Évolution des protocoles de transport du point de vue de l'équité

Romuald Corbel

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Romuald CORBEL

Évolution des protocoles de transport du point de vue de l'équité

Thèse présentée et soutenue à Rennes, le 4 décembre 2019
Unité de recherche : IMT Atlantique et Orange Labs
Thèse N° : 2019IMTA0160

Rapporteurs avant soutenance :

Mme Nadia BOUKHATEM	Professeur, Telecom Paris
M. Thierry TURLETTI	Directeur de recherche, INRIA

Composition du Jury :

Président :	M. Jean-Louis ROUGIER	Professeur, Telecom ParisTech
Examinateurs :	Mme Nadia BOUKHATEM	Professeur, Telecom ParisTech
	M. Thierry TURLETTI	Directeur de recherche, INRIA
	M. Arnaud BRAUD	Ingénieur de recherche, Orange Labs
Dir. de thèse :	M. Gwendal SIMON	Professeur, IMT Atlantique
Co-dir. de thèse :	Mme Annie GRAVEY	Professeur, IMT Atlantique

Invités :

M. Stéphane TUFFIN	Ingénieur de recherche, Orange Labs
M. Xavier MARJOU	Ingénieur de recherche, Orange Labs

Remerciements

Je tiens à remercier Monsieur Gwendal Simon, Professeur à l'IMT Atlantique et Madame Annie Gravey, Professeur à l'IMT Atlantique, ainsi que mes encadrants au sein d'Orange Labs : Stéphane Tuffin, Xavier Marjou et Arnaud Braud pour m'avoir encadré, orienté, aidé et conseillé à chaque fois que j'en avais besoin. Qu'ils soient aussi remercié pour leur gentillesse, leur disponibilité permanente et leurs nombreux encouragements qu'ils m'ont prodigués.

J'adresse tous mes remerciements à Nadia Boukhatel, Professeur à Telecom Paris, et Thierry Turletti, Directeur de recherche à l'INRIA, de l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse. Je tiens à remercier Jean-Louis Rougier d'avoir accepté d'être président du jury. Je remercie également tous les membres du jury d'avoir accepté d'assister à la présentation de ce travail.

Je remercie Isabelle Hamchaoui et Alexandre Ferrieux, d'Orange Labs, pour leur aide précieuse qui nous a permis d'améliorer considérablement les résultats de la thèse.

Merci à tous les membres des équipes ARC et OSONS, pour l'agréable atmosphère de travail et l'ambiance cordiale qui a régné pendant ces quelques années. Merci en particulier à Gaël Fromentoux de m'avoir accueilli dans son équipe et à Nathalie Labidurie, Marc Bouillon et Émile Stephan pour le précieux soutien qu'ils m'ont apporté durant ces dernières années.

Je remercie très spécialement mes parents Guylaine et François pour le soutient moral qu'il m'ont apporté en étant toujours là pour moi. Merci. Je remercie aussi les membres de ma famille et mes amis pour leurs encouragements.

Table des matières

Table des figures

Liste des tableaux

Acronymes

ADSL Asymmetric Digital Subscriber Line.

AIMD Additive Increase and Multiplicative Deacrease.

API Application Programming Interface.

BBR Bottleneck Bandwidth and Round-trip propagation time.

BDP Bandwidth Delay Product.

CoDel Controlled Delay.

CPU Central Processing Unit.

CQI Channel Quality Indicator.

diffserv services différenciées.

ECN Explicit Congestion Notification.

FIFO First In First Out.

fq_codel fair queuing controlled delay.

GCC Google Congestion Control.

GSO Generic Segmentation Offload.

HOL Head-of-line.

HTTP HyperText Transfer Protocol.

hystart hybrid start.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IRTF Internet Research Task Force.

LTE Long Term Evolution.

NADA Network-Assisted Dynamic Adaptation.

NAT Network address translation.

OS Operating System.

OSI Open Systems Interconnection.

PCC Performance-oriented Congestion Control.

QoE Quality of Experience.

QoS Quality of Service.

QUIC Quick UDP Internet Connections.

RED Random Early Detection.

RTO Retransmission TimeOut.

RTT Round-Trip Time.

SACK Selective Acknowledgments.

SCTP Stream Control Transmission Protocol.

SFA Session Fairness Assessment.

TCP Transmission Control Protocol.

TFRC TCP Friendly Rate Control.

TLS Transport Layer Security.

TSO TCP Segmentation Offload.

UDP User Datagram Protocol.

UDT UDP-based Data Transfer Protocol.

UE User Equipement.

WSFA Weighted Session Fairness Assessment.

Traduction

Buffer Mémoire tampon.

Checksum Somme des données.

Congestion avoidance Évitement de congestion.

Fast recovery récupération rapide.

flag Drapeau.

Open source Source libre.

Peering Interconnexion.

Proxy Intermédiaire.

Queuing policy Politique de file d'attente.

Slice tranche.

Slow-start Démarrage lent.

Smartphone Téléphone mobile.

Stream flot.

Switch Commutateur.

Timeout Dépassement de délai.

Token Jeton.

User-space Espace utilisateur.

Chapitre 1

Introduction

Contents

Depuis sa création, l'Internet est devenu un composant central des communications dans le monde entier et ne cesse d'évoluer pour améliorer et proposer de nouveaux services aux utilisateurs. Ainsi, des technologies inédites sont apparues sur les réseaux au cours du temps. Ils ont permis l'augmentation considérable des débits avec le déploiement du réseau 4G ou encore la fibre optique jusqu'aux clients finaux. Ces deux exemples ont été deux des dernières évolutions majeures sur le réseau internet. Mais ces transformations continuent à se produire pour améliorer continuellement le réseau internet ; d'autres technologies plus performantes sont attendues dans les années à venir avec la 5G qui sera bientôt en cours de déploiement alors que la communauté de recherche commence déjà à aborder les problématiques pour le réseau 6G [1]. L'augmentation du débit sur le réseau n'est pas la seule évolution, d'autres services qui ne nécessitent pas un débit aussi important sont apparus. C'est particulièrement le cas des services de l'Internet of Things (IoT), qui multiplient de façon draconienne le nombre d'équipements connectés au réseau à faible débit, comme avec l'apparition de capteurs. Cela démontre une diversité importante des usages du réseau internet et nous informe que chaque service a des besoins différents. Ceci se traduit par une complexification de l'écosystème du réseau internet. En effet, les acteurs se multiplient sur le réseau tout en proposant de nouvelles solutions pour améliorer ses performances tout en avivant la concurrence entre eux. Dans ce contexte, la question de l'équité reste un point primordial pour un opérateur.

Dans un premier temps, nous commençons par réaliser une introduction générale permettant de présenter les enjeux principaux de la thèse. Dans un second temps, nous exposons les problématiques identifiées. Enfin, nous présentons les différents chapitres de ce manuscrit.

1.1 Introduction générale

1.1.1 Chaque service a ses propres besoins

L'ensemble des utilisateurs évalue en permanence la qualité des prestations lorsqu'ils utilisent un service sur Internet. Pour optimiser la satisfaction de leurs utilisateurs, les acteurs de l'Internet cherchent à obtenir un débit plus important, une plus faible latence et un taux de perte inexistant. Pour mesurer le contentement des utilisateurs, les professionnels de l'Internet peuvent réaliser des mesures de Quality of Experience (QoE) [2], c'est-à-dire prendre en compte l'ensemble des caractéristiques objectives et subjectives afin de mesurer la satisfaction et la fidélisation des clients.

Néanmoins, l'ensemble des acteurs de l'Internet n'a pas le même objectif en ce qui concerne la gestion de la QoE sur le réseau : certains acteurs comme les fournisseurs de services vont essayer de maximiser la QoE de leurs utilisateurs sans se soucier de leurs concurrents tandis que d'autres comme les opérateurs réseau cherchent à maximiser la QoE de chaque client, indépendamment des services.

Dans le but de maximiser la QoE, les différents services ont besoin d'obtenir des caractéristiques réseau précises afin de fonctionner et ainsi obtenir une expérience optimale pour les utilisateurs. Les différents acteurs de l'Internet ont la possibilité de prioriser certains flux, de façon à répartir au mieux les ressources réseau pour optimiser le transport des données de chaque flux. Ce concept est appelé la Quality of Service (QoS). Ainsi les politiques de QoS appliquées sur le réseau par les différents acteurs de l'Internet permettent d'offrir aux utilisateurs des débits et des temps de réponse différenciés par application. Par exemple, si nous considérons le visionnage d'une vidéo en streaming, la QoE peut être directement affectée par un des évènements suivants : le temps de démarrage de la vidéo, la qualité de la vidéo reçue et aussi la présence de phases de *re-buffering* [3] (c'est-à-dire une interruption de la vidéo pendant son visionnage suite à la diminution du débit). Pour éviter ces situations, les acteurs de l'Internet ont la possibilité de pouvoir gérer les flux vidéos de façon différenciée dans le but de maximiser la QoE des utilisateurs. En effet, les acteurs de l'Internet cherchent à diminuer la latence, la gigue et le taux d'erreur lors de la transmission de la vidéo sur le réseau [4]. En ce qui concerne le téléchargement des pages web, c'est le temps d'affichage qui sera essentiel pour la satisfaction des clients. Les pages web ne représentent pas forcément de grande quantité de données à transmettre mais elles doivent être réalisées rapidement. Du point de vue de la QoS, la latence sera un critère important.

Ces exemples ne sont que quelques services proposés aux utilisateurs. Aujourd'hui, ils se diversifient de plus en plus sur le réseau et l'exigence des utilisateurs s'accroît, en particulier avec l'arrivée du réseau mobile 5G qui identifie trois axes majeurs pour fournir la QoS à l'ensemble des clients : une faible latence, un débit important et un nombre très important d'appareils à se connecter au réseau avec l'IoT [5].

Durant cette thèse, nous nous concentrons sur un aspect de la QoS, à savoir le débit de l'utilisateur. Pour cela, nous appliquons notre réflexion sur l'équité réseau qui nous permet de pouvoir mesurer la répartition du débit en fonction des demandes des utilisateurs. Nous prendrons l'exemple du téléchargement de contenus, qui sera le fil rouge de cette thèse.

1.1.2 Évolution des usages en terme de débit

Les réseaux mobiles ont évolué au fil du temps afin d'améliorer la qualité de l'expérience des utilisateurs, notamment en terme de débit. Les applications à venir, par exemple la vidéo 4K, la réalité virtuelle, et bien d'autres, nécessitent des débits nettement supérieurs à ceux proposés aujourd'hui. Pour cela, les futures générations de réseaux mobiles s'engagent à fournir des débits plus élevés. En effet, la 5G vise des débits dix fois supérieurs à ceux de la 4G et par conséquent entraînera une amélioration de la QoE des utilisateurs [6].

Les opérateurs s'engagent à réaliser des investissements très importants afin de pouvoir proposer une nouvelle infrastructure de réseau mobile, nommée la 5G. Pour proposer ces nouveaux services aux utilisateurs, ils promettent le déploiement de nouveaux équipements réseaux sur l'ensemble du territoire. Mais la performance de leurs réseaux ne dépend pas uniquement des infrastructures réseau déployées. Entre autres, les protocoles de transport réseau ont un impact significatif sur le débit des utilisateurs, car ce sont ces protocoles qui définissent le débit des flux de chaque service sur le réseau. Aujourd'hui avec les nouveaux protocoles qui sont déployés, les flux parviennent généralement à obtenir l'intégralité de la bande passante disponible sur le réseau. Mais le problème de pouvoir vérifier que les ressources réseau, dans notre cas la bande passante, sont partagées correctement entre les divers flux et donc sur les différents services reste aujourd'hui une question ouverte.

Aujourd'hui, la bande passante ne peut pas être divisée en parts égales entre les différents utilisateurs. Effectivement, nous constatons un accroissement de l'écart des débits nécessaires pour faire fonctionner les différents services sur le réseau. En effet, les débits peuvent aller de quelques kilooctets (ko) pour l'écoute d'une musique en ligne à plusieurs mégaoctets (Mo) pour le visionnage d'une vidéo en 4K. Ainsi différents types de services peuvent cohabiter sur le réseau internet tout en ayant des exigences complètement différentes. Divers services peuvent alors se retrouver en concurrence sur un même point de saturation. Deux cas peuvent alors être distinguées :

— **Plusieurs utilisateurs sont en concurrence sur un même point de saturation :** Le premier cas correspond à la situation où plusieurs utilisateurs créent une congestion sur un même point de saturation. Nous pouvons prendre l'exemple de deux utilisateurs visionnant une vidéo qui partagent une bande passante insuffisante par rapport à leurs demandes (création de congestion dans le réseau).

— **Un utilisateur a plusieurs flux en concurrence sur un même point de saturation :** Le deuxième cas correspond à un utilisateur qui utilise plusieurs services simultanément et donc plusieurs flux. Or ces différents flux peuvent avoir des demandes supérieures à la bande passante disponible. Ils se retrouvent donc en concurrence sur un même goulot d'étranglement. Par exemple lorsque l'on écoute de la musique en arrière plan avec une application de musique en continu et que l'on navigue sur Internet, ou lors du téléchargement d'un contenu pendant qu'on visionne une vidéo ou encore un appel téléphonique alors qu'on navigue sur le web.

1.1.3 La congestion réseau

La congestion réseau correspond aux points de saturation du réseau, c'est-à-dire qu'un équipement réseau reçoit plus de données à transmettre qu'il ne peut le faire. Lorsque la congestion est trop importante, des pertes de paquets peuvent arriver et les données de l'utilisateur seront perdues, charge à l'émetteur de les retransmettre. Ces pertes de paquets peuvent alors avoir un impact significatif sur la QoE de l'utilisateur.

Ces points de saturation peuvent se situer à différents emplacements sur le réseau :

— *Réseau d'accès fixe :* Que ce soit les particuliers ou les professionnels, ils peuvent avoir un point de congestion dans les équipements réseau qu'ils possèdent. En effet, aujourd'hui les débits dans les réseaux fixes se sont grandement améliorés pour atteindre jusqu'à 10Gbit/s avec une seule connexion en fibre optique. Généralement, ces améliorations ont déplacé les points de saturation du réseau d'accès de l'opérateur à l'équipement des utilisateurs. Lorsque l'usager est connecté sur un réseau WiFi, le point de saturation est déplacé à l'interface fibre/WiFi. En effet, le réseau WiFi actuel peut supporter au maximum 54Mbit/s. Les différentes technologies déployées chez un particulier ou par un administrateur réseau dans son entreprise peuvent avoir un impact direct sur le débit d'un utilisateur ou d'une application et donc une influence sur l'équité.

