires à l'expérience précédente : même si les connexions présentent des temps de réponse supérieurs au cas sans trafic sur la voie de retour (ce qui est normal si l'on prend en compte le modèle du chapitre 4), deux groupes apparaissent distinctement : FQ CoDel, SFQ et SPA sont bien plus efficaces que les autres mécanismes.

Dans le groupe des AQM, les différences sont faibles à l'exception de SPA qui atteint des performances comparables aux algorithmes de *Fair Queuing*.

Les valeurs pour les connexions les plus longues souffrent d'un biais lié à la distribution des tailles de connexions. Les connexions les plus longues ne se terminent pas durant la période pendant laquelle les mesures sont effectuées. Les connexions qui arrivent à terminer bénéficient de temps de réponse plus faibles (on peut les considérer comme des connexions "chanceuses"). Ce biais explique les faibles temps de réponse pour des connexions de plus de 6 MSS.

7.4.2 Trafic Type #2

Au cours de cette expérience, l'utilisation du Trafic Type #2 correspond à des conditions extrêmes avec une majorité de connexions courtes. Ces conditions ont pour but d'observer le comportement des différents algorithmes lorsqu'on s'approche de leurs limites.

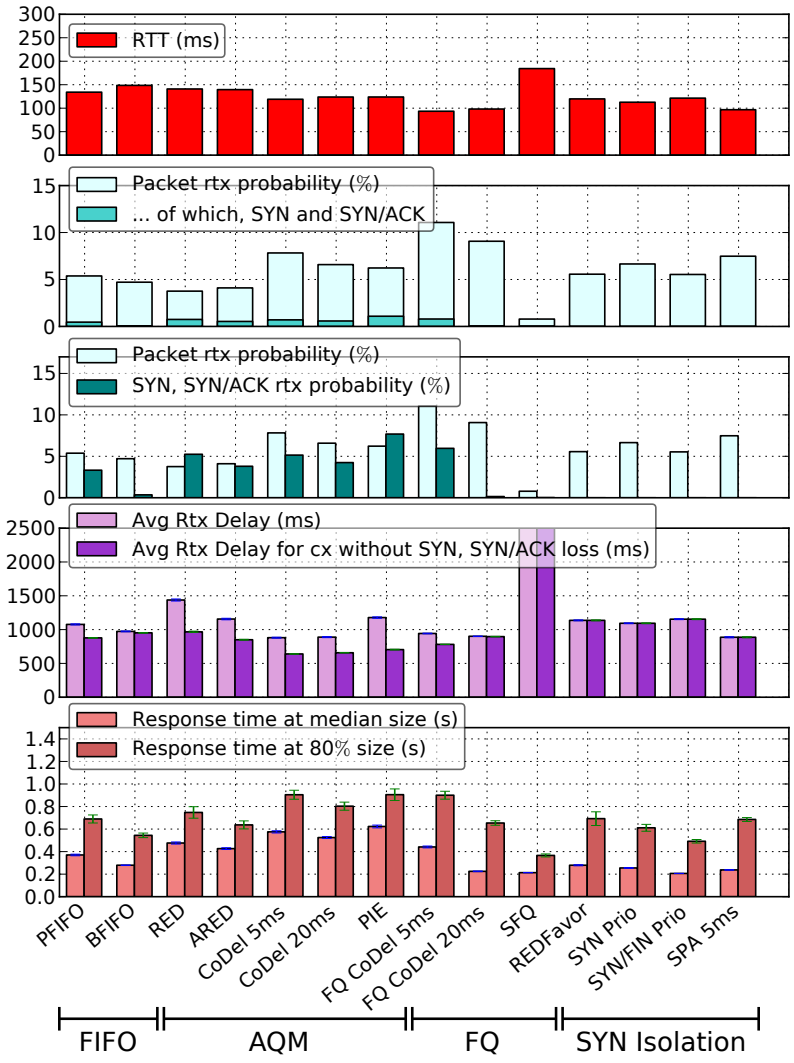


FIGURE 7-8: RTT moyen, pourcentage de perte, temps de retransmission et temps de réponse pour divers algorithmes de file d'attente sous le trafic Type #1. Plus de pertes de SYN qu'avec le trafic Type #1 : délais supérieurs (RTT et retransmission) pour CoDel et PIE. FQ CoDel, SPA, SFQ et SYN/FIN Prio obtiennent les temps de réponses les plus faibles pour les connexions courtes, tandis que SFQ et SYN/FIN Prio présentent les meilleurs résultats pour des connexions plus longues.

La Figure 7-8 montre que, comme précédemment, FQ CoDel, SPA, SFQ obtiennent les temps de réponses les plus bas pour les connexions courtes. SYN/FIN Prio sort curieusement du lot au cours de cette expérience, avec des temps de réponse comparables à ceux des algorithmes de *Fair Queing*. Le RTT moyen est plus faible, entre 100 et 180 ms, en comparaison avec les valeurs de plus de 200 ms observées lors des expériences précédentes. Ce phénomène est dû à la taille des connexions : en effet, avec une moyenne de 3 segments et une fenêtre de congestion initiale de 4 paquets la plupart des connexions n'ont besoin d'envoyer qu'une rafale de paquets pour transmettre l'intégralité des données. De plus, en moyenne, 3 petits segments (SYN,ACK et FIN) sont présents dans la file pour 3 paquets de données, ce qui divise la taille effective des files en paquets par deux, avec les mêmes