— *Réseau d'accès mobile :* Le déploiement des équipements au niveau des réseaux d'accès est particulièrement coûteux pour les opérateurs et nécessite des investissements très importants. C'est pourquoi les réseaux sont dimensionnés pour les besoins des utilisateurs. Si une demande ponctuelle est plus importante alors une congestion se créera dans le réseau.

— *Points de peering :* Les points de *peering* sont les points d'interconnexion entre différents acteurs de l'Internet (opérateurs, fournisseurs de services...). Pour que de telles connexions existent, des accords commerciaux sont trouvés entre ces différents acteurs ; même si elles sont peu coûteuses, les débits y sont plafonnés. Si trop de données sont à transmettre alors un point de congestion peut se créer dans le réseau.

Dans cette thèse nous nous sommes uniquement concentré sur les deux points de saturation suivants : le réseau d'accès mobile (lorsque plusieurs flux d'un même utilisateur sont en concurrence entre-eux) et fixe (lorsque plusieurs utilisateurs d'une même point d'accès sont en concurrence entre-eux). Il est important de noter dès maintenant que l'équité n'est pas gérée de la même façon dans les deux types de réseau d'accès. Sur les réseaux d'accès fixe (Ethernet et WiFi) les *queuing policies* sont des First In First Out (FIFO), c'est-à-dire que les paquets des flux sont traités par ordre d'arrivée. En cas de congestion, aucun des flux n'est priorisé par rapport aux autres. Tandis que sur les réseaux mobiles, ce sont typiquement des algorithmes de *proportional fairness* qui sont utilisés (voir section 2.3.2). En ce qui concerne l'équité au niveau des points de *peering*, elle n'a pas été traitée dans cette thèse et elle nécessiterait de nouvelles mesures dans des conditions complètement différentes de celles réalisées dans cette thèse.

1.1.4 Les protocoles de transport

Depuis l'essor d'Internet, les chercheurs ont mis au point des protocoles et des algorithmes pour accroître l'efficacité globale du réseau. Les protocoles de transport occupent une place très importante dans ces évolutions, car généralement ce sont eux qui implémentent les algorithmes de contrôle de congestion qui définissent le débit des utilisateurs.

L'évolution des besoins des utilisateurs (par exemple le streaming vidéo HD/UHD, la réalité virtuelle, etc.) ainsi que l'augmentation du trafic mondial ont conduit à la conception de nouveaux protocoles de transport tels que Quick UDP Internet Connections (QUIC) [7, 8], qui vise à remplacer à la fois le protocole Transmission Control Protocol (TCP) et Transport Layer Security (TLS) pour assurer le transport du protocole applicatif comme HyperText Transfer Protocol (HTTP). QUIC est déjà largement déployé sur Internet, notamment par Google, et représente actuellement environ 7% du trafic internet mondial [7]. Actuellement l'Internet Engineering Task Force (IETF) l'étudie pour le normaliser. Les principaux atouts de QUIC [8] sont les faibles délais de l'établissement d'une connexion (appelée "0-RTT") et le chiffrement au niveau de la couche transport pour améliorer la sécurité du transport des données des utilisateurs ainsi que pour lutter contre l'ossification du protocole causée par des intermédiaires.

La coexistence de plusieurs flux QUIC et TCP soulève la question classique de "l'équité". Cette question est encore plus importante lorsque l'on considère que les versions de QUIC peuvent être multiples tout en ayant des cycles de mise à jour rapides ; c'est parce que QUIC est directement implémenté dans les applications utilisateurs (par exemple Google Chrome,

LiteSpeed Web Server) qu'il bénéficie de modifications plus rapides et plus simples que TCP qui est implémenté dans le noyau du système d'exploitation [7]. Le nombre de fournisseurs de contenus utilisant des fonctions et des paramètres de transport personnalisés peut augmenter considérablement. Par exemple, Google a sa propre implémentation côté serveur, ce qui nécessite qu'une autre implémentation de QUIC soit calibrée en fonction de ses caractéristiques pour obtenir des résultats similaires [9]. Les petits fournisseurs de contenus ont les mêmes possibilités avec QUIC mais pas nécessairement les mêmes capacités que Google pour réaliser l'optimisation du protocole. Ceci nous fait penser que de nombreuses versions de QUIC peuvent coexister dans le réseau. La gestion et le contrôle de l'équité deviennent alors plus complexes. Cela entrave les efforts déployés par les exploitants de réseaux pour faire respecter l'équité collective entre les flux simultanés.

De plus, avec le nouveau protocole QUIC, il est important de noter que l'algorithme de contrôle de congestion se situe au niveau de l'application (Google Chrome, serveur web) et non plus dans le système d'exploitation du serveur. Cela facilite la diversification des algorithmes de contrôle de congestion sur Internet et permet leur mise à jour en continu contrairement aux algorithmes de contrôle de congestion comme sur TCP, qui sont implémentés dans le noyau du système d'exploitation à la fois du côté client et serveur. Par conséquent, la mise à jour des algorithmes de contrôle de congestion basés sur le noyau comme TCP-CUBIC et TCP-RENO nécessitent la mise à jour d'un nombre important de périphériques tandis qu'avec QUIC une simple mise-à-jour de l'application permet de modifier l'algorithme de contrôle de congestion.

En outre, ce constat est plus alarmant du fait que de nouveaux algorithmes de contrôle de congestion sont actuellement à l'étude ou en cours de déploiement. En effet, les algorithmes de contrôle de congestion les plus populaires, à savoir RENO [10] et CUBIC [11], identifient les congestions au moyen de la détection de pertes de paquets. Ces algorithmes permettent d'ajuster le débit de données parmi les flux réseau simultanés ; cependant, les algorithmes de contrôle de congestion basés sur la perte de paquets ne répondent pas aux exigences de latence des services réseau entrants telles que les applications en temps réel, les applications à faible latence sur le réseau mobile 5G, la réalité virtuelle, etc. Les récents efforts de recherche ont permis d'obtenir de nouveaux algorithmes, en particulier Bottleneck Bandwidth and Round-trip propagation time (BBR) [12, 13] et Performance-oriented Congestion Control (PCC) [14]. Ces deux propositions visent à améliorer la performance du réseau en maximisant à la fois le débit du réseau et la satisfaction des utilisateurs. BBR et PCC diffèrent de RENO et CUBIC car ils ne comptent plus sur la détection des pertes. L'algorithme BBR est basé sur l'évolution de la latence tandis que le PCC s'appuie sur la prédiction du débit. De plus, ces deux algorithmes ne se fondent plus sur le principe Additive Increase and Multiplicative Deacrease (AIMD), qui permettait d'assurer l'équité sur le réseau. Par conséquence, l'équité entre les algorithmes de contrôle de congestion devient alors moins évidente sur le réseau.

L'ensemble de ces évolutions est accompagné d'un chiffrement généralisé des données jusqu'aux protocoles de transport. Ceci rend alors complètement opaque la vision du trafic sur le réseau et les mesures d'équité sont alors très compliquées à réaliser. En effet, nous nous trouvons sur un réseau, où très peu d'informations circulent pour caractériser les services des différents flux afin de calculer l'équité réseau. Nous pouvons alors nous demander, avec l'ensemble des évolutions qui ont lieu sur Internet, si nous nous dirigeons vers la loi du plus fort, ou si nous voulons encore réaliser l'équité entre les différents services ?

1.1.5 Et la neutralité du net ?

Aujourd'hui les opérateurs réseau sont contraints à la net neutralité pour des raisons complètement légitimes. En effet, ils n'ont le droit en aucune manière de prioriser une entreprise de l'Internet par rapport à une autre. Mais l'évolution des réseaux, comme détaillée précédemment, nous démontre que l'équité des transports de données est bien une question d'actualité. En effet, des inéquités peuvent être crées directement au sein des entreprises les plus puissantes de l'Internet au détriment des autres. Nous pouvons alors nous demander si cette réglementation ne serait pas transposable aux fournisseurs de services ?

1.2 Problématiques

Différents sujets sont abordés durant cette thèse, et apportent des éléments de réponses aux questions suivantes :

— Comment améliorer la mesure de l'équité sur le réseau ?

— Le positionnement des protocoles migre de l'Operating System (OS) vers la partie applicative : cela se traduit par le fait que chaque fournisseur pourra facilement personnaliser ses propres protocoles de transport dans ses applications. Nous dirigeons-nous vers de l'équité ou une anarchie où la politique du plus fort empêche les autres services d'exister ?

— Quelle est la limite entre améliorer les performances ou respecter l'équité sur le réseau ?

1.3 Organisation de la thèse

Ce manuscrit de thèse débute tout d'abord par la présentation de l'état de l'art de l'ensemble des sujets abordés durant cette thèse. Il commence par présenter le fonctionnement de deux des protocoles de transports les plus utilisés aujourd'hui sur les réseaux : User Datagram Protocol (UDP) et Transmission Control Protocol (TCP). Ceux-ci montrent leurs limites et sont difficiles à mettre à jour sur les équipements réseaux. C'est pourquoi Google en a présenté un nouveau, nommé QUIC, qui est exposé dans la suite de l'état de l'art. Par la suite, nous

abordons plus dans le détail les protocoles de transport pour présenter certaines évolutions des algorithmes de contrôle de congestion. Ils composent généralement une partie du protocole de transport qui définit le débit des flux pour éviter les congestions sur le réseau. Par la suite, nous abordons la QoS en introduisant les différentes classes de service et l'équité sur le réseau. Enfin, nous terminons cet état de l'art par les mesures d'équité et de performance réalisées sur les protocoles de transports de congestion et les algorithmes de contrôle de congestion.

Le second chapitre de cette thèse est consacré à la présentation d'une plateforme de mesures rendant possible l'évaluation de l'équité. Cette plateforme permet de mettre en concurrence plusieurs flux, sur un réseau à débit constant ou variable, pour observer l'évolution de chacun d'entre eux ainsi que le comportement de ces derniers dans le *buffer* du point de saturation. Enfin, nous présentons l'ensemble des mesures réalisables à partir de cette plateforme.

Le troisième chapitre présente une métrique permettant de réaliser des mesures d'équité durant la session d'un flux. Pour cela, nous commençons par justifier les choix d'un tel type de métrique. Puis nous présentons une première métrique, nommée Session Fairness Assessment (SFA), qui considère l'ensemble des moments de la session de la même manière. Enfin dans la dernière partie, nous introduisons une deuxième métrique, nommée Weighted Session Fairness Assessment (WSFA) permettant de considérer les moments de la session de différentes manières. Dans notre exemple, nous considérons que les premiers instants de la session sont plus importants que les derniers.

Le quatrième chapitre commence par une analyse des implémentations de l'algorithme de contrôle de congestion CUBIC dans les protocoles de transport TCP et QUIC. Nous mettons en évidence des modifications entre les différentes implémentations qui peuvent impacter l'équité. Après avoir exposé les scénarios de tests, nous avons déterminé trois paramètres des protocoles de transport qui ont un impact sur l'équité : l'option hybrid start (hystart), le nombre de connexions TCP émulées dans QUIC et la limitation de la taille de la fenêtre de congestion dans QUIC. Les mesures d'équité ont été réalisées sur des réseaux à débit constant de type Asymmetric Digital Subscriber Line (ADSL) et des réseaux à débit variable de type réseau 4G. Puis nos appliquons la méthode s'appuyant sur notre plateforme et la métrique SFA pour déterminer l'équité entre les flux.

Le cinquième et dernier chapitre montre une évaluation de l'équité des algorithmes de contrôle de congestion, après une présentation des nouveaux scénarios de tests. Cette évaluation de l'équité est réalisée sur des réseaux à débit constant de type ADSL et elle analyse l'équité en fonction des deux métriques présentées dans le chapitre 3 (SFA et WSFA). Enfin nous déterminons le nombre de situations réseau nécessaire pour obtenir une valeur d'équité représentative de la réalité.

Chapitre 2

Protocoles, Algorithmes de Contrôle de Congestion et Équité : État de l'art

Contents

L'ensemble des acteurs de l'Internet cherche en permanence à améliorer les performances de l'Internet. Généralement, cela peut se traduire de différentes manières : augmenter le débit, accélérer la transmission des données en diminuant les temps de latence... Afin d'y parvenir, les protocoles de l'Internet évoluent continuellement pour toujours mieux satisfaire les utilisateurs. Mais d'ordinaire, ces changements sont longs à être déployés, car une adaptation du réseau ou une mise à jour des équipements réseau est nécessaire. Généralement, nous observons une période de cohabitation entre anciens et nouveaux protocoles. C'est dans ce laps de temps que la question de l'équité doit être omniprésente.

Dans ce chapitre, nous traitons l'ensemble de l'état de l'art sur les évolutions majeures des protocoles de transport, des algorithmes de contrôle de congestion et de l'équité réseau. Pour ce faire, la première partie de l'état de l'art traite la dernière évolution majeure des protocoles de transport, avec la création du protocole Quick UDP Internet Connections (QUIC) ayant pour but d'améliorer le protocole historique Transmission Control Protocol (TCP). Puis dans un second temps, nous nous concentrons sur l'évolution des algorithmes de contrôle de congestion ; en particulier depuis l'affirmation par Google en 2015, qu'un changement majeur des algo-

rithmes de contrôle de congestion doit être réalisé pour améliorer les performances du réseau. En effet, il préconise un changement radical sur les algorithmes de référence comme la fin de la détection de la congestion par les pertes et la fin du principe Additive Increase and Multiplicative Deacrease (AIMD). Enfin, la dernière partie est consacrée à l'équité entre les différents algorithmes de contrôle de congestion et protocoles de transport.

2.1 Évolution des protocoles de transport : de TCP à QUIC

2.1.1 Protocoles de transport

Pour acheminer les données de diverses applications (par exemple : pages web, téléchargement de contenus), d'une machine émettrice vers une machine réceptrice par un chemin réseau (ensemble de machines traversées par les données), plusieurs niveaux de protocoles sont nécessaires. Chacun de ces niveaux représente une fonction précise sur le réseau. Ils sont au nombre de sept et sont différenciés dans le modèle Open Systems Interconnection (OSI) comme suit : Physique, Liaison de données, Réseau, Transport, Session, Présentation et Application. Le niveau 4, qui gère le transport des données entre une machine émettrice et une machine réceptrice, est le point de départ de cet état de l'art. L'ensemble des protocoles du niveau 4 sert d'intermédiaire entre les protocoles des couches supérieures et ceux de la couche réseau. Par exemple, lors du téléchargement d'une page web, c'est généralement le protocole de transport TCP [15] qui est utilisé. Ce dernier reçoit les données des protocoles de session (par exemple Transport Layer Security (TLS) [16] pour des connexions chiffrées) ou des protocoles applicatifs (par exemple HyperText Transfer Protocol (HTTP) [17, 18] pour des connexions non chiffrées). Une fois les données arrivées au niveau du protocole de transport, elles seront enrichies avec l'ajout d'un en-tête (permettant un échange d'informations entre l'émetteur et le récepteur pour s'assurer de la bonne transmission des données). Ensuite elles sont transmises à la couche réseau qui définit la source et la destination des données.

Historiquement, User Datagram Protocol (UDP) [19] et TCP [15] sont deux des protocoles de transport les plus utilisés qui existent pour acheminer les données à travers un réseau. Ces protocoles de transport sont directement implémentés dans le système d'exploitation des machines. Ainsi, ils assurent aussi le lien entre la partie "*user-space*" et le noyau de la machine.

Le premier, UDP, permet de réaliser des échanges de données de façon très simple. Il rend possible l'envoi des données en mode non connecté, c'est-à-dire qu'il n'y a aucun établissement de connexion pour transmettre les données, aucune retransmission lors de la perte d'un paquet, ou encore une vérification d'intégrité facultative des données sur IPv4. Ce mode de transport, non fiable, est souvent utilisé pour les connexions en temps réel ou le visionnage de vidéos en streaming. Ces types de services ne nécessitent généralement pas de retransmissions, car elles arriveraient avec trop de retard pour l'utilisateur. En effet, si l'utilisateur regarde une vidéo et qu'un paquet de données est perdu, la retransmission lui parviendra une fois que

la scène qu'il visionnait est terminée. Le second protocole, TCP est lui utilisé pour les communications fiables. Son fonctionnement est alors plus complexe pour garantir l'intégrité des données, éviter la congestion dans le réseau et la retransmission des données perdues.

2.1.2 Protocole de transport fiable : TCP

Le protocole TCP [15] est un protocole de transport fiable, en mode connecté. Aujourd'hui, il est largement répandu sur les réseaux internet. Il est principalement utilisé pour des communications de type web, de téléchargement de contenus et pour toute application nécessitant un transport fiable des données. Ce protocole imaginé en 1981 a évolué au fil du temps ; beaucoup d'options ont été créées afin d'améliorer ses performances. Néanmoins, ses fonctionnalités principales ont subsisté et se résument en cinq parties :

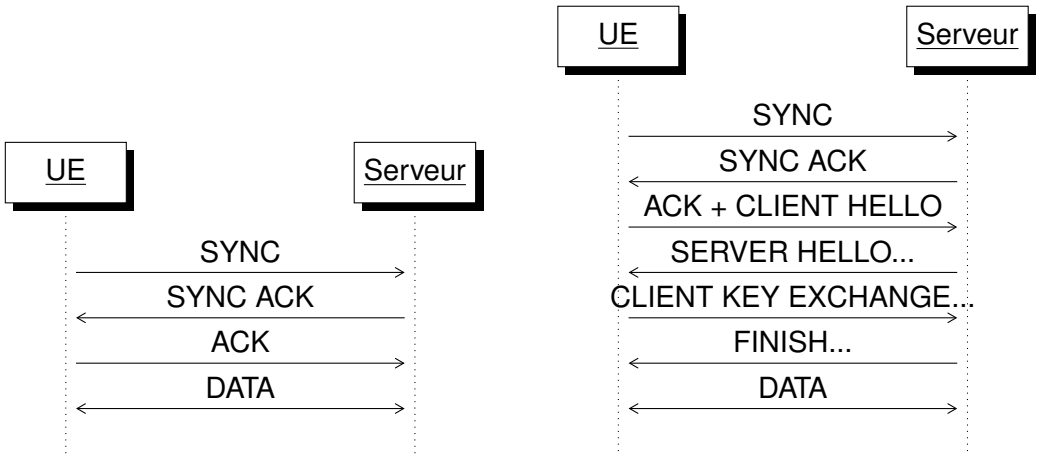
— *Établissement de la connexion :* L'établissement de la connexion permet d'établir un flux TCP entre la machine émettrice et la machine réceptrice (voir figure 2.1a). Cette connexion se déroule en trois étapes : *(i)* l'ouverture de la connexion de l'User Equipement (UE) vers le serveur (paquet *"SYNC"*), *(ii)* l'acquittement d'ouverture de la connexion du serveur vers l'UE reconnaissable avec le paquet *"SYNC, ACK"* et enfin *(iii)* l'acquittement final d'ouverture de l'UE vers le serveur (paquet *"ACK"*). Ces trois paquets permettent aussi la négociation des différents paramètres et options du flux TCP.

— *Séquencement des données :* Le séquencement des données permet de numéroter les paquets qui circulent sur le réseau et d'indiquer au récepteur l'ordre des paquets reçus pour pouvoir rassembler les données. En effet, lors de la transmission des données, les différents paquets peuvent parcourir différents chemins réseau et arriver dans un ordre différent de celui de l'émission.

— *Checksum :* Le *checksum* permet la vérification de l'intégrité des données transmises. Pour cela, un premier calcul est réalisé au niveau de l'émetteur, qui est ensuite communiqué dans l'en-tête TCP. Le récepteur peut alors refaire le même calcul pour le comparer à celui de l'émetteur. Si ces deux calculs sont identiques alors il n'y a eu aucune altération des données lors du transport.

— *Acquittement des données :* L'acquittement des données permet d'indiquer au récepteur la bonne réception des données. Cet acquittement est envoyé après la vérification de l'intégrité des données. En cas de problème de transmission, le récepteur le signale par la transmission de 3 paquets *"DUP ACK"*.

— *Algorithmes de contrôle de congestion :* Les algorithmes de contrôle de congestion sont un mécanisme permettant la régularisation des débits sur un réseau. Ces mécanismes évitent des congestions sur le réseau.



(a) TCP. (b) TCP et TLS.

FIGURE 2.1 – Établissement des connexions avec les protocoles TCP et TCP avec TLS.

2.1.3 Limitation du protocole de transport : TCP

Dans [7], les auteurs illustrent les problèmes actuels du protocole TCP, comme : la durée d'établissement des connexions, l'absence de multiplexage des connexions du niveau supérieur qui dégradent les performances du réseau, le manque d'adaptabilité aux attaques et la complexité de mise à jour.

Durée d'établissement de la connexion

Les figures 2.1a et 2.1b représentent les différentes étapes pour l'établissement d'une connexion avec les protocoles TCP et TLS. Lors de l'établissement d'une connexion TCP, trois messages ont besoin d'être échangés entre l'émetteur et le récepteur (*"SYNC"*, *"SYNC ACK"* et *"ACK"*). La durée d'établissement d'une connexion TCP est alors de $1,5$ RTT. Le Round-Trip Time (RTT) est le temps d'aller-retour (envoi et acquittement des données) entre l'émetteur et le récepteur. De plus, avec le chiffrement qui se généralise sur l'internet, la sécurisation des données des utilisateurs et donc du protocole TCP s'impose. La sécurisation du protocole TCP est réalisé avec le protocole TLS (couche session). Ce dernier nécessite lui aussi une phase de connexion d'une durée de $1,5$ RTT (voir figure 2.1b). Il faut alors 3 RTT pour établir une connexion sécurisée, et pouvoir émettre les premiers octets de données chiffrés. Ce temps d'établissement de connexion reste relativement long et augmente la durée de réception des données pour un utilisateur.

Multiplexage des connexions

Lors du téléchargement d'une page web, le nombre de requêtes HTTP peut être important pour télécharger l'intégralité des données dont l'utilisateur a besoin. Par exemple, plusieurs requêtes HTTP sont nécessaires pour télécharger l'intégralité des fichiers requis et fournir le contenu de la page web d'un utilisateur. Mais le protocole TCP ne permet pas directement de réaliser un multiplexage de requêtes HTTP. Il oblige alors le navigateur web à les exécuter les unes à la suite des autres (sauf avec HTTP/2 qui permet un multiplexage au niveau applicatif). Avec l'utilisation du protocole TCP, le Head-of-line (HOL) blocking se produit en cas de perte de paquets lorsque le destinataire attend qu'un paquet manquant soit livré pour acquitter les paquets reçus avant une perte. Pour pallier ce problème, les différents navigateurs comme Firefox, Google Chrome et d'autres réalisent l'établissement de plusieurs flux TCP en parallèle avec un même serveur. Par exemple, Firefox et Google Chrome utilisent au maximum 6 connexions TCP simultanément. Cette solution permet de multiplexer les différentes requêtes HTTP sur l'ensemble des flux TCP. Mais cette solution admet pour inconvénient de multiplier les délais d'ouverture de connexion.

Mise à jour et adaptabilité aux attaques

Le protocole TCP est aujourd'hui implémenté dans le système d'exploitation des machines. Pour toute modification apportée au protocole, celle-ci nécessite la mise à jour de l'ensemble des systèmes d'exploitation qui l'utilisent. Ces mises à jour sont complexes, en particulier sur les Operating System (OS) des serveurs, et deviennent naturellement beaucoup plus rares que les mises à jour d'applications. Ainsi, le protocole TCP reste figé et n'évolue que très lentement au cours du temps. De plus, lorsqu'une attaque est découverte sur ce dernier, les mises à jour sont alors très longues à être déployées à travers le réseau.

Bilan des limitations du protocole de transport TCP

Dans cette sous-section, il a été démontré que le protocole de transport admet des limites dans plusieurs domaines : que ce soit sur la durée d'établissement d'une connexion, le multiplexage des connexions, ou encore la mise à jour lente de ce protocole. C'est pourquoi pour répondre à ce problème, un nouveau protocole a été proposé par Google : Quick UDP Internet Connections (QUIC). Il a pour but d'améliorer les performances des applications tout en augmentant la sécurité avec le chiffrement généralisé.

2.1.4 Nouveau protocole : QUIC

Le protocole QUIC [7, 8, 20] est un nouveau protocole de transport en cours de normalisation à l'Internet Engineering Task Force (IETF). Ce nouveau protocole bouleverse la hiérarchie du modèle OSI et propose une solution qui regroupe à la fois un protocole de transport fiable

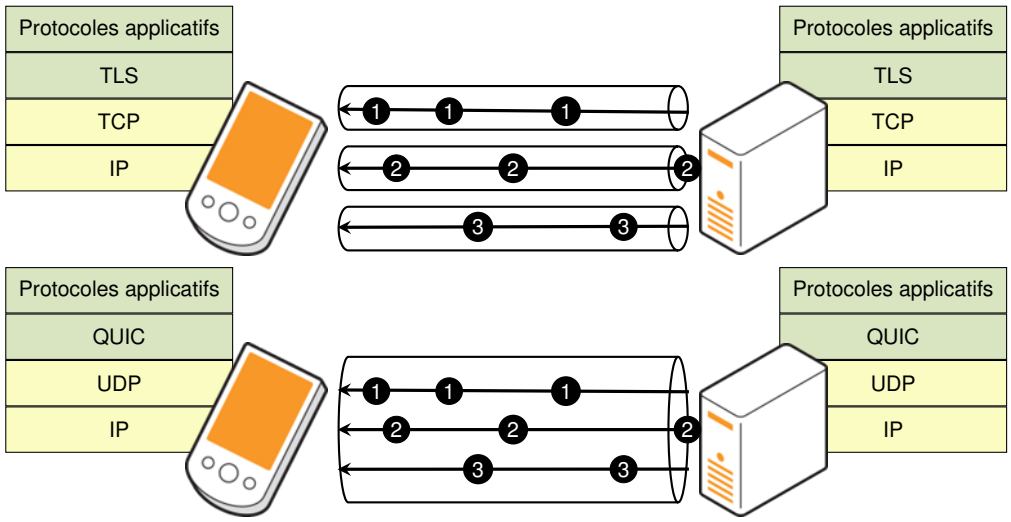
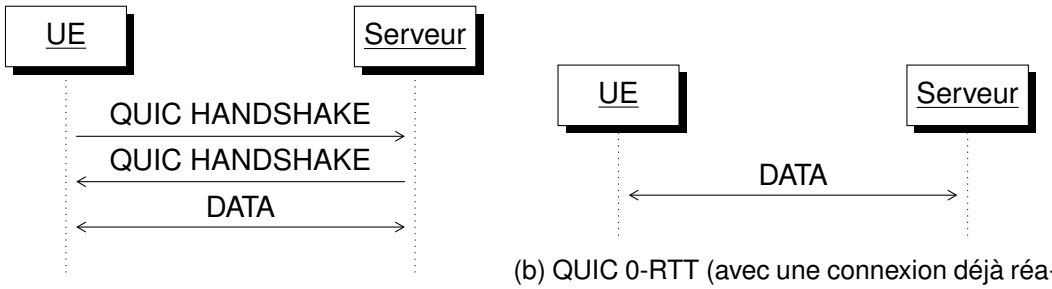


FIGURE 2.2 – Évolution des algorithmes de contrôle de congestion.

et connecté (niveau 4 du modèle OSI) mais aussi un protocole de sécurité (niveau 5 du modèle OSI). Afin de pouvoir faciliter le déploiement du nouveau protocole sur le réseau il est implémenté sur un protocole de transport déjà existant : UDP. Ainsi, en se basant sur un protocole déjà existant, les traversées de firewall ou de Network address translation (NAT) et autres équipements de sécurité sont facilitées du faîte qu'aucune configuration supplémentaire n'y est nécessaire. Aujourd'hui, il est déjà déployé sur le réseau internet par Google. En effet, lors de l'utilisation des applications de Google (par exemple Google Chrome) et sur les serveurs de Google c'est ce protocole qui est utilisé. En 2019, il est déjà implémenté sur $3,2\%$ des sites internet [21] et représente 7% du trafic internet [7].

Le protocole QUIC implémente une grande partie des fonctionnalités et option de TCP, c'est-à-dire les fonctions de base de TCP comme l'établissement d'une connexion, l'acquittement et l'intégrité des données, la gestion de la congestion ainsi que le séquencement des données. Des améliorations y sont apportées avec l'établissement d'une connexion plus rapide (voir figure 2.3). En effet, dès lors que l'application et le serveur ont déjà échangé des données, alors les nouvelles connexions peuvent être établies directement sur le réseau et commencer immédiatement l'émission des données. Contrairement à TCP et TLS qui étaient obligé d'attendre 3 RTT pour commencer l'émission des données sécurisée sur le réseau.