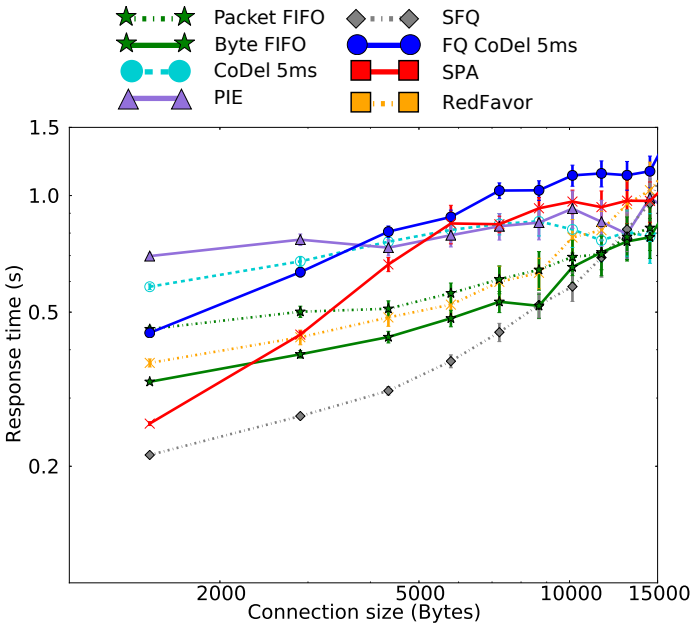


FIGURE 7-9: Temps de réponse des connexions pour divers algorithmes de gestion des files d'attente sous le trafic Type #2 : distribution des tailles à file lourde avec une majorité de très petites connexions. Les résultats sont plus resserrés que pour le trafic #1. FQ CoDel doit être configuré pour des performances acceptables. SPA suit de très près.

effets bénéfiques que constaté au chapitre 4.

Byte FIFO présente des temps de réponse plus bas que la plupart des mécanismes AQM, du fait de son très faible taux de retransmission de SYN.

Les files fondées sur RED présentent des taux de pertes de SYN anormalement élevés. Ce phénomène est en partie dû à l'implantation de RED dans le noyau Linux : l'occupation moyenne de la file est calculée à intervalles réguliers, tandis que lorsque la congestion apparaît, les connexions déjà établies stoppent immédiatement la transmission, ce qui vide rapidement la file. À cause de cet intervalle, l'algorithme met un certain temps à détecter que l'occupation de la file est revenue en dessous du palier de perte, et les paquets suivant continuent à être perdus. Cet effet est donc spécifique à l'implantation de RED sous Linux.

En général, les mécanismes d'AQM causent de nombreuses retransmissions de SYN (jusqu'à 8% des connexions pour PIE), ce qui impacte fortement les temps de réponse des connexions. Étant donné que les connexions sont très courtes, un *timeout* en début de connexion est catastrophique pour le temps de réponse total d'une connexion.

Comme auparavant, les temps de retransmission de SFQ sont très élevés à cause du phénomène de *starvation*. Il est aussi notable que FQ CoDel est peu efficace lorsque la valeur du délai sur lequel les opérations sont effectuées est laissée aux 5 ms par défaut. En effet, dans ce scénario, les SYN arrivent plus vite que le débit du lien. Les SYN restent donc plus longtemps dans la file que ce que

le délai de base de FQ CoDel ne permet, et finissent par être perdus. En modifiant ce délai pour une valeur de 20 ms, les performances attendues sont retrouvées.

Les techniques d'isolation de SYN comme SYN Prio, SYN/FIN Prio et RedFavor améliorent de façon distincte les temps de réponse des connexions en comparaison avec leur équivalent à une seule file, ainsi qu'avec les autres mécanismes AQM. En effet, dans ce type de trafic, le ratio des SYN par rapport aux autres paquets est beaucoup plus élevé ; il en va de même pour la probabilité de perte. Ici, la protection des SYN ne peut donc qu'être bénéfique.

De même, l'utilisation de SPA résulte en une amélioration significative par rapport à l'utilisation de CoDel. Les mesures indiquent des performances supérieures à FQ CoDel (20 ms), avec de faibles RTT et temps de retransmission.

La Figure 7-9 présente les temps de réponse en fonction de la taille du transfert pour les divers mécanismes de gestion de file. L'analyse ci-dessous se focalise sur les deux premiers points de chaque courbe, la majorité des connexions envoyant moins de 3 paquets de données.

Tout d'abord, CoDel et FQ CoDel doivent être correctement configurés pour obtenir des performances acceptables : le simple fait de passer le paramètre fautif à 20 ms résulte en une amélioration de 15% pour CoDel, et divise le temps de réponse par trois dans le cas de FQ CoDel. En contraste, SPA est relativement insensible à ce changement.

De manière peu surprenante, SFQ est très efficace pour les connexions courtes, malgré le problème de *starvation* qui n'affecte qu'un petit nombre de connexions longues.

FQ CoDel correctement configuré suit de près SFQ et SPA, bien que ce jeu de solutions puisse aussi causer de la *starvation* pour les connexions les plus longues. La seule exception ici est REDFavor qui offre des performances correctes pour toutes les tailles de connexion.

Enfin, toutes les solutions à une seule file (FIFO, RED, CoDel et PIE) ont des temps de réponse similaires au *Packet* FIFO traditionnel, avec des temps de réponse plus courts que d'habitude (respectivement 30 à 50% inférieurs à CoDel et PIE, et presque toujours meilleurs que FQ CoDel 5 ms). *Byte* FIFO présente des performances impressionnantes en comparaison des autres solutions. Il s'agit du meilleur algorithme du groupe des politiques à une seule file, et obtient des temps de réponse plus bas que SYN Prio et REDFavor pour toutes les tailles de connexion grâce à son faible taux de retransmission de SYN associé à la robustesse du mécanisme FIFO.

Avec de cette expérience, des résultats très contrastés sont observés. Tout d'abord, les mécanismes fondés sur le délai et RED ne sont clairement pas conçus pour ce type de charge, et présentent des taux de pertes et des temps de réponses supérieurs à une simple file FIFO en paquets.

Byte FIFO continue à montrer des taux négligeables de retransmission de SYN, et reste un très bon choix face à *Packet* FIFO, en dépit d'un RTT légèrement plus élevé.

Ici, les techniques d'isolation de SYN montrent une amélioration significative en termes de temps de réponse, avec de très bonnes performances pour SYN/FIN Prio et SFQ. Ces derniers sont suivis de très près par FQ Codel (correctement configuré), et SPA, qui ont aussi pour avantage d'obtenir des RTT et des temps de retransmission très bas, ce qui les rend particulièrement adaptés au trafic interactif.