Un autre changement majeur est le multiplexage des données. Pour cela, le protocole QUIC réutilise les fonctions de transport de l'état de l'art. En effet, de la même manière que Stream Control Transmission Protocol (SCTP) [22], QUIC est capable de multiplexer plusieurs sessions applicatives (par exemple HTTP) sur une seule connexion de transport alors que TCP utilise plutôt plusieurs connexions pour éviter le *HOL blocking* ; comme illustré dans Figure 2.2. Chaque session QUIC multiplexe plusieurs *stream*s QUIC qui regroupent chacune une ou plusieurs requêtes applicatives (HTTP). Mais ce multiplexage de données peut introduire des pro-



(a) QUIC (première connexion).

(b) QUIC 0-RTT (avec une connexion déjà réalisée avec le serveur).

FIGURE 2.3 – Établissement des connexions avec le protocole QUIC.

blèmes d'équité. Par exemple, un cas fréquent est la concurrence entre plusieurs connexions TCP avec une seule connexion QUIC qui transporte plusieurs sessions. Ces deux méthodes peuvent être utilisées simultanément pour télécharger des ressources de pages web depuis différents domaines (supportant différents protocoles). La pratique dans les navigateurs web est soit d'ouvrir plusieurs connexions TCP ou soit plusieurs *stream*s QUIC dans une seule connexion QUIC pour chaque domaine (selon le protocole implémenté sur chaque domaine). Or, le contrôle de la congestion se fait au niveau de la connexion QUIC et non au niveau des *stream*s. Donc, une application utilisant N connexions TCP pourrait théoriquement atteindre N fois le débit QUIC. Dans plusieurs implémentations QUIC (c'est-à-dire QUIC-GO [23], Chromium [24] et QUIC Google [25]), le nombre N de connexions TCP émulées a été introduit dans l'algorithme de contrôle de congestion CUBIC pour rendre la concurrence équitable. Cette évolution est réalisée en modifiant l'évolution de la taille de la fenêtre de congestion qui définit le nombre de paquets que l'expéditeur est autorisé à envoyer sans accuser de réception (la taille de la fenêtre de congestion est proportionnelle au débit d'une connexion). Une étude plus approfondie de l'évolution de la taille de la fenêtre de congestion sera présentée dans la partie 5.1 de cette thèse.

D'autres différences existent entre le protocole QUIC et TCP. Par exemple, pour augmenter la sécurité du transport des données utilisateurs, QUIC chiffre l'ensemble de ces données et entêtes circulant sur le réseau. Pour cela, il se base sur le protocole TLS qui est réimplémenté dans QUIC.

Enfin, une autre différence majeure entre le protocole QUIC et TCP est à noter : le positionnement de QUIC dans l'architecture logicielle de l'émetteur et du récepteur. Ce changement de positionnement a été effectué pour pallier la lenteur des mises à jour de TCP. Aujourd'hui, le protocole QUIC n'est plus implémenté dans le système d'exploitation, mais directement dans les applications (par exemple dans Google Chrome). Cette modification d'emplacement permet donc un cycle plus rapide de mise à jour et une plus grande flexibilité. En d'autres termes, le nombre de fournisseurs de contenus utilisant des fonctions et des paramètres de transport personnalisés augmente considérablement. Par exemple, nous savons que Google a sa propre implémentation sur ses serveurs, car pour obtenir des performances similaires, Kakhki et al. ont

réalisé une calibration d'une autre implémentation de QUIC pour obtenir des caractéristiques similaires aux implémentations de Google [9]. Les petits fournisseurs de contenus ont la même possibilité technique avec QUIC, mais pas nécessairement les mêmes capacités d'implémentation que Google pour optimiser leur implémentation en fonction des services réalisés. Ainsi de nombreuses versions de QUIC, peuvent coexister dans le réseau, cela rend la gestion et le contrôle de l'équité plus complexes et entrave les efforts déployés par les opérateurs réseau pour faire respecter l'équité collective entre les flux simultanés.

2.2 Évolution des algorithmes de contrôle de congestion

Conjointement aux protocoles de transport, les algorithmes de contrôle de congestion évoluent en permanence pour améliorer les performances du réseau et s'adapter à ses évolutions. Aujourd'hui, un très grand nombre d'algorithmes de contrôle de congestion existe (par exemple le nombre d'algorithmes de contrôle de congestion implémentés par défaut dans Linux 3.13 est au nombre de 6), mais aucun algorithme de contrôle de congestion n'est parfait dans l'ensemble des situations réseau possibles. La suite de cette section retrace l'évolution d'une partie des algorithmes de contrôle de congestion (voir figure 2.4) qui ont soit été très largement déployés ou en cours de déploiement sur le réseau internet soit encore en discussion à l'IETF pour être normalisés. Les algorithmes présentés sont tous implémentés et peuvent être déployés sur Internet ou sur des plateformes de laboratoire.

Les algorithmes de contrôle de congestion permettent de réguler le débit d'un flux (par exemple : TCP ou QUIC) afin d'éviter les congestions dans un réseau. Une congestion se manifeste lorsqu'un flux de paquets est plus important en entrée qu'en sortie d'un équipement réseau (par exemple dans un routeur, un *switch* ou encore un *proxy*). Différentes informations réseau peuvent être observées par la machine émettrice pour déterminer ces congestions. La perte de paquets, l'évolution de la latence, et les *flag*s Explicit Congestion Notification (ECN) sont les trois principaux indicateurs utilisés par les algorithmes de contrôle de congestion. Lorsqu'un évènement de perte de paquets se produit, il indique généralement un débordement du *buffer*. En effet, lorsque le réseau se trouve en situation de forte congestion, c'est-à-dire que l'intégralité de l'espace mémoire du *buffer* est occupée par des paquets en attente de transmission, tout nouveau paquet est alors supprimé et ne peut donc pas être transmis aux récepteurs. En ce qui concerne l'évolution de la latence, elle représente un indicateur du taux de remplissage des *buffer*s. En effet, lorsque les *buffer*s du chemin réseau se remplissent, alors le temps de transmission croît. Ceci se traduit donc par une augmentation de la latence. La réciproque est aussi vraie, lorsque les *buffer*s se vident, le temps de transmission diminue, par conséquent la latence diminue. Enfin, ECN est une option du protocole TCP et IP permettant aux équipements intermédiaires de signaler une congestion à l'émetteur pour qu'il diminue son débit avant que des pertes de paquets ne se produisent.

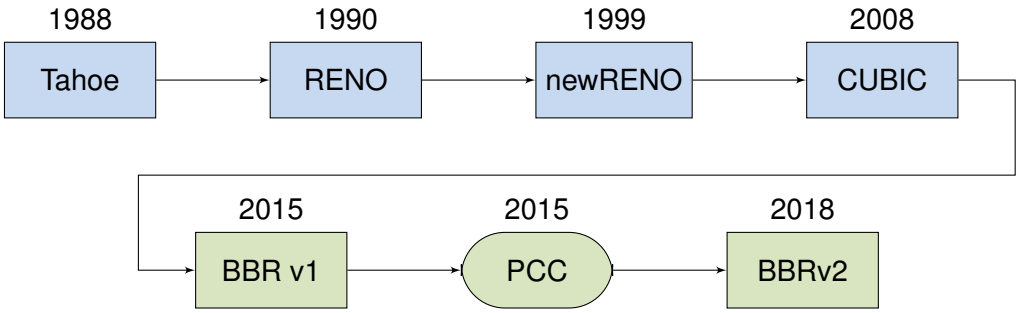


FIGURE 2.4 – Évolution exhaustive des algorithmes de contrôle de congestion.

Pour rappel, une grande partie des algorithmes de contrôle de congestion se base sur la taille de la fenêtre de congestion (Congestion Window Size en anglais) pour déterminer le débit d'un flux. La taille de la fenêtre de congestion correspond au nombre de paquets que l'émetteur a le droit d'envoyer sans acquittement. Donc plus cette valeur est importante, plus le nombre de paquets envoyés par l'émetteur sera important par unité de temps. Le débit est défini par l'équation suivante :

$$D = \frac{CWS}{RTT + \Delta_{emission}} \tag{2.1}$$

avec CWS correspondant à la taille de la fenêtre de congestion, RTT correspond aux RTT de la connexion et $\Delta_{emission}$ correspond aux temps d'émission entre le premier paquet et le dernier paquet d'une même fenêtre de congestion. Ainsi la taille de la fenêtre de congestion est proportionnelle au débit d'émission d'un flux sur un réseau.

D'autres algorithmes de contrôle de congestion, non étudié dans cette thèse, ne possèdent pas de fenêtre de congestion et détermine le débit d'une toute autres manières, c'est le cas des algorithmes de contrôle de congestion dit "rate based" comme Performance-oriented Congestion Control (PCC), Google Congestion Control (GCC) ou Network-Assisted Dynamic Adaptation (NADA)

L'évolution des algorithmes de contrôle de congestion les plus connus sont visibles dans la figure 2.4.

2.2.1 Introduction des algorithmes de contrôle de connexion : RENO

L'algorithme de contrôle de congestion RENO [26, 27] est l'un des premiers à voir le jour en 1990. Il a été très largement déployé sur le réseau internet et est aujourd'hui toujours utilisé. Son fonctionnement reprend le même principe que le premier algorithme de contrôle de congestion, Tahoe (1988). Ils fonctionnent tous deux sur le principe AIMD et détectent les congestions uniquement en se basant sur les pertes de paquets dans le réseau. Le principe de AIMD permet un accroissement linéaire de la taille de la fenêtre de congestion aussi long-

temps qu'aucun évènement de congestion n'est détecté et une diminution de la taille de la fenêtre de congestion grâce à un facteur multiplicatif lors d'évènements de congestion. Sur la figure 2.5, les trois étapes (*slow-start* (SS), *congestion avoidance* (CA) et la détection de congestion(C)) sont alors distinguables sur la courbe d'évolution de la taille de la fenêtre de congestion et sont détaillées ci-dessous.

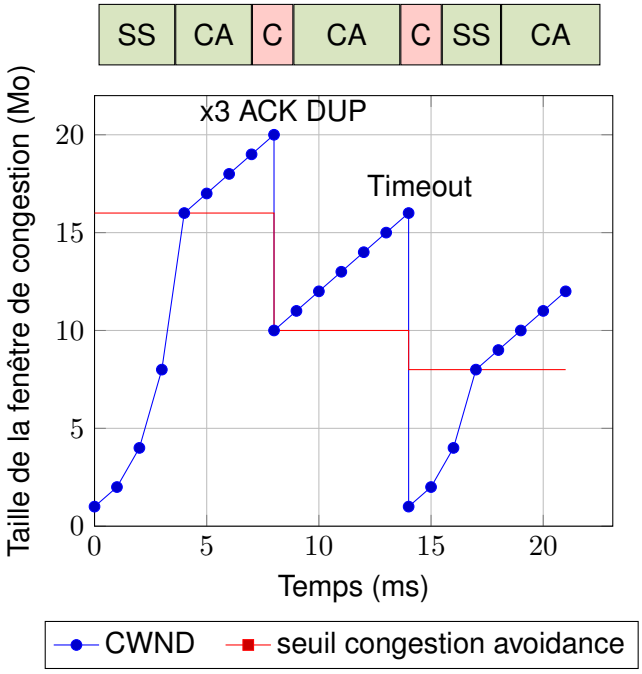


FIGURE 2.5 – Évolution de la taille de la fenêtre de congestion avec l'algorithme RENO.

Slow-Start

Lors de l'initialisation d'une nouvelle connexion de transport ou après un évènement de congestion important comme le *timeout*, le protocole de transport ne connaît pas la bande passante disponible sur le réseau. Afin de saisir rapidement toutes les ressources disponibles, l'algorithme de contrôle de congestion commence à augmenter la taille de la fenêtre avec une phase nommée "*slow-start*". L'augmentation de la taille de la fenêtre de congestion (CWND) lors du "*slow-start*" est calculée avec cette équation :

$$CWND = CWND + MSS$$

où MSS (Maximum Segment Size) représente la taille maximale d'un paquet dans le flux. Par exemple sur Linux, la taille de la fenêtre de congestion est par défaut initialisée à 10 paquets et augmente rapidement pour déterminer le débit disponible le plus rapidement possible.

Congestion avoidance

La phase de *congestion avoidance* correspond à une augmentation lente du débit, car l'algorithme de contrôle de congestion estime qu'il arrive à la limite des ressources réseau disponibles. L'algorithme estime de deux manières différentes qu'il approche du débit maximal : soit après une perte de paquet n'entraînant pas de *timeout* soit lorsque la taille de la fenêtre de congestion dépasse un seuil (seuil congestion avoidance) recalculé après chaque evenement de perte.

Dans cette phase, l'algorithme de contrôle de congestion a alors pour objectif de ne pas accroître son débit trop rapidement pour éviter de multiplier les pertes de paquets. En ce qui concerne l'algorithme RENO, cette étape correspond à une augmentation linéaire de la taille de la fenêtre de congestion. L'augmentation de la fenêtre de congestion lors d'une phase de *congestion avoidance* est réalisée avec l'équation suivante :

$$CWND = CWND_{t-1} + \frac{MSS^2}{CWND_{t-1}}$$

avec $CWND$ la taille de la fenêtre de congestion, $CWND_{t-1}$ la taille de la fenêtre de congestion précédent ce nouveau calcul et MSS la taille maximale d'un paquet dans le flux.

Détection de la congestion et diminution du débit

Toujours pour l'algorithme RENO, la détection d'une congestion entraîne alors une diminution de la taille de la fenêtre de congestion. Deux types de congestion sont alors à distinguer :

— *Réception de trois paquets "DUP ACK" :* Lorsque le récepteur détermine qu'un paquet est manquant, il informe l'émetteur en envoyant un paquet "DUP ACK" pour l'informer qu'un paquet est manquant. Ce paquet "DUP ACK" est envoyé à chaque fois que le récepteur reçoit un paquet de données alors qu'un paquet précédent n'a pas été reçu. Lorsque l'émetteur en reçoit 3 consécutifs, il considère le paquet perdu et détecte une faible congestion dans le réseau. Il rentre alors dans le mode *fast recovery* pour retransmettre le(s) paquet(s) manquant(s) et diminue ensuite la taille de la fenêtre de congestion de moitié pour limiter la congestion dans le réseau.

— *Le timeout :* lorsque l'émetteur n'a pas reçu d'acquittement du récepteur durant une période déterminée au préalable, appelée Retransmission TimeOut (RTO), le récepteur considère le paquet comme perdu et en déduit que le chemin réseau est fortement congestionné. Il décide alors de diminuer la taille de la fenêtre de congestion à 1 et d'augmenter cette nouvelle valeur avec une phase de *slow-start*.

2.2.2 Amélioration de RENO : newRENO

Les performances de RENO sont correctes lorsque le taux de perte est faible sur le réseau, mais lorsque le nombre de pertes augmente, les performances de RENO décroissent fortement. La principale raison en est que l'algorithme RENO est étudié pour détecter une unique perte durant l'émission des paquets d'une fenêtre de congestion. Donc lorsqu'il y aura plusieurs pertes de paquets dans la même fenêtre d'émission, RENO aura besoin de l'acquittement du premier paquet manquant pour détecter la perte d'un second paquet et ainsi de suite. C'est pour cette raison que RENO considère alors différents évènements distincts de congestion (réception de trois paquets "DUP ACK"), et réalise alors plusieurs diminutions de la taille de la fenêtre de congestion. Or ces multiples pertes sont dues au même évènement de congestion, de ce fait la diminution du débit est trop importante donc les performances de RENO sont alors dégradées.

Proposé en 1999, NewRENO [10] est une modification mineure de l'algorithme RENO, mais qui augmente grandement les performances de l'algorithme. La modification a lieu lors de la réception de trois paquets "DUP ACK". Ainsi lorsque l'algorithme rentre en phase de *fast recovery*, il reste dans cet état aussi longtemps que tous les paquets de la fenêtre de congestion ne sont pas acquittés. C'est uniquement lorsqu'ils sont acquittés qu'il fera la diminution de la taille de la fenêtre de congestion. Si plusieurs pertes sont commises dans la même fenêtre de congestion, alors newRENO les considère comme une unique perte et réalise une seule diminution de la taille de fenêtre de congestion.

2.2.3 L'option Selective Acknowledgments (SACK)

L'optimisation de RENO par newRENO n'est toujours pas une solution optimale. La détection de plusieurs pertes sur la même fenêtre de congestion est lente, c'est-à-dire qu'il faut n RTT (temps aller-retour entre l'émetteur et le récepteur) pour détecter n pertes. C'est pourquoi une nouvelle option de TCP créée en 1995 fut nommée Selective Acknowledgments (SACK). Cette option se négocie lors de l'ouverture de la connexion TCP et permet l'amélioration de la détection d'erreurs. SACK indique l'intégralité des paquets reçus même si ces paquets sont discontinus. Pour cela, il acquitte tous les paquets jusqu'aux numéros de séquence N, mais précise aussi à l'émetteur qu'il a reçu $N+2$ et $N+4$ (si $N+1$ et $N+3$ sont manquants). Ainsi en un seul RTT, l'émetteur peut savoir le nombre de paquets manquants et les retransmettre plus rapidement lors de la phase de *fast recovery*.

2.2.4 Augmentation des débits des réseaux : CUBIC

Limitation de l'algorithme de contrôle de congestion RENO

L'algorithme de contrôle de congestion RENO admet des limites dans plusieurs configurations réseau [28]. Par exemple, si les conditions de réseau possèdent un débit important (de l'ordre de 10Gbit/sec) et une latence de 100ms, alors l'algorithme de contrôle de congestion RENO a besoin de plus d'une heure pour atteindre le débit optimal [28]. Le même problème persiste pour les réseaux ayant des valeurs importantes de RTT. En effet, RENO actualise les valeurs de la taille de la fenêtre à chaque réception d'acquittement. Lorsque le RTT est important, la mise à jour de la fenêtre de congestion est plus lente. Donc, dans les situations décrites précédemment, plus généralement avec un Bandwidth Delay Product (BDP) important, RENO sous-utilise sévèrement la capacité du lien pendant le téléchargement d'un contenu. CUBIC propose alors une solution pour répondre à cette nouvelle problématique.

Algorithme de contrôle de congestion : CUBIC

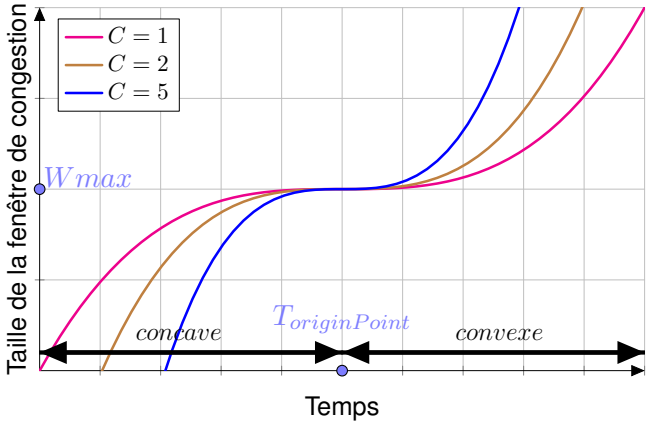


FIGURE 2.6 – Évolution de la taille de la fenêtre de congestion avec l'algorithme de contrôle de congestion CUBIC.

Les situations réseau avec une latence et/ou un débit important imposent une modification de l'algorithme de contrôle de congestion RENO dans la phase de *congestion avoidance*. CUBIC [11, 28] propose donc une nouvelle vision de la phase de *congestion avoidance*, mais les phases de *slow-start* et de diminution de la taille de la fenêtre de congestion de RENO restent rigoureusement identiques dans cette nouvelle version d'alogirithme de controle de congestion. Durant la phase de *congestion avoidance*, CUBIC modifie l'accroissement linéaire de RENO par une forme CUBIC, défini comme ceci :

$$W(t) = C(t - K)^3 + W_{max}. \tag{2.2}$$

avec C une constante permettant de définir "l'agressivité" de l'algorithme de contrôle de congestion, W_{max} la taille de la fenêtre de congestion maximale avant l'évènement de congestion et K (correspondant à la valeur de $T_{originPoint}$) est la période de temps que prend la fonction cubic pour augmenter la taille actuelle de la fenêtre à W_{max}. La figure 2.6 illustre un exemple d'accroissement de la taille de la fenêtre de congestion avec CUBIC lors d'une phase de *congestion avoidance* avec plusieurs valeurs de C.

Cette modification permet d'accroître plus rapidement la taille de la fenêtre de congestion jusqu'à la précédente capacité du lien connu avant le dernier évènement de congestion (partie concave de la courbe cubique). Puis la valeur plateau est atteinte et l'algorithme stabilise le débit autour de cette valeur (transition entre la partie concave et convexe). Enfin, l'accroissement de la taille de la fenêtre de congestion est plus important avec la partie convexe de la courbe qu'avec RENO. Cet accroissement permet d'augmenter rapidement le débit dans des réseaux à débit important, chose que ne permettait pas RENO. Une autre différence majeure entre RENO et CUBIC est le fait qu'il ne repose plus sur la réception des acquittements pour augmenter la taille de la fenêtre de congestion. En effet, celle-ci repose uniquement sur l'instant où CUBIC a connu un évènement de congestion. Ainsi le RTT n'impacte plus la phase d'augmentation du débit.

Aujourd'hui, l'algorithme de contrôle de congestion CUBIC est l'un des plus répandus sur le réseau. Lors des derniers travaux de recherche à l'IETF, l'algorithme de contrôle de congestion CUBIC est défini comme référence pour évaluer l'équité et la performance des nouveaux algorithmes de congestion pour les flux de données temps réel [29].

2.2.5 Une nouvelle option pour la sortie du slow-start : l'hystart

Pour rappel, deux phases existent dans les algorithmes de contrôle de congestion présentés précédemment : le *slow-start* et le *congestion avoidance*. Une nouvelle option, nommée hybrid start (hystart), a été créée pour passer du *slow-start* à *congestion avoidance*.

De façon plus générale, la transition entre ces deux phases peut se déclencher lorsqu'un de ces évènements se produit :

— *Perte de paquets :* Comme une diminution du débit, la perte de paquets est aussi un évènement pour passer de la phase de *slow-start* à la phase de *congestion avoidance*. En effet, lorsqu'un évènement de perte de paquets se produit, c'est-à-dire que le réseau commence à saturer, la taille de la fenêtre de congestion est réduite (par exemple d'un tiers avec l'algorithme de contrôle de congestion CUBIC) et l'algorithme de contrôle de congestion passe dans une phase de *congestion avoidance*. Cette réaction à la perte de paquets est implémentée dans des algorithmes de contrôle de congestion tels que RENO et CUBIC.

— *Le dépassement d'un seuil :* Lorsque la taille de la fenêtre de congestion atteint une certaine valeur seuil, l'algorithme de contrôle de congestion passe dans la phase de *congestion avoidance*. La valeur de seuil par défaut dans l'implémentation Ubuntu 16.04 de CUBIC est de 644 paquets.

Une nouvelle solution permet de sortir du *slow-start* pour passer en *congestion avoidance* : hystart. Proposée en 2011, l'option hystart a été présentée dans les documents [30, 31] et est détaillée dans le RFC 8312 [11]. Hystart est une option implémentée dans les protocoles de transport comme TCP et QUIC. L'option hystart est présentée comme une solution pour déterminer un point de sortie "sûr" du *slow-start*, c'est-à-dire pour passer d'une phase de *slow-start* à *congestion avoidance*. Si une augmentation du RTT est détectée pendant le *slow-start*, hystart considère que cela indique un début de congestion du réseau et passe du *slow-start* à *congestion avoidance* afin d'éviter d'augmenter la perte de paquets. L'option hystart est largement déployée sur Internet. En effet, elle est activée par défaut sur Linux et dans différentes piles QUIC telle que QUIC-GO.

2.2.6 Changement de concept des algorithmes de contrôle de congestion : BBR

Limitation des précédents algorithmes de contrôle de congestion

Après de nombreuses années où les algorithmes de contrôle de congestion RENO et CUBIC ont été largement utilisés, des chercheurs de Google affirme, en 2015, que le déplacement de données sur l'Internet n'est pas optimisé [12], notamment que les connexions sur des réseaux WiFi publics sont dégradées et sous-utilisent les ressources du réseau. Les auteurs de l'article identifient deux problèmes majeurs au niveau des algorithmes de contrôle de congestion. Le premier est l'utilisation des pertes de paquets comme moyen de détection de congestion sur un réseau. De même le principe AIMD pour l'algorithme de contrôle de congestion RENO atteint sa limite dans les nouvelles architectures réseau.

En effet, les cartes réseau actuelles possèdent des débits de l'ordre du Gbit/sec et des tailles des *buffer*s de l'ordre du Mbit (contrairement aux précédentes cartes réseau qui était de l'ordre du Mbit/s pour le débit et du kbit pour la taille du *buffer*). Avec la détection de pertes comme évènements de congestion et le principe AIMD, les algorithmes de contrôle de congestion remplissent l'intégralité de la mémoire du *buffer* du point de saturation. Ainsi, avec l'augmentation de la taille des goulots d'étranglement (directement proportionnelle à la bande passante disponible), cela engendre des latences plus importantes sur le réseau. Pour pallier ce problème d'augmentation de la latence, la diminution de la taille des *buffer*s pourrait être envisagée. Mais une trop forte réduction de cette mémoire a pour effet une augmentation des pertes de paquets et donc une dégradation importante des performances des algorithmes de contrôle de congesiton. C'est pourquoi une préconisation a été faite de quitter ces types d'algorithmes de contrôle de congestion pour en proposer de nouveaux.

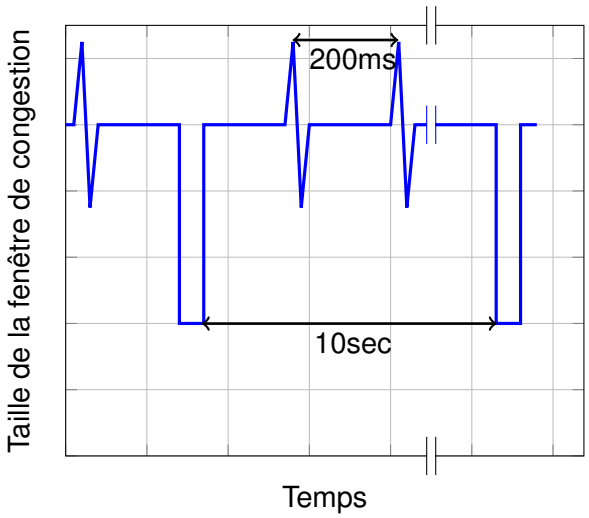


FIGURE 2.7 – Évolution de la taille de la fenêtre de congestion avec l'algorithme de contrôle de congestion BBR.

Nouveau concept dans les algorithmes de contrôle de congestion : BBR

L'algorithme Bottleneck Bandwidth and Round-trip propagation time (BBR) [12,13], proposé par Google, modifie complètement le principe de fonctionnement des algorithmes de contrôle de congestion. BBR ne se base plus sur les pertes de paquets comme les précédents algorithmes de contrôle de congestion RENO ou CUBIC, mais sur une estimation de la bande passante disponible dans le point de saturation. Pour réaliser cette approximation, BBR se base sur deux paramètres : le débit de l'utilisateur et le RTT (temps aller-retour entre l'émetteur et le récepteur). L'algorithme d'évaluation du débit est détaillé dans la figure 2.7.

Sur cette évaluation du débit, deux phases sont distinctes :

— *L'évaluation du RTT :* l'algorithme BBR cherche à estimer le RTT minimum (RTT_{min}) entre l'émetteur et le récepteur. La période d'actualisation de cette valeur est de 10sec. Pour évaluer une valeur représentative du RTT_{min}, BBR réalise un vidage des *buffer*s pendant 200ms. Pour cela, il abaisse la taille de la fenêtre de congestion à 4. La valeur du RTT_{min} permet d'estimer le remplissage des *buffer*s sur l'intégralité du chemin parcouru. En effet, l'écart entre la valeur du RTT en cours et la valeur du RTT_{min} est proportionnel au taux de remplissage des *buffer*s du chemin réseau. Ainsi lorsque les *buffer*s ont un nombre important de paquets, alors le RTT en cours est plus important que le RTT_{min} et réciproquement.