7.4.3 Trafic Type #3

Afin de confirmer les résultats obtenus dans les sections précédentes, une dernière expérience est effectuée sur un lien à 10 Mb/s avec des paramètres similaires au trafic Type #1 sans connexion sur la voie de retour. La Figure 7-10 présente les résultats de cette expérience.

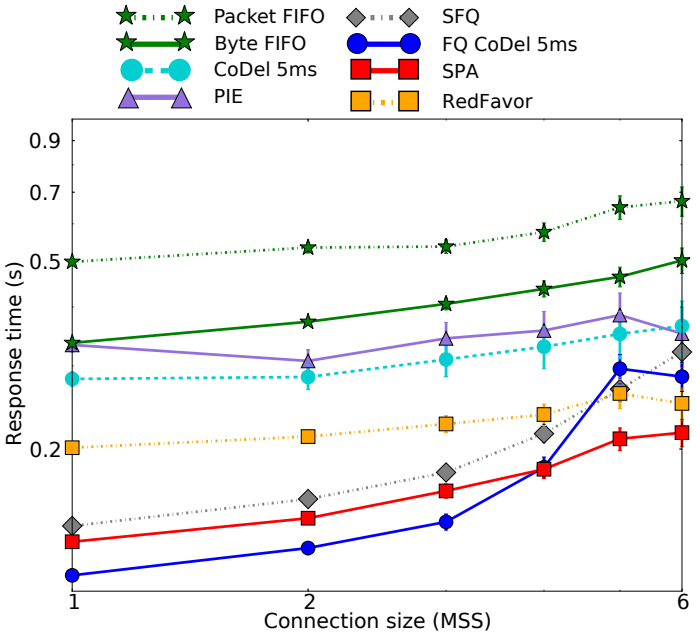


FIGURE 7-10: Temps de réponse conditionnels des connexions pour divers algorithmes de gestion des files en utilisant le trafic Type #3 sur un lien à 10 Mb/s. FQ CoDel, SPA et SFQ présentent toujours les meilleures performances.

Cette expérience est très proche de celle décrite en section 7.4.1, et les résultats sont similaires : FQ CoDel, SPA et SFQ se détachent nettement des autres algorithmes. Bien qu'étant le deuxième plus mauvais mécanisme, *Byte* FIFO présente des temps de réponse 30% inférieurs à *Packet* FIFO.

La différence principale se situe dans le fait qu'ici, PIE et CoDel sont nettement plus efficaces que FIFO (en paquets ou octets). De plus, RED Favor résulte désormais en de meilleures performances que les autres mécanismes AQM, et dépasse même SFQ et FQ CoDel pour les connexions les plus longues. Ceci est dû au fait que le produit délai bande passante est plus élevé que dans le cas du Trafic Type #1 et #2. Ainsi, les résultats sont plus lissés et PIE et CoDel entrent dans des conditions

plus adaptées à leur fonctionnement. Pour des liens de plus grande capacité, cette tendance devrait se poursuivre, bien qu'avec une intensité plus faible.

7.4.4 Conclusion des résultats expérimentaux

En se référant aux quatre expériences décrites ci-dessus, *Byte* FIFO peut être considéré comme un concurrent intéressant à *Packet* FIFO. Toutefois, cette solution est souvent plus complexe à implanter que les autres solutions testées et n'est pas toujours disponible.

Bien que les solutions fondées sur le délai compensent leurs fortes pertes de SYN par des RTT plus bas et des temps de retransmission plus rapides, les temps de réponse restent similaires aux autres politiques, et pourraient être fortement améliorés avec plus de discernement dans la manière dont les paquets sont traités.

Enfin, isoler les SYN et les FIN dans des files séparées présente, dans le pire des cas, des performances similaires à leur équivalent à simple file d'attente : dans trois scénarios sur quatre, une amélioration notable est observée pour les solutions basées sur RED et FIFO. En revanche, SPA présente constamment des résultats toujours supérieurs à CoDels, si bien qu'il a fallu le comparer à des algorithmes de *fair queuing* qui sont nettement plus complexes. Dans certains cas, SPA présente des performances supérieures à ces politiques. Tout au long des expérimentations, cette solution représente de loin le meilleur compromis entre performance et complexité.

7.5 Conclusion

Dans ce chapitre, nous avons pu évaluer l'impact des retransmissions de SYN sur les temps de réponse des connexions en se référant à des traces issues de liens de transit, ainsi que des expériences extensives sur banc d'essai. L'analyse des traces publiques montre que les pertes de SYN ou SYN/ACK sont bien plus probables que les pertes d'autres segments, ce qui influence fortement les temps de réponses, notamment pour les petites connexions.

Le taux de retransmission de SYN constaté est en réalité bien supérieur à ce qu'il est possible d'imaginer comparativement au taux de perte de paquets de données. Ce point reste à élucider dans l'avenir.

Atteindre des temps de réponse faibles requiert donc de protéger les SYN et SYN/ACK des pertes. Pour ce faire, SPA utilise deux files d'attente gérées par CoDel — l'une à haute priorité pour les paquets SYN et FIN et l'autre à faible priorité pour les segments de données et les ACKs.

Des expériences menées sur un banc d'essai permettent de comparer les politiques de gestion de file

d'attente les plus connues pour 3 conditions de trafic différentes. Les mesures effectuées montrent que la plupart des politiques de gestion de file active donnent des résultats très similaires : les délais dans la file sont bas, au prix de pertes de SYN significatives. Les politiques de *fair queuing*, quant à elles, conduisent à bonnes performances, bien que FQ CoDel puisse nécessiter plus de configuration dans des conditions de trafic extrêmes. SFQ obtient des temps de réponse très faibles, mais la plupart des pertes sont récupérées à la suite de *timeout*s multiples. Enfin, l'inconvénient majeur des algorithmes de *fair queuing* est leur complexité et l'occupation mémoire en comparaison des mécanismes d'*AQM*. Sur ce point, *Byte* FIFO reste une solution simple qui se traduit par de très bonnes performances.