— *L'évaluation du débit :* un mécanisme permet d'évaluer si une augmentation de débit engendre plus de congestion sur le réseau. Pour cela, il modifie la fenêtre de congestion pendant un RTT à 1.25 fois le débit et sur la période suivante à 0.75 fois le débit. En observant l'évolution du RTT en cours, nous pouvons savoir si la bande passante disponible du

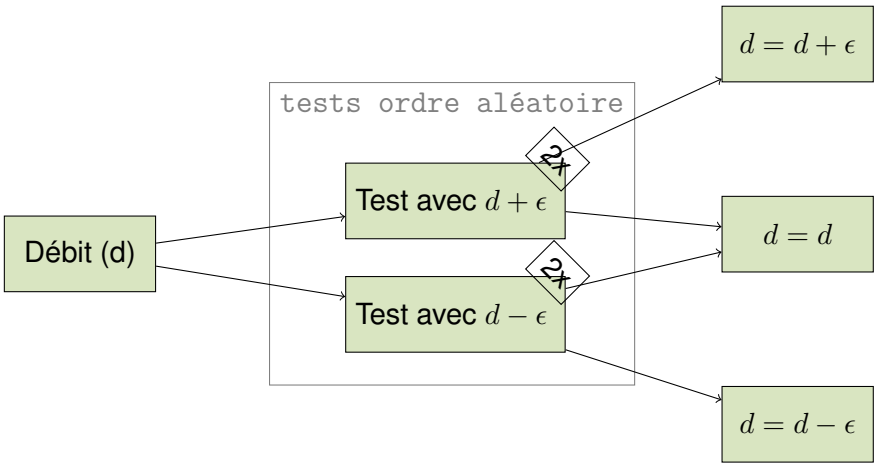


FIGURE 2.8 – Algorithme pour l'évolution du débit de PCC

point de saturation a augmenté (aucune évolution du RTT lorsque le débit augmente de 1.25) ou si le débit reste inchangé (augmentation de la latence lorsque le débit augmente à 1.25).

À partir de l'ensemble de ces informations, BBR arrive à déterminer la taille de la fenêtre de congestion et donc le débit du flux.

2.2.7 Autre concept pour modifier le principe AIMD : PCC

PCC [14] est un nouvel algorithme de contrôle de congestion proposé par l'université de l'Illinois. Cet algorithme, comme BBR, est aussi en opposition au fonctionnement AIMD. En effet, il ne se base ni sur les pertes de paquets pour déterminer la congestion (comme RENO ou CUBIC) ni sur un fonctionnement fondé sur l'évolution de latence comme BBR. Il s'appuie sur une méthode de tests avec différentes valeurs de débit décrites dans la figure 2.8.

L'algorithme de contrôle de congestion entame l'augmentation de la taille de la fenêtre de congestion par une phase du *slow-start* comme l'ensemble des algorithmes de contrôle de congestion présentés. Lorsqu'il sort de la phase de *slow-start*, son débit d sert de référence. Ensuite, il réalise les 4 tests dans un ordre aléatoire avec des débits différents (deux tests qui ont pour débits $d + \epsilon$ et deux autres qui ont pour débit $d - \epsilon$). La décision pour l'augmentation ou la diminution de la taille de la fenêtre de congestion est basée sur deux paramètres : le débit réel mesuré et le nombre de paquets perdus dans chacun des tests. Ensuite l'algorithme choisit le plus intéressant pour l'utilisateur, c'est-à-dire :

— *Augmentation du débit :* Les deux tests avec $d + \epsilon$ sont plus avantageux pour l'utilisateur, c'est à dire qu'il possède un débit supérieur à d et sans aucune augmentation du taux de perte.

— *Diminution du débit :* Les deux tests avec $d - \epsilon$ sont plus avantageux pour l'utilisateur, c'est-à-dire un débit similaire avec moins de perte.

— *Débit inchangé :* Un test d'augmentation du débit est plus avantageux pour l'utilisateur et un autre test de diminution de débit l'est également.

2.2.8 Bilan sur les modifications des algorithmes de contrôle de congestion

Les algorithmes de contrôle de congestion doivent s'adapter en permanence aux évolutions du réseau et aux besoins des utilisateurs. Ainsi depuis les années 1980, les chercheurs ont sans cesse amélioré les algorithmes de référence comme RENO et CUBIC. Même s'ils sont aujourd'hui très largement déployés sur le réseau internet, de nouveaux algorithmes cherchent à les remplacer pour améliorer leurs performances. Depuis 2015, un changement majeur a été réalisé sur le principe des algorithmes de contrôle de congestion avec l'abandon du principe AIMD (qui garantissait l'équité réseau) et de la détection de pertes comme indicateur principal de congestion (par exemple avec l'apparition de BBR et PCC). Aujourd'hui, BBR est en cours de déploiement sur le réseau et révolutionne les algorithmes de contrôle de congestion avec des principes complètement différents de ses prédécesseurs. De nombreux autres algorithmes de contrôle de congestion sont en cours d'étude ; c'est le cas de fastCC présent dans l'implémentation de pico-QUIC [32], ou encore COPA [33], Vivase [34] et beaucoup d'autres. Dans cet état de l'art, nous avons décidé de présenter uniquement ceux qui sont étudiés dans cette thèse. Aujourd'hui, la question de l'équité est toujours aussi importante voir plus que précédemment. En effet, comment les algorithmes de contrôle de congestion se répartissent-ils la bande passante disponible lorsqu'ils sont en concurrence ? Peuvent-ils être équitables entre-eux et fournir les débits souhaités sans pénaliser un autre flux ?

2.3 Qualité de service

La Quality of Service (QoS) est la possibilité de transporter les données (en fonction des services) des utilisateurs dans de bonnes conditions réseau (par exemple en termes de débit, délais de transmission, gigue, taux de perte de paquets, ...). Il est alors important de pouvoir classifier les flux pour répondre aux exigences des utilisateurs pour qu'ils obtiennent une qualité optimale. Puis dans un second temps nous aborderons la question de l'équité pour aborder le problème de partage de ressources lorsque celle-ci viennent à manquer.

2.3.1 Classification des services

Avant de parler d'équité dans le réseau, il est important de comprendre que chaque flux n'a pas les mêmes exigences en termes de ressources réseau pour qu'ils obtiennent un fonctionnement optimale. Plusieurs études ont été réalisées pour classifier les flux en fonction de leurs exigences : par exemple pour les applications de la 5G [5], ou pour les services différenciées (diffserv) [35].

C'est le deuxième cas que nous détaillerons dans cet état de l'art : diffserv est une architecture réseau permettant de spécifier, par un mécanisme simple, le type de flux qui est transporté par les paquets IP. Il utilise 8 bits dans l'entête IP pour caractériser le flux. Ainsi, l'intégralité des équipements du réseau à la possibilité de lire ces bits, et peut alors appliquer des services différents à chacun des flux en fonction des besoins pour que l'utilisateur obtienne une QoS optimale. Par exemple, les services pour le transport de la voix ou de streaming vidéo recherche plutôt les transmissions à faible latence et ils pourront donc être priorisés dans les équipements réseaux.

La RFC4594 [36] définit différents types de classes de trafic sur le réseau comme le trafic interactif en temps réel, la téléphonie, le streaming vidéo, les service à faible latence, ... Chacune de ces catégories est ensuite affectée à un profil de trafic qui caractérise la latence ou le taux de perte qui peut être supporté par ce types de services.

Lorsque l'ensemble des flux peut obtenir les ressources réseau souhaitées, aucun problème ne se pose. Mais quand le réseau est congestionné alors un partage des ressources doit être effectué. C'est dans ces conditions que l'équité doit être mesurée pour s'assurer qu'aucun utilisateur ne possède de ressources réseau au détriment d'un ou plusieurs autres utilisateurs.

2.3.2 Équité réseau

Lorsque l'ensemble des flux ne peut pas obtenir l'ensemble des ressources souhaitées, alors un partage doit être réalisé entre les différents flux. Pour savoir si ce partage est juste nous parlons alors d'équité entre les flux. De multiples définitions de l'équité existent au travers de la littérature. Dans le domaine du réseau, deux définitions sont largement utilisées pour évaluer l'équité : *max-min fairness* et *proportional fairness*.

Max-min fairness

L'équité "*max-min fairness*" est un principe d'allocation de ressources afin de pouvoir partager au mieux les ressources entre différents utilisateurs. Intuitivement, le partage des ressources devrait être égal entre les utilisateurs de façon à ne pénaliser aucun d'entre eux. Mais certains peuvent demander une plus faible allocation de ressources que d'autres. C'est-à-dire qu'ils n'ont pas forcement besoin d'obtenir une part équivalente de la bande passante (par exemple un service de discussion et une vidéo en streaming). Ainsi l'équité *max-min fairness*

permet de donner la priorité aux faibles demandes d'allocation et de répartir la partie restante sur les utilisateurs qui ont des besoins plus importants. Si ces besoins ne sont pas satisfaits, alors chacun aura une allocation identique au maximum des capacités disponibles. Plus formellement, l'article [37] définit l'attribution des ressources *max-min fair*, si et seulement si :

— Les plus faibles demandes sont priorisées par rapport aux plus fortes demandes de ressources ;

— Aucun utilisateur n'obtient plus que ce qu'il a demandé ;

— Les utilisateurs qui font une demande supérieure à ce qui est disponible obtiennent tous la même quantité de ressources.

Différentes solutions existent pour déterminer l'équité *max-min fairness* ; une des plus connues est l'algorithme de *progressive filling* [38]. Cet algorithme initialise l'ensemble des allocations de ressources à 0 et les augmente toutes au même rythme. Lorsque la plus petite demande d'allocation de ressources a obtenu l'intégralité des ressources voulues, son accroissement est arrêté, les autres continuent d'augmenter et ainsi de suite. L'algorithme de *progressive filling* s'arrête lorsque tous les utilisateurs ont obtenu l'allocation de ressources voulue, ou que la capacité maximale de ressources disponibles est atteinte. Cette allocation est appelée *max-min fair*. Par exemple, quatre utilisateurs souhaitent respectivement $\{2, 3, 4, 5\}$ sur une capacité maximale de 12. En appliquant l'algorithme de *progressive filling*, le partage des ressources est respectivement $\{2, 3, 3.5, 3.5\}$ pour chaque utilisateur.

Dans [37], les auteurs discutent des avantages et des inconvénients d'une telle solution. En ce qui concerne les avantages, cette solution garantit un minimum d'équité entre les flux comparé à une logique de premier arrivé, premier servi. Cette équité est garantie par le fait qu'un flux qui veut obtenir plus d'allocations de ressources ou qui est mal géré par l'algorithme de contrôle de congestion ne pourra pénaliser que lui-même, car le débit est garanti pour les autres utilisateurs. Par exemple, la file d'attente *fair queuing* garantit l'équité *max-min fair*. Mais cette définition admet des limites, lorsqu'un flux n'utilise pas toute la part de l'allocation qu'il a demandée. S'il repartage le surplus avec d'autres utilisateurs, l'équité *max-min fair* n'est plus optimale.

Weighted Max-Min Fairness

Dans [37], les auteurs approfondissent le concept d'équité *max-min fairness* avec une notion de priorité, qui n'est plus liée à l'utilisateur qui a la plus faible demande de ressources, mais plutôt liée à un poids qui est accordé à chaque utilisateur. Plus formellement, l'attribution des ressources est *weighted max-min fair* si et seulement si :

— Les ressources sont allouées par ordre d'importance croissante à chaque utilisateur, normalisées par le poids ;

— Aucun utilisateur n'obtient plus que ce qu'il a demandé ;

— Les utilisateurs ayant des demandes insatisfaites obtiennent une allocation des ressources proportionnelles à leur poids.

Nous pouvons toujours déterminer l'équité *weighted max-min fairness* à partir d'un algorithme de *progressive filling* légèrement modifié pour répondre à cette nouvelle définition. Par exemple, quatre utilisateurs demandant respectivement $\{2, 3, 4, 5\}$ en ayant un poids respectif de $\{2, 1, 3, 4\}$ sur une capacité maximale de 12. En appliquant les règles de *weighted max-min fairness*, le partage des ressources est respectivement $\{2, 1, 4, 5\}$ pour chaque utilisateur.

Proportional fairness

Définie dans [39], l'algorithme de *proportional fairness* permet d'accorder une importance à l'efficacité de l'utilisation des ressources, c'est-à-dire qu'elle attribue les ressources par rapport aux demandes des utilisateurs proportionnellement à l'efficacité avec laquelle ils les utilisent. En d'autres termes, elle maximise la somme des logarithmes des charges des utilisateurs, c'est-à-dire les ressources doivent être allouées aux utilisateurs dont les charges augmentent le plus rapidement.

$$\max \sum_{i=1}^{c} \log C_i(A_i) \tag{2.3}$$

où C_i la charge demandée du i-ème utilisateur, A la répartition proportionnellement équitable du i-ème utilisateur, et c le nombre d'utilisateurs. Notez que la fonction log est utilisée pour obtenir un comportement lent, c'est-à-dire pour éviter de négliger les utilisateurs avec des faibles demande de bande passante [40].

2.3.3 Métrique

La propriété de l'équité max-min fairness décrite précédemment peut se baser sur différentes méthodes de calcul. L'organisme de l'IETF a produit un document à cet égard, à savoir *Metrics for Evaluation of Congestion Control Mechanisms* [41], qui présente trois mesures principales d'évaluation de l'équité :

(i) La première métrique a été définie à l'origine par Jain [42]. Soit T_i le débit mesuré du lien i, soit n le nombre de flux, et soit O_i le débit optimal obtenu par la définition de max-min fairness. L'*indice de Jain* est défini comme suit

$$J = \frac{(\sum_{i=0}^{i=n} x_i)^2}{n \sum_{i=0}^{i=n} x_i^2}. \tag{2.4}$$

avec $x_i = T_i/O_i$. Son principal avantage est qu'il est borné entre $1/n$ et 1 et qu'il ne tient pas en compte la bande passante disponible pour déterminer les valeurs d'équité. Ainsi l'équité pourra être comparée indifféremment des situations réseaux. Cependant, il ne fonctionne que sur un moment précis de concurrence entre plusieurs flux et donc il ne capture pas l'équité durant la durée d'une session d'un utilisateur comme lui pourrait le faire.

(ii) La deuxième mesure d'équité s'appelle *product fairness* et est définit ci-dessous :

$$P = \prod_{i=0}^{i=n} d_i \tag{2.5}$$

Maximiser la valeur de P nécessite l'évitement des débits nuls (ou très faibles), et donc d'atteindre à nouveau un partage max-min équitable des débits. Cette mesure souffre de la même faiblesse que l'indice de Jain : elle s'applique sur un moment donné et ne prend toujours pas en compte la durée de la session. De plus, sa valeur maximale (qui est égale à $\left(\frac{B}{n}\right)^n$ où B est la bande passante disponible au goulot d'étranglement) dépend de cette bande passante. Elle ne permet donc pas de comparer l'équité dans différentes configurations de réseau avec différentes bandes passantes.

(iii) La troisième métrique proposée est l'*epsilon fairness* [43] :

$$\frac{\max d_i}{\min d_i} < 1 - \epsilon \tag{2.6}$$

avec d_i le débit d'un flux i et ϵ un coefficient pour déterminer l'équité entre les flux. La frontière entre équitable et inéquitable est déterminée par la variable ϵ : plus ϵ est élevée, plus le ratio entre équitable et inéquitable est strict. Cette mesure met l'accent sur les débits extrêmes et ignore les débits intermédiaires.

2.3.4 Bilan sur l'équité et les métriques

Plusieurs catégories de flux existent et doivent se partager l'intégralité des ressources du réseau lorsque celui est congestionné. Pour simplifier nos études, dans la suite de la thèse nous considérons une unique classe pour réaliser les études d'équité réseau (téléchargement de contenus). Plusieurs définitions de l'équité existent. Aujourd'hui, les plus utilisées sur le réseau sont les principes de *max-min fairness* et *proportional fairness*. Pour pouvoir réaliser ces mesures, différentes métriques existent comme l'indice de Jain, le *product fairness*, ou encore l'*epsilon fairness*. Mais toutes ces métriques s'appliquent à un instant donné d'une session. Donc ces méthodes mesurent l'équité si et seulement si on admet que les flux convergent vers des valeurs de débit stable (même débit durant toute la session). Cependant, la réalité du partage de la bande passante est que les débits évoluent considérablement au cours d'une session. Cela démontre donc la limite de toutes ces mesures.

2.4 Mesure d'équité des algorithmes et des protocoles de transport

2.4.1 L'équité entre les algorithmes de contrôle de congestion

Lors de la présentation des résultats des nouveaux algorithmes de contrôle de congestion, l'ensemble des articles [12, 14, 28] propose une section sur l'évaluation de l'équité qui est généralement sous-évaluée.

Dans le papier présentant le nouvel algorithme de contrôle de congestion BBR [12], les auteurs évaluent l'équité entre plusieurs flux basés sur l'algorithme de contrôle de congestion BBR et démontrent qu'ils sont équitables. Les seules comparaisons avec un autre algorithme de contrôle de congestion (CUBIC) sont une étude de performance sur leurs propres réseaux, nommés B4. En effet, ils évaluent les performances de BBR et démontrent une augmentation du débit entre 2 et 25 fois supérieure à CUBIC. Ils ne démontrent en aucun cas l'évolution de l'équité et l'impact pour les utilisateurs lorsque BBR partage un goulot d'étranglement avec un autre algorithme de contrôle de congestion.

Dans un autre papier [14], présentant l'algorithme de contrôle de congestion PCC, nous retrouvons les même tests d'équité (à savoir si PCC est équitable avec lui-même) et d'amélioration des performances (10 fois supérieures à celles de TCP). Même si les auteurs indiquent cette fois qu'il faut plusieurs connexions TCP pour obtenir des débits similaires, l'évaluation de l'équité et l'impact pour les utilisateurs ne sont pas clairement démontrés.

De plus, il a été prouvé dans [44] que les caractéristiques réseau ont une forte influence sur la performance des algorithmes de contrôle de congestion. Par exemple dans une certaine configuration réseau, BBR peut obtenir plus de débit que CUBIC et vise-versa. Il est alors primordial de ne pas se baser sur quelques tests isolés pour réaliser l'évaluation d'équité ou de performance, mais au contraire de faire varier les situations réseau pour l'évaluer.

2.4.2 Fairness et performances des protocoles de transport

Mesure d'équité entre TCP et QUIC

Plusieurs études récente retrace la performance et l'équité du protocole QUIC versus TCP. Une première étude [9] propose une première analyse de l'équité de QUIC sur un réseau à débit constant. Les auteurs démontrent que QUIC est plus rapide que TCP. Dans cette étude, la version utilisée de QUIC est calibrée pour obtenir un comportement similaire aux serveurs déployés par Google sur Internet. Lors de leur calibrage, ils montrent que la valeur initiale de la taille de la fenêtre de congestion a un impact sur le temps de téléchargement d'un gros fichier (jusqu'à 50%).

D'autres études ont été réalisées dans la thèse de Amit Srivastava de l'institut polytechnique de Worcester [45]. Durant cette thèse, ils se sont focalisés sur trois axes d'études, à savoir la latence, le taux de pertes et l'équité. Les différentes mesures ont été réalisées sur des réseaux à débit constant avec différentes valeurs de latence, taux de perte, débit et nombre de flux. Ils ont démontré que la latence n'a aucun impact dans les performances de QUIC. En ce qui concerne le taux de perte, ils ont montré que cette valeur a un impact significatif sur les performances de QUIC comparées à celles de TCP. En effet, ils ont prouvé que lorsqu'une perte de paquets se produit (une congestion) le protocole QUIC a tendance à moins réduire son débit que TCP. Pour terminer, ils ont aussi réalisé des mesures d'équité entre le protocole QUIC et TCP et ont établi que le protocole QUIC obtient une part de la bande passante disponible supérieure à celle de TCP. Ces études permettent de poser une base dans l'évaluation de l'équité et des performances de QUIC versus TCP. Durant nos études présentées dans cette thèse, nous étendrons ces résultats sur des réseaux à débit variable et approfondirons les interprétations des résultats afin de déterminer les paramètres de QUIC qui ont un impact sur l'équité.

A notre connaissance, nos études ont été les premières à mesurer l'impact sur l'équité du nombre de connexions TCP émulées dans QUIC avec un flux TCP concurrent sur un réseau mobile et fixe avec une implémentation par défaut de QUIC.

2.4.3 Phénomènes impactant les performances d'un protocole de transport

Les performances des protocoles de transport peuvent dépendre de différents phénomènes qui impactent directement le débit des utilisateurs.

Impact de l'option hystart sur les performances : Pour rappel, lors du démarrage d'une connexion de transport, le protocole débute par une phase de *slow-start* pour déterminer le plus rapidement possible le débit de l'utilisateur avant de passer dans la phase de *congestion avoidance*. Ces deux phases sont décrites dans la partie présentant l'algorithme de contrôle de congestion RENO (voir 2.2.1) ainsi que les transitions possibles entre les deux phases dans la sous-section 2.2.5.

Une précédente étude [46] démontre que l'option hystart peut poser des problèmes lors de la transition entre ces deux états. En effet, les auteurs analysent la performance de l'algorithme de contrôle de congestion CUBIC avec l'option hystart active. Tous les résultats ont été obtenus avec le simulateur *ns-3* et une plateforme nommée *w-ilab.t* [47]. Dans le cas d'un réseau mobile, le document met en lumière un véritable problème de performance dans certaines situations. Mais cette étude n'aborde ni l'option hystart implémentée dans le protocole QUIC, ni la mesure de l'impact de cette option sur l'équité.

Performances de TCP : Les performances de TCP dans les réseaux mobiles sont un sujet largement couvert dans la littérature, par exemple dans [48, 49]. Dans [44], les performances des contrôles de congestion sont évaluées avec TCP et aussi QUIC. Ces évaluations s'ap-

puient sur l'émulateur de réseau mobile Mahimahi [50, 51] que nous avons également utilisé (voir section 3.1.2). Cependant, à notre connaissance, l'équité du protocole QUIC dans le réseau mobile n'a pas été spécifiquement analysée dans la littérature.

Variation de débit sur un réseau mobile : Le débit d'un utilisateur sur un réseau mobile peut être affecté par plusieurs phénomènes, dont l'un est la qualité du signal reçu par UE. La qualité du signal dépend de multiples paramètres tels que la distance UE - antenne, la présence d'obstacles entre eux et les interférences. Typiquement, une grande distance UE - antenne peut réduire considérablement le débit. Un autre phénomène bien connu est la charge cellulaire. En effet, le débit alloué à l'UE dépend directement de cette charge [52] car une cellule partage ses ressources entre les utilisateurs actifs.

De même, les retards du réseau mobile sont causés par plusieurs phénomènes et peuvent être répartis entre les retards de l'interface hertzienne, de la station de base, du réseau filaire de l'opérateur et les retards liés à la localisation du serveur sur Internet. Selon [53] les valeurs typiques de ces retards sont respectivement de l'ordre de 25ms, 30ms et 50ms pour un serveur du même pays. Cela s'ajoute aux délais de mise en file d'attente et d'ordonnancement dans la station de base qui dépendent généralement de la taille du *buffer* alloué à un utilisateur et de la variation du débit. Dans les pratiques d'ingénierie actuelles, le dimensionnement des *buffer*s (BS) se fait en fonction de : $BS = D * \text{RTT}_{\text{EE}}$ où D est le débit du réseau le plus élevé et RTT_{EE} est le temps aller-retour de bout en bout typique. Les variations du débit de la liaison radio entraînent des variations dans les délais de mise en file d'attente. À titre d'exemple numérique, considérons un *buffer* de la taille d'un RTT de bout en bout typique de 50ms et un débit maximal de 100 Mbit/s. Une source de trafic élastique (par exemple régi par CUBIC) qui tend à remplir un tel *buffer* voit son délai de mise en file d'attente passer à 500ms lorsque le débit de la liaison radio diminue à 10Mbit/s.

Le phénomène dit du *buffer-bloat* a été observé sur les réseaux cellulaires dans plusieurs études (par exemple [54, 55]) qui rapportent constamment des valeurs RTT entre 500ms et 2s avec un transfert de données en masse depuis un serveur TCP qui charge la liaison descendante.

2.5 Bilan

Le protocole de transport TCP est un des protocoles les plus utilisés sur le réseau internet. Mais aujourd'hui, plusieurs limites font que ce protocole devient obsolète et sa mise à jour sur l'ensemble des équipements réseau reste complexe. Aujourd'hui, l'absence du multiplexage, la difficulté des mises à jour et la résistance face aux attaques sont quelques exemples des limites de TCP. Pour résoudre ce problème, Google propose un nouveau protocole nommé QUIC. Le principal objectif de QUIC est de pallier les différents points faibles de TCP. Pour cela, il se repose sur le protocole TCP et différentes améliorations provenant de son état de l'art comme le multiplexage des *stream*s avec SCTP. Mais ce nouveau protocole est implémenté dans l'ap-

plication des clients ou les serveurs. Il favorise des implémentations fermées (non partagées) comme le démontre l'étude [9]. Ainsi plusieurs versions de ce protocole peuvent facilement se retrouver en concurrence sur Internet avec différentes options implémentées ou différentes valeurs par défaut. La question de l'équité devient alors un sujet primordial. D'autant plus, qu'à l'heure actuelle l'organisme de normalisation IETF laisse beaucoup de liberté aux développeurs comme par exemple sur l'implémentation de l'algorithme de contrôle de congestion.

Les protocoles ne sont pas les seuls à subir des modifications pour être améliorés. En effet les algorithmes de contrôle de congestion subissent aussi de nombreux changements pour s'adapter aux nouvelles exigences des utilisateurs et à l'augmentation du débit sur les réseaux. Depuis leur création, les algorithmes de contrôle de congestion n'ont fait qu'évoluer. Mais en 2015, l'apparition d'une nouvelle génération d'algorithmes de contrôle de congestion modifie la façon de détecter les congestions. Par exemple BBR, ne se base plus sur le principe AIMD et sur la perte de paquets pour détecter une congestion comme le faisait ses prédécesseurs, mais sur l'évolution de la latence pour évaluer le remplissage des *buffer*s et donc la congestion. Lors des propositions des nouveaux algorithmes de contrôle de congestion, nous avons constaté que l'évaluation de l'équité était généralement sous-testée.

2.6 Problématiques identifiées

Les problématiques identifiées durant cet état de l'art sont doubles et seront le fil conducteur de cette thèse :

— Quelle est l'influence des paramètres par défaut d'une pile QUIC sur l'équité ?

— Comment améliorer l'évaluation de l'équité entre les mécanismes de contrôle de congestion et les protocoles de transport différents ?

La première question est basée sur l'impact du nouveau protocole QUIC sur le réseau d'un point de vue de l'équité. Nous nous basons sur une solution *open source* QUIC-GO [23] comme le ferait un fournisseur de contenus et évaluons l'impact de différents paramètres sur l'équité. Cette étude permet d'identifier les différents paramètres facilement modifiables et de quantifier leurs impacts sur l'équité.

Enfin, la deuxième question soulève la question de l'équité lorsque les nouveaux algorithmes de contrôle de congestion cohabitent sur le réseau. Afin de s'assurer de la bonne répartition des ressources entre eux, nous évaluons l'équité des algorithmes suivants : RENO, BBR et PCC. C'est pourquoi nous voulons définir une procédure de tests impartiale afin de pouvoir évaluer l'équité dans différentes situations de réseau.

Chapitre 3

Plateforme de simulation de réseau fixe et mobile

Contents

Dans ce manuscrit de thèse, nous nous focalisons sur l'évaluation de l'équité dans diverses situations réseau. Afin de pouvoir tester les nouveaux algorithmes de contrôle de congestion et les protocoles en conditions réelles, il est indispensable de se reposer sur des plateformes de tests qui peuvent prendre plusieurs formes : soit sur un réseau réel, soit en laboratoire.

Plateforme de tests réels : Aujourd'hui, il existe plusieurs solutions pour tester les performances des algorithmes de contrôle de congestion et des protocoles. Par exemple, Pantheon [44], Planet-Lab [56] et Emulab [57] sont des exemples de plateformes de tests sur des réseaux réels. Elles permettent de réaliser des tests sur des connexions locales ou sur des connexions trans-continentales. Leurs principaux objectifs sont de mesurer les performances d'un flux à travers le réseau dans des conditions réelles d'utilisation.

Plateforme de tests en laboratoire : Il existe deux types de plateformes de tests en laboratoire : les simulateurs tels que *ns-3* qui permettent d'analyser plusieurs flux réseau et des plateformes de tests qui reproduisent des conditions du réseau internet avec des implémentations réelles telles qu'on pourrait les obtenir sur des équipements réseaux.

En ce qui concerne les plateformes de tests réels, nous pouvons prendre l'exemple de Pantheon [44]. Les auteurs énumèrent plusieurs scénarios de tests à réaliser pour faire varier les conditions réseau. Mais ceux-ci dépendent essentiellement de la charge du réseau internet. Il est alors difficile de réaliser les tests dans toutes les configurations possibles. De plus, dans ces différentes solutions, le réseau est considéré comme une boite noire, où il est difficile

d'obtenir des informations permettant d'analyser les résultats. En effet, à partir des captures réalisées sur un réseau réel, nous pouvons déduire des paramètres réseau tels que le taux de perte, le RTT à partir des informations obtenues sur les clients et les serveurs de la plateforme. Tandis que pour déterminer la charge du réseau comme le remplissage des *buffer*s ou les caractéristiques des flux en concurrence, ceci reste extrêmement complexe.