SPA atteint un compromis entre la simplicité et l'empreinte mémoire de la majorité des algorithmes d'AQM, tout en atteignant les délais faibles de CoDel et des performances similaires aux politiques de *fair queuing*. Cet algorithme répond au besoin de forts débits et faibles délais requis par tout trafic interactif : ce type de connexion ne peut pas se permettre d'attendre 3 secondes de *timeout* causés par une perte de SYN.

Chapitre 8

Conclusions, Perspectives et Anti-Perspectives

8.1 Anti-Perspectives

Ce document est aussi défini par ce qu'il ne contient pas. Cette section répertorie les diverses idées finalement écartées, déceptions et autres pistes infructueuses ayant mené aux diverses contributions présentées dans ce manuscrit.

En premier lieu, concernant les expérimentations, l'utilisation de plateformes de tests de type *Grid5000* se révèle contraignante à l'usage de par leur système de réservation, qui empêche souvent de réserver des machines connectées au même commutateur. Ce fonctionnement force à revoir la topologie de l'expérience à chaque lancement. Il est ainsi difficile de parler de résultats reproductibles dans ces conditions.

La mise en pratique de certains principes peut se révéler hasardeuse. Ainsi, modifier directement la pile TCP/IP est la solution qui peut sembler la plus instinctive au premier abord : simple à déployer, ne nécessite pas d'intervention en cœur de réseau et peut être aisément implantée sur simulateur. C'est ainsi qu'un algorithme fondé sur l'espacement entre les ACKs a été développé, pour lequel les simulations mettaient en évidence de très bons résultats. Toutefois, la variance des mesures effectuées lors du passage sur banc d'essai a démontré que la solution n'était pas viable en conditions réelles. Cette problématique a ainsi grandement contribué à la méthodologie expérimentale décrite chapitre 3 et suivie tout au long de ces travaux.

En ce qui concerne les contributions de ce manuscrit, la protection des SYN est issue de nombreux tâtonnements. Afin de protéger les SYN ainsi que les paquets en fin de connexion, nous avons tout

d'abord tenté une solution de proxy qui injectait des paquets de type *KEEPALIVE*, afin de générer les acquittements dupliqués nécessaires à la retransmission. La première version reposait sur un socket locale connectée à un deuxième socket TCP. L'encapsulation de TCP dans TCP a mené à de fortes interférences entre les deux algorithmes de contrôle de congestion, menant à des retransmissions non sollicitées de part et d'autre, avec un impact désastreux sur les performances. De plus, de fortes contraintes apparaissaient, que ce soit l'échappement nécessaire des données additionnelles, les performances réduites dues à l'analyse paquet par paquet du flux ou encore l'émission de *Keepalive*, qui devait rester unidirectionnelle, au risque d'entraîner des boucles infinies dans l'algorithme.

Enfin, une dernière observation plus personnelle concerne la pratique de la recherche en elle-même. Dans un champ très actif tel que Multipath TCP, il est important de contacter et de se rapprocher des équipes travaillant sur le sujet afin d'éviter de conclure de longues semaines de travail par la découverte d'un article fraichement publié par une autre équipe sur le même sujet.

8.2 Conclusions

Malgré l'émergence de nouvelles technologies, les liens asymétriques restent la norme en ce qui concerne le raccordement final des particuliers, que ce soit dans le cas de fibre optique (dans le cas des Fournisseurs d'Accès à Internet grand public) ou de liens cellulaires, ces derniers présentant de plus des débits et délais variables dans le temps, et inconnus de l'utilisateur.

Si les débits sont en constante augmentation, les tampons surdimensionnés restent très fréquents sur le réseau. De plus, dans le cas de technologies à débit variable comme le WiFi, les tampons restent les mêmes quelle que soit la bande passante, ajoutant du *bufferbloat* à des débits réduits, ce qui dégrade considérablement la qualité du service. Une approche fondée sur les délais et non sur la quantité de données permettrait dans ce cas d'offrir une latence raisonnable quelles que soient les conditions.

Comme nous avons pu le voir dans l'état de l'art des solutions existantes, l'avenir reste toujours à TCP et ses à variantes, fondées sur le cadencement des accusés de réception pour réguler la transmission. Nous avons toutefois pu observer que les variantes qui reposent sur ce mécanisme pouvaient subir et causer de fortes latences lorsqu'employées dans un tampon suivant une politique "Premier arrivé, premier servi". De plus, les variantes actuelles, adaptées aux liens à fort produit délai-bande passante, réagissent souvent plus mal que TCP New Reno dans le cas de connexions antiparallèles.

8.2.1 Expériences et reproductibilité

Tout au long de ce travail de recherche, l'accent a été mis sur des expériences reproductibles, dans un environnement isolé afin de pouvoir étudier avec précision l'impact de chaque modification apportée au standard.

Ces réflexions ont mené à la réalisation d'un banc d'essai extensible, permettant l'évaluation d'un grand nombre de solutions – de bout-en-bout et en cœur de réseau – dans des conditions proches du réel. Ainsi, les variantes de TCP les plus utilisées ont pu être évaluées, en combinaison avec un grand nombre de politiques de file d'attente en cœur de réseau.

Une plateforme de redistribution des environnements de tests accompagne ce banc d'essai. Celle-ci permet d'intégrer facilement la mise à disposition des expériences au sein d'une application de gestion des publications.

8.2.2 Dynamiques des connexions

À travers une modélisation des interactions entre connexions et un ensemble d'expériences couvrant un grand nombre de situations, nous avons pu étudier de manière approfondie les phénomènes à l'œuvre sur les liens asymétriques.

Ces analyses mènent à des conclusions similaires : pour des algorithmes de contrôle de congestion basés sur le cadencement des accusés de réception, si la file d'attente sur l'*uplink* suit une politique FIFO et qu'elle est de plus surdimensionnée, le débit chute considérablement pour le *download* alors que l'*upload* occupe le lien montant dans sa totalité.

L'effet de balancier est à l'origine de ces perturbations : la majorité du temps, un seul des deux tampons peut être occupé, tandis que le lien opposé est sous-utilisé. Lorsque les deux tampons sont correctement dimensionnés, le balancier oscille en permanence, et chaque lien est correctement utilisé. Cependant, lorsque le tampon montant est surdimensionné, le balancier passe plus de temps du côté montant, entraînant une sous-utilisation du lien descendant.

Ce phénomène est causé par le cadencement des ACKS dans une file d'attente FIFO, la plupart des variantes qui fondent leur contrôle de congestion sur les pertes de paquets sont donc touchées. TCP Vegas, qui utilise les délais afin de réguler la transmission, est la seule version étudiée qui permette d'atteindre des débits corrects en *download* lorsqu'utilisé sur le lien montant. Toutefois, les connexions utilisant Vegas souffrent de fortes baisses de débit, ce qui reste un problème mineur dans le cas qui nous intéresse, en comparaison des avantages apportés.

La solution idéale se situe en cœur de réseau avec l'utilisation de politiques de files d'attente adaptées à ce problème. Un premier pas réside dans l'utilisation de files FIFO bien dimensionnées, ce qui n'est

pas toujours applicable, notamment dans les cas où le débit et les délais du lien sont variables. Une solution plus récente est la mise en place de files d'attente conçues afin de minimiser les délais, que ce soit à travers l'utilisation de *flow queuing* afin de favoriser les flux les plus courts, ou par la mise en place de CoDel/PIE (voire FQ CoDel) qui ont été spécifiquement conçues afin de réduire les délais tout en apportant une plus grande équité aux connexions partageant le même lien. Dans certains cas, nous avons même pu observer qu'une simple file FIFO dimensionnée en octets pouvait améliorer grandement les performances.

8.2.3 Solutions originales

Au-delà des solutions évoquées précédemment aux problèmes liés aux tampons surdimensionnés en cœur de réseau, une approche atypique peut être développée. Cette approche repose sur le fait que la plupart des tampons sont conçus pour accepter un certain nombre de paquets quelle que soit leur taille.

En envoyant un flux de petits paquets sur le réseau, il est ainsi possible d'occuper des emplacements de la file en permanence. Ces petits paquets permettent ainsi de réduire virtuellement la taille du tampon tout en ayant un impact négligeable sur les débits.

Cette solution présente ainsi le paradoxe suivant : envoyer du trafic supplémentaire est bénéfique pour toutes les connexions partageant le tampon. Si ce type de comportement peut être mal vu par les opérateurs lorsqu'effectué intentionnellement, une forte charge sur le lien peut également diminuer artificiellement la taille du tampon par la présence des ACKs sur la voie de retour. Ainsi, un trafic important dans le sens descendant peut entraîner une utilisation plus équitable du lien et réduire les latences observées.

8.2.4 D'autres causes de latences

Les latences des connexions TCP ne sont pas toujours dues à la présence de tampons surdimensionnés. Lorsque des pertes ne peuvent être récupérées par les mécanismes de *Fast Recovery* et *Fast Retransmit*, les segments sont retransmis à la suite de l'expiration d'un *timer*. Les valeurs minimales choisies pour ces temporisateurs sont très souvent élevées (certains diront même, d'un autre âge), notamment pour les paquets en début de connexion qui utilisent systématiquement un *timeout* en cas de perte : SYN, SYN/ACK, DNS etc.

Les solutions de type *fair queuing* sont très efficaces pour protéger les premiers paquets d'une connexion, mais présentent souvent des problèmes de *starvation* et de *bufferbloat* pour les connexions les plus longues, en plus d'une occupation mémoire et processeur démultipliée. Une solution utilisant seulement deux files d'attente, chacune gérée par CoDel permet de résoudre ces problèmes. Les SYN

sont envoyés dans une file à priorité haute, tandis que le reste du trafic est redirigé dans la file à priorité basse. Ce fonctionnement permet de protéger les premiers paquets d'une connexion, tout en procurant de faibles délais par l'utilisation de CoDel. Les performances observées sont équivalentes à celles de FQ CoDel, pour une complexité fortement réduite (deux files d'attente au lieu de 1024 dans l'implantation par défaut).

Contrairement à ce que nous avons pu observer dans les chapitres précédents, ici, une politique FIFO en octets réduit spécifiquement ce type de latences, car les paquets de contrôle, souvent plus petits, sont presque toujours acceptés. Toutefois, les latences moyennes observées au cours de la connexion sont souvent plus élevées que dans le cas d'une file FIFO en paquets.

8.3 Perspectives

Le modèle proposé dans le chapitre 4 donne une bonne explication des interactions entre deux connexions antiparallèles sur un lien asymétrique avec des files régies par une politique FIFO. Une des premières perspectives pourrait être d'étendre ce modèle et de quantifier les bascules dans le cas d'un agrégat de connexions. Une deuxième approche réside dans l'intégration de la phase de *slow start* au sein du modèle afin d'évaluer l'impact d'une nouvelle connexion sur un système existant et inversement.

Dans un deuxième temps, il pourrait être intéressant d'intégrer de nouvelles politiques de files d'attente comme CoDel ou PIE, afin de déterminer analytiquement leur impact sur le trafic. De même, certaines solutions de bout en bout sont conçues afin de réduire l'impact de blocages causés par l'*upload*. Ces dernières mériteraient d'être intégrées à l'étude.

Enfin, à travers l'utilisation des champs *Timestamp Value* et *Timestamp Echo Reply* spécifiés par la RFC 1323 [1], il devrait être possible de détecter de quel coté du réseau se déroule l'épisode de congestion, et de modifier l'augmentation de la fenêtre de congestion en conséquence.

En parallèle, les travaux effectués sur les expérimentations reproductibles en environnement isolé seront poursuivis, que ce soit par l'amélioration des conditions de redistribution des sources, ou l'adaptation du banc d'essai à d'autres solutions à plus grande échelle telles que les plateformes d'expérimentation.

En ce qui concerne l'envoi de petits paquets afin de résoudre les problèmes de *bufferbloat*, il reste à concevoir un outil s'adaptant automatiquement aux délais dans le réseau afin de ne pas le surcharger.

Les tests sur SPA doivent être améliorés afin de tester sur des liens plus gros, de type fibre optique, avec un trafic plus important. De surcroît, combiner les principes suivis par SPA à d'autres politiques de gestion des files d'attente telles qu'EFD permettrait de répondre au problème “des souris et des

éléphants" en plus de proposer de faibles latences et de garantir la réactivité de l'établissement de la connexion.

Enfin, à plus long terme, il serait intéressant de porter la réflexion aux problématiques de chemins multiples comme Multipath TCP. Ce mode de fonctionnement pose des problématiques intéressantes, notamment au travers de l'utilisation de liens présentant des caractéristiques souvent très différentes, variables dans le temps (ex : lien cellulaire et WiFi).

Index

Liste des publications

[pub1] T. Braud, M. Heusse, and A. Duda. SYN Priority with Active Queue Management for Short TCP Response Times. 2016. En cours de soumission.

[pub2] T. Braud and M. Heusse. Étude des variantes de TCP dans le cas de connexions antiparallèles. In *Algotel 2016*, 2016.

[pub3] T. Braud, M. Heusse, and A. Duda. Dynamics of Two Antiparallel TCP Connections on an Asymmetric Link. In *IEEE International Conference on Communications*, Mai 2016. Taux d'acceptation 39%.

[pub4] T. Braud, M. Heusse, and A. Duda. TCP over large buffers : When adding traffic improves latency. In *26th International Teletraffic Congress (ITC)*, pages 1–8, Sept 2014. Taux d'acceptation 37.5%.

Bibliographie

[1] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992. Obsoleted by RFC 7323.

[2] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), October 1996.

[3] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001. Updated by RFCs 4301, 6040.

[4] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, and P. Hurtig. Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP). RFC 5827 (Experimental), May 2010.

[5] Nandita Dukkipati, Neal Cardwell, Yuchung Cheng, and Matt Mathis. Tail loss probe (tlp) : An algorithm for fast recovery of tail losses. Internet-Draft draft-dukkipati-tcpm-tcp-loss-probe-01, IETF Secretariat, February 2013. `http://www.ietf.org/internet-drafts/draft-dukkipati-tcpm-tcp-loss-probe-01.txt`.

[6] Micro transport protocol. `http://blog.bittorrent.com/2009/10/05/changing-the-game-with-%CE%BCtp/`. Accessed : 2015-08-08.

[7] Fast performance transport software. `http://asperasoft.com/technology/transport/fasp/`. Accessed : 2015-08-08.

[8] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006. Updated by RFCs 5595, 5596, 6335, 6773.

[9] Chromium blog : Experimenting with quic. `https://blog.chromium.org/2013/06/experimenting-with-quic.html`. Accessed : 2015-08-08.

[10] Spdy : An experimental protocol for a faster web. `https://www.chromium.org/spdy/spdy-whitepaper`. Accessed : 2015-08-2.

[11] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext transfer protocol version 2. Internet-Draft draft-ietf-httpbis-http2-16, IETF Secretariat, November 2014. `http://www.ietf.org/internet-drafts/draft-ietf-httpbis-http2-16.txt`.

[12] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013.

[13] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335, 7053.

[14] Understanding delay in packet voice networks. `https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/5125-delay-details.html`. Accessed : 2016-04-08.

[15] J.C.R. Licklider. Man-computer symbiosis. *Human Factors in Electronics, IRE Transactions on*, HFE-1(1) :4–11, March 1960.

[16] J. C. R. Licklider and Welden E. Clark. On-line man-computer communication. In *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*, AIEE-IRE '62 (Spring), pages 113–128, New York, NY, USA, 1962. ACM.

[17] JCR Licklider. Memorandum for : Members and affiliates of the intergalactic computer network. april 23, 1963.

[18] L. Kleinrock. *Information Flow in Large Communication Nets.* Ph.d. thesis proposal, Massachusetts Institute of Technology, May 1961.

[19] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. Brief History of the Internet. `http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet`.

[20] V. Cerf and R.E. Kahn. A protocol for packet network intercommunication. *Communications, IEEE Transactions on*, 22(5) :637–648, May 1974.

[21] V. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675, December 1974.

[22] Vint Cerf. TCP version 2 specification, March 1977.

[23] J. Postel. Comments on Internet protocols and TCP, August 1977.

[24] J. Postel. DoD standard Internet Protocol. RFC 760, January 1980. Obsoleted by RFC 791, updated by RFC 777.

[25] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.

[26] J. Postel. Internet Control Message Protocol. RFC 792 (INTERNET STANDARD), September 1981. Updated by RFCs 950, 4884, 6633, 6918.

[27] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[28] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, January 1984.

[29] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18(4) :314–329, August 1988.

[30] Raj Jain and K. K. Ramakrishnan. Congestion avoidance in computer networks with a connectionless network layer : Concepts, goals and methodology, 1987.

[31] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), January 1997. Obsoleted by RFC 2581.

[32] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[33] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.

[34] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582 (Proposed Standard), April 2012.

[35] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. RFC 2883 (Proposed Standard), July 2000.

[36] Lawrence S. Brakmo, Sean W. O'malley, and Larry L. Peterson. Tcp vegas : New techniques for congestion detection and avoidance. In *In SIGCOMM*, 1994.

[37] Richard J. La, Jean Walrand, and Venkat Anantharam. Issues in tcp vegas, 2001.

[38] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524 vol.4, March 2004.

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

[40] Geoff HustonGeoff Huston. The ISP Column – Faster. `http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet`.

[41] S. Belhareth, D. Lopez-Pacheco, L. Sassatelli, D. Collange, and G. Urvoy-Keller. Understanding synchronization in tcp cubic. In *Teletraffic Congress (ITC), 2014 26th International*, pages 1–9, Sept 2014.

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

[43] Carlo Caini and Rosario Firrincieli. Tcp hybla : a tcp enhancement for heterogeneous networks. *INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS AND NETWORKING*, 22, 2004.

[44] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood : Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, pages 287–297, New York, NY, USA, 2001. ACM.

[45] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[46] Curtis Villamizar and Cheng Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*, 24(5) :45–60, October 1994.

[47] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *SIGCOMM Comput. Commun. Rev.*, 34(4) :281–292, August 2004.

[48] S. Hassayoun and D. Ros. Loss synchronization, router buffer sizing and high-speed tcp versions : Adding red to the mix. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, pages 569–576, Oct 2009.

[49] Damon Wischik and Nick McKeown. Part i : Buffer sizes for core routers. *SIGCOMM Comput. Commun. Rev.*, 35(3) :75–78, July 2005.

[50] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Part iii : Routers with very small buffers. *SIGCOMM Comput. Commun. Rev.*, 35(3) :83–90, July 2005.

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

[52] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED : An Algorithm for Increasing the Robustness of RED's Active Queue Management. Technical report, August 2001.

[53] Wu-chang Feng, Kang G. Shin, Dilip D. Kandlur, and Debanjan Saha. The blue active queue management algorithms. *IEEE/ACM Trans. Netw.*, 10(4) :513–528, August 2002.

[54] J. Nagle. RFC 970 On Packet Switches with Infinite Storage. `http://www.ietf.org/rfc/rfc970.txt`, December 1985.

[55] L. Kleinrock and R. R. Muntz. Processor sharing queueing models of mixed scheduling disciplines for time shared system. *J. ACM*, 19(3) :464–482, July 1972.

[56] Ellen Louise Hahne. Round robin scheduling for fair flow control in data communication networks. 1986.

[57] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Symposium Proceedings on Communications Architectures &Amp ; Protocols*, SIGCOMM '89, pages 1–12, New York, NY, USA, 1989. ACM.

[58] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis. Weighted round-robin cell multiplexing in a general-purpose atm switch chip. *Selected Areas in Communications, IEEE Journal on*, 9(8) :1265–1279, Oct 1991.

[59] Liang Guo and Liang Ibrahim Matta. The war between mice and elephants, 2001.

[60] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A First Look at Traffic on Smartphone. In *Internet Measurement Conference*, New York, NY, USA, 2010. ACM.

[61] Nicolas Kuhn, Emmanuel Lochin, and Olivier Mehani. Revisiting Old Friends : is CoDel Really Achieving What RED Cannot ? In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing*. ACM, August 2014.

[62] M. Mellia and H. Zhang. TCP Model for Short Lived Flows. *IEEE Communications Letters*, 6(2) :85–87, February 2002.

[63] Delia Ciullo, Marco Mellia, and Michela Meo. Two Schemes to Reduce Latency in Short Lived TCP Flows. *IEEE Communications Letters*, 13(10) :806–808, 2009.

[64] Pascal Anelli, Fanilo Harivelo, and Richard Lorion. TCP SYN Protection : an Evaluation. In *Proc. Eleventh International Conference on Networks. ICN'12*, February 2012.

[65] Dragana Damjanovic, Philipp Gschwandtner, and Michael Welzl. Why Is This Web Page Coming Up so Slow ? Investigating the Loss of SYN Packets. In Dragana Damjanovic, Philipp Gschwandtner, and Michael Welzl, editors, *Networking 2009*, pages 895–906, Berlin, Heidelberg, 2009.

[66] Ashish Vulimiri, Oliver Michel, P Brighten Godfrey, and Scott Shenker. More is Less : Reducing Latency via Redundancy . In *Proc. 11th ACM Workshop on Hot Topics in Networks*, pages 13–18, New York, New York, USA, 2012. ACM Press.

[67] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing Web Latency : the Virtue of Gentle Aggression. In *Proceedings of the ACM SIGCOMM 2013 Conference*. ACM Press, August 2013.

[68] S. Floyd. Limited slow-start for tcp with large congestion windows. RFC 3742, RFC Editor, March 2004.

[69] R. Sallantin, C. Baudoin, E. Chaput, F. Arnal, E. Dubois, and A. L. Beylot. Initial spreading : A fast start-up tcp mechanism. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 492–499, Oct 2013.

[70] J.K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros. Network coding meets tcp. In *INFOCOM 2009, IEEE*, pages 280–288, April 2009.

[71] M. Heusse, G. Urvoy-Keller, A. Duda, and T.X. Brown. Least attained recent service for packet scheduling over wireless lans. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, pages 1–9, June 2010.

[72] K. Avrachenkov, U. Ayesta, P. Brown, and E. Nyberg. Differentiation between short and long tcp flows : predictability of the response time. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 762–773 vol.2, March 2004.

[73] Jinbang Chen, M. Heusse, and G. Urvoy-Keller. Analysis of the early flow discard (efd) discipline in 802.11 wireless lans. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pages 1–9, June 2012.

[74] S. Cheshire. It's the Latency, Stupid. `http://rescomp.stanford.edu/~cheshire/rants/Latency.html`.

[75] B. Turner. Has AT&T Wireless Data Congestion Been Self-Inflicted? `http://blogs.broughturner.com/2009/10/is-att-wireless-data-congestion-selfinflicted.html`.

[76] J. Gettys. Bufferbloat : Dark Buffers in the Internet. *IEEE Internet Computing*, 15(3) :96–96, May 2011.

[77] T. Høiland-Jørgensen. The State of the Art in Bufferbloat Testing and Reduction on Linux. `http://www.ietf.org/proceedings/86/slides/slides-86-iccrg-0.pdf`, March 2013.

[78] Browser statistics. `http://www.w3schools.com/browsers/browsers_stats.asp`. Accessed : 2015-08-2.

[79] ios : Multipath tcp support in ios 7. `https://support.apple.com/en-us/HT201373`. Accessed : 2015-08-08.

[80] Sctp homepage. `http://www.sctp.be/`. Accessed : 2015-08-08.

[81] Christian Collberg, Todd Proebsting, Gina Moraila, Zuoming Shi, and Alex M Warren. Measuring Reproducibility in Computer Systems Research. pages 1–37, 2014.

[82] Lucas Di Cioccio, Renata Teixeira, and Catherine Rosenberg. Is It Me? Understanding the Impact of the Home Network on End-to-End Measurements. pages 33–34.

[83] Lampros Kalampoukas, Anujan Varma, and K. K. Ramakrishnan. Improving TCP Throughput over Two-Way Asymmetric Links : Analysis and Solutions. *SIGMETRICS Perform. Eval. Rev.*, 26(1) :78–89, 1998.

[84] Fatma Louati, Chadi Barakat, and Walid Dabbous. Handling Two-Way TCP Traffic in Asymmetric Networks. In *7th IEEE International Conference on High Speed Networks and Multimedia Communications HSNMC'04*, number 3079 in LNCS, pages 233–243. Springer-Verlag, 2004.

[85] Hari Balakrishnan, Randy H. Katz, and Venkata N. Padmanbhan. The Effects of Asymmetry on TCP Performance. *Mob. Netw. Appl.*, 4(3) :219–241, 1999.

[86] Scott Shenker, Lixia Zhang, and David D. Clark. Some Observations on the Dynamics of a Congestion Control Algorithm. *SIGCOMM Comput. Commun. Rev.*, 20(5) :30–39, 1990.

[87] Lixia Zhang, Scott Shenker, and David D. Clark. Observations on the Dynamics of a Congestion Control Algorithm : the Effects of Two-Way Traffic. *SIGCOMM Comput. Commun. Rev.*, 21(4) :133–147, 1991.

[88] Denis Collange. Performance Model of Opposite TCP Connections on Asymmetric Capacities. In *10th Symposium on Performance Evaluation of Computer and Telecommunications Systems (SPECTS 2007)*, San Diego (USA-CA), July 2007.

[89] Martin Heusse, Sears A Merritt, Timothy X Brown, and Andrzej Duda. Two-way TCP Connections : Old Problem, New Insight. *ACM SIGCOMM Computer Communication Review*, 41(2) :5–15, 2011.

[90] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the 2012 ACM Internet Measurement Conference*, IMC'12, pages 329–342, New York, NY, USA, 2012. ACM.

[91] IPMT Test Suite. `http://ipmt.forge.imag.fr`.

[92] Luigi A. Grieco and Saverio Mascolo. Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control. *SIGCOMM Comput. Commun. Rev.*, 34(2) :25–38, April 2004.

[93] Browser os statistics. `http://www.w3schools.com/browsers/browsers_os.asp`. Accessed : 2015-14-1.

[94] Codel overview. `http://www.bufferbloat.net/projects/codel/wiki`. Accessed : 2015-14-1.

[95] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 329–342, New York, NY, USA, 2012. ACM.

[96] CACM Staff. Bufferbloat : What's Wrong with the Internet ? *Communications of the ACM*, 55(2) :40–47, 2012. A discussion with Vint Cerf, Van Jacobson, Nick Weaver, and Jim Gettys.

[97] M. Allman. Comments on Bufferbloat. *ACM SIGCOMM Computer Communication Review*, January 2013.

[98] I.A. Rai, E.W. Biersack, and G. Urvoy-Keller. Size-based scheduling to improve the performance of short tcp flows. *Network, IEEE*, 19(1) :12–17, Jan 2005.

[99] Kathleen Nichols and Van Jacobson. Controlling Queue Delay. *ACM Queue*, 10(5) :20–34, May 2012.

[100] Rong Pan, Preethi Natarajan, Fred Baker, and Greg White. PIE : a Lightweight Control Scheme to Address the Bufferbloat Problem. Internet-Draft draft-ietf-aqm-pie-00, October 2014.

[101] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3) :67–82, July 1997.

[102] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput : A simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.*, 28(4) :303–314, October 1998.

[103] Httparchive. `http://httparchive.org/interesting.php`. Accessed : 2015-06-04 – Statistics for 2015-03-15.

[104] FreeBSD SYN Cache Source. `sys/netinet/tcp_syncache.c`. Accessed on : FreeBSD-9.3.

[105] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proc. INFOCOM 2003*, pages 1742–1751 vol.3. IEEE, 2000.

[106] T. Høiland-Jørgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. FlowQueue-CoDel. IETF Internet draft, March 2014. draft-hoeiland-joergensen-aqm-fq-codel-00.

[107] Linux traffic control. `http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html`. Accessed : 2015-03-18.

[108] P.E. McKenney. Stochastic Fairness Queueing. *Proceedings of IEEE INFOCOM*, 1990.

Résumé

L'utilisation de TCP sur des liens asymétriques entraîne fréquemment des débits plus faibles qu'attendus au point de nuire sensiblement à la qualité de service ressentie par l'utilisateur. Ces baisses de performances peuvent prendre diverses formes parmi lesquelles une forte latence en début de connexion, une sous-utilisation de la capacité du lien ou encore des latences excessivement hautes pour l'ensemble de la connexion.

Afin de contrer ces effets, plusieurs approches sont possibles, que ce soit de bout-en-bout par des modifications de la pile TCP/IP ou en cœur de réseau avec divers mécanismes d'ordonnancement.

L'objectif de cette thèse est d'explorer comment un résultat similaire à celui obtenu par des méthodes d'ordonnancement au goulot d'étranglement peut être obtenu en travaillant de bout-en-bout, c'est-à-direlà où les ressources de calcul et de mémoire sont les plus abondantes. Ce questionnement est accompagné par une analyse en profondeur des phénomènes causant une dégradation des performances, ainsi que l'évaluation des solutions existantes. Finalement, des solutions nouvelles, en cœur de réseau ainsi que de bout en bout, ont été apportées et testées sur banc d'essai.

Mots clés : TCP/IP ; Ordonnancement ; Liens asymétriques

Abstract

Using TCP on asymmetric links may lead to unexpected and significant performance drops, severely degrading user experience. Those performance drops can come in various forms, among which a huge latency at the beginning of a connection, under-utilization of link capacities, or even excessive delays for the whole connection.

In order to prevent those effects to happen, several approaches exist, either end-to-end through changes in the TCP/IP stack, or in the network core with a collection of scheduling algorithms.

The first goal of this thesis is to explore if and how an end-to-end policy (*i.e* where CPU and memory resources are the most abundant) can achieve similar results as operating in the core of the network. Secondly, we provide an in-depth analysis of the root cause of the performances drops, and evaluate existing algorithms. Finally, new solutions, both end-to-end and in the core of the network, are brought forward and tested in testbench networks.

Keywords : TCP/IP ; Scheduling ; Asymmetric Links