Dans cette thèse, nous avons fait le choix de travailler sur une plateforme de laboratoire de type plateforme de tests. Cette solution permet de se rapprocher le plus possible des situations réelles de l'Internet sans avoir toute la complexité d'interprétation des résultats d'une plateforme de tests réels. En effet, nous avons la possibilité de contrôler l'intégralité des paramètres réseau (le taux de perte, la latence, la taille des *buffer*s ou encore les types de *queuing policies*) pour pouvoir reproduire au mieux l'ensemble des situations réseau dans lesquelles les utilisateurs pourraient se retrouver. De plus, une plateforme de ce type, permet de tester des implémentations réelles des protocoles de transport afin d'analyser leurs comportements comme s'ils étaient déployés sur Internet. Enfin, cette plateforme nous permet aussi une reproductibilité des tests d'équité tout en obtenant de multiples informations sur les flux telles que le calcul du taux de perte, la mesure des remplissages des buffers en temps réel, et le contrôle de l'intégralité des flux qui circulent pour nous aider dans l'analyse des résultats obtenus.

Dans la suite de ce chapitre, nous présentons une plateforme de tests en laboratoire basée sur des machines réelles permettant la simulation de conditions typiques des réseaux à débit constant (jusqu'à 1Gbit/s) et variable (réseau 4G). Elle permet l'émulation de plusieurs flux client-serveur dans diverses situations de réseau (par exemple l'utilisation d'un réseau à débit constant ou variable, différentes valeurs de taux de perte ou de latence, etc). Chacun de ces flux partage un même goulot d'étranglement afin de pouvoir mesurer l'équité entre eux. Mais cette plateforme peut être facilement adaptée à d'autres types de mesures qui nécessiteraient des études dans différentes situations réseau, par exemple pour des mesures de performances sur de nouvelles applications ou protocoles.

Dans ce chapitre, nous abordons pour commencer une présentation détaillée de la plateforme de simulation réseau permettant, dans notre cas, la réalisation de mesures de l'équité. Cette plateforme est présentée sous deux versions : la première permet uniquement d'analyser l'équité sur des réseaux à débit constant (elle a été utilisée pour réaliser les tests d'équité des algorithmes de contrôle de congestion, voir chapitre 6) puis la seconde version permet de mesurer l'équité soit sur des réseaux à débit constant soit sur des réseaux à débit variable (elle a été utilisée pour réaliser les tests d'équité entre les différents protocoles des transports, voir chapitre 5). Puis dans un second temps, nous étudierons les paramètres qui peuvent être déterminés à partir de la plateforme.

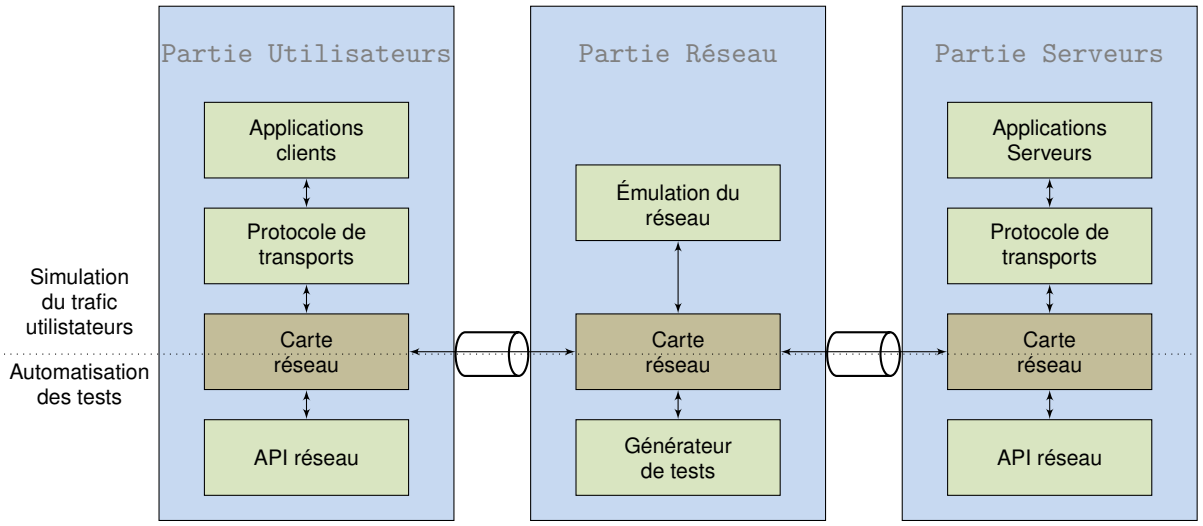


FIGURE 3.1 – Architecture logicielle de la partie serveur.

3.1 Architecture de la plateforme de tests

L'objectif de la plateforme de tests est d'analyser le comportement des algorithmes de contrôle de congestion et des protocoles de transport lorsqu'ils sont en concurrence dans différentes situations réseau et partagent un même goulot d'étranglement. Le développement de la plateforme a été réalisé de façon à pouvoir modifier l'ensemble des paramètres réseau possibles et reproduire des conditions réelles d'utilisation. La plateforme de tests a été déployée dans un réseau local afin de réaliser une mesure précise de l'équité en maîtrisant l'intégralité des flux qui y circulent. Le fait de contrôler l'ensemble des paramètres de la plateforme de tests permet alors la reproductibilité des tests.

L'architecture générale de la plateforme de tests est présentée sur la figure 3.1. L'implémentation de cette plateforme se décompose en trois parties distinctes. Chaque partie permet la simulation d'un élément du réseau ; la partie utilisateur permet d'émuler un ou plusieurs utilisateurs, la partie réseau reproduit un réseau à débit constant ou variable et la partie serveur implémente un fournisseur de contenus. Pour éviter tout problème de performances sur la plateforme, les diverses parties sont implémentées sur des ordinateurs différents. Chaque partie est détaillée dans les sections 3.1.1, 3.1.2 et 3.1.3.

L'ensemble de ces parties est décomposé en deux sous-parties : la première regroupant tous les éléments rendant possible le trafic des données client-serveur et la seconde permettant l'automatisation de l'ensemble des tests et la configuration du réseau.

— **Trafic de données des utilisateurs :** La première partie implémente les différents éléments nécessaires pour la réalisation des échanges de données sur la plateforme. Nous y retrouvons entre autres les implémentations des clients, les simulateurs réseau et les implémentations des serveurs.

Protocoles	Algorithmes de contrôle de congestion	Implémentions
QUIC	N/A	code QUIC-GO [23]
TCP	N/A	Linux Debian
UDP (PCC)	PCC	code PCC [58]

Tableau 3.1 – Implémentations des algorithmes de contrôle de congestion et protocoles dans le serveur.

— **Automatisation des tests :** La seconde partie est utilisée pour la configuration automatique de la plateforme dans le but de réaliser les tests. Pour cela, un générateur de tests est implémenté dans la partie réseau. Ce générateur s'appuie sur deux Application Programming Interface (API) réseau présentes à la fois dans la partie utilisateur et dans la partie serveur. Ces deux API réseaux permettent la configuration des machines pour paramétrer les différentes situations réseaux et la gestion de l'ensemble des flux sur le réseau.

En ce qui concerne le dimensionnement de la plateforme, elle est aujourd'hui opérationnelle dans notre laboratoire jusqu'à 80Mbit/s. Au-delà de ce débit, une amélioration des performances aux niveau des ordinateurs est nécessaire (RAM, CPU). L'ensemble des caractéristiques spécifiques à chaque ordinateur est présenté dans les implémentations de chaque partie (voir 3.1.1, 3.1.2, 3.1.3). Chaque ordinateur est équipé d'une carte réseau gigabit. Or, la plateforme de tests a été validée jusqu'à des débits de 80Mbit/sec. Dans ces conditions, les protocoles de niveau 1 et 2 du modèle OSI sont alors négligeables du fait que les cartes réseau sont sur-dimensionnées par rapport aux débits maximaux testés. Nous pouvons ainsi évaluer l'équité des protocoles et algorithmes de contrôle de congestion sans tenir compte des couches les plus basses.

3.1.1 Implémentation des clients

La partie utilisateur a pour but la simulation de différents clients réalisant des téléchargements de contenus. Ils sont implémentés sur un ordinateur avec un OS Debian basé sur le noyau 3.16. Cette machine possède un microprocesseur Intel Core 2 avec une RAM de 2Go et un disque dur de 250Go. La figure 3.2 représente l'architecture logicielle détaillée de la partie utilisateur.

Implémentation des protocoles sur la partie utilisateur

Dans cette sous-section, nous présentons l'implémentation des protocoles de transport (côté client) pouvant être utilisés pour réaliser des échanges de données entre un client et un serveur. L'ensemble des protocoles de transport implémentés dans la partie utilisateur est résumé dans le tableau 3.1.

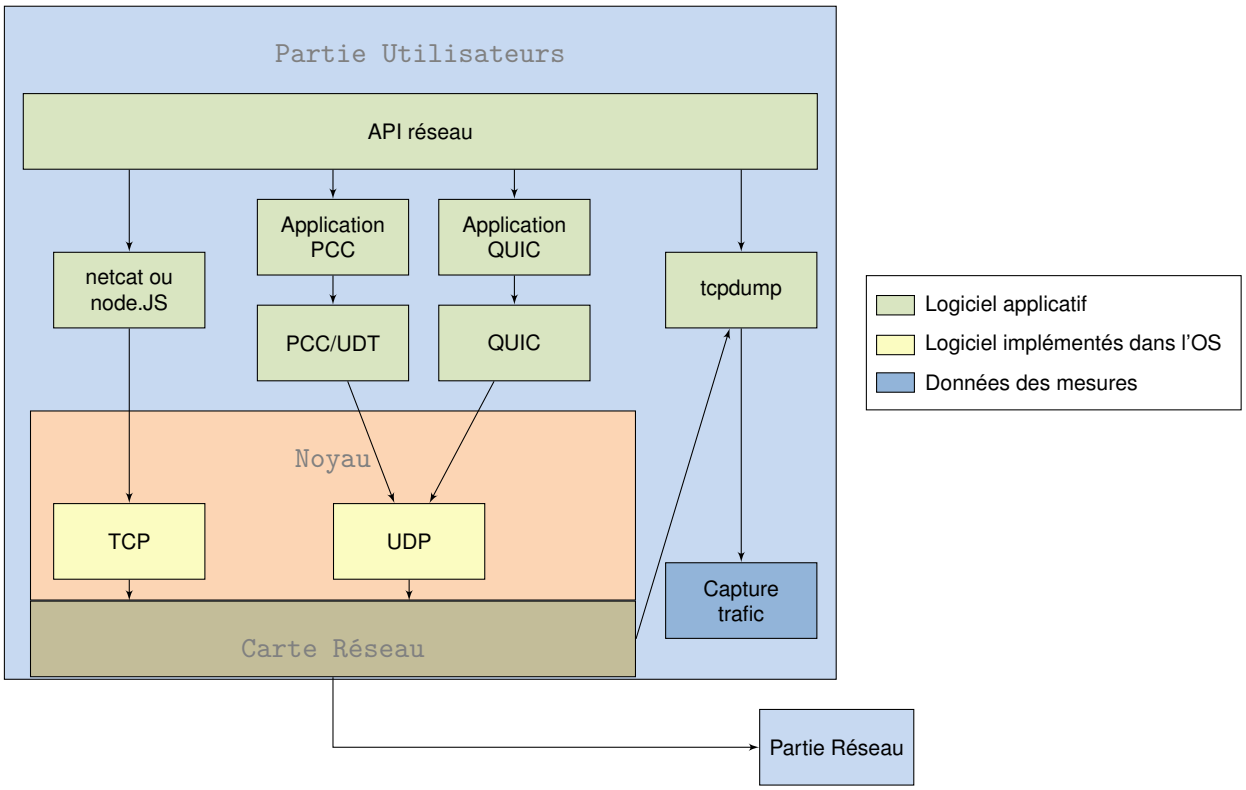


FIGURE 3.2 – Architecture logicielle de la partie utilisateur.

Avec les algorithmes de contrôle de congestion testés sur la plateforme, l'implémentation des algorithmes de contrôle de congestion est ignorée dans la partie utilisateur. En effet, les décisions d'évolution de débit sont exclusivement réalisées du côté serveur. Donc nous utilisons uniquement les implémentations des protocoles sans tenir compte des algorithmes de contrôle de congestion dans la partie cliente.

En ce qui concerne le protocole TCP, ce sont les programmes *netcat* et *node.JS* qui peuvent être utilisés ; tous deux reposent sur l'implémentation de TCP présente dans le noyau de l'OS. En ce qui concerne les protocoles QUIC et PCC, ils sont tous deux implémentés dans la partie *user-space* de l'ordinateur et reposent sur le programme QUIC-GO [23] et PCC [58]. Ces deux programmes s'appuient sur le protocole UDP présent dans le noyau de l'OS.

API réseau côté client

La partie utilisateur est aussi composée d'une API, nommée API réseau. Elle permet de configurer l'ensemble des paramètres nécessaires pour le bon fonctionnement des tests. Son objectif est donc double : *(i)* gérer le programme *tcpdump* qui permet de capturer les données reçues sur la carte réseau, c'est à dire l'ensemble des paquets que reçoit chaque client sur la carte réseau. Puis, dans un second temps, ces données sont traitées pour déterminer l'équité

perçue par les utilisateurs (voir section 3.2). *(ii)* Le deuxième objectif de l'API réseau est d'initier chacune des connexions à un moment précis pour évaluer différentes situations réseau. Ainsi un flux peut démarrer avant l'autre ou encore quasi simultanément.

3.1.2 Implémentation du réseau

La partie réseau permet de réaliser l'émulation de différentes situations réseau. Elle dispose d'une machine basée sur OS Ubuntu 16.04 utilisant un noyau 4.15. La machine possède un microprocesseur Intel Xenon, 8Go de RAM ainsi qu'un disque dur de 500Go. L'implémentation logicielle de la partie simulation réseau comporte deux versions : une première version qui permet uniquement la simulation d'un réseau à débit constant, présenté sur la figure 3.3 et une seconde version permettant de réaliser des tests sur des réseau à débit constant et à débit variable. Cette seconde version est détaillée dans la figure 3.5.

Version 1 : Réseau à débit constant

L'architecture de la première version du simulateur réseau à débit constant est présentée dans la figure 3.3.

L'objectif de la partie réseau est double : *(i)* permettre la simulation d'un réseau en émulant un réseau à accès fixe et *(ii)* gérer l'automatisation des tests à partir d'un générateur de tests. En ce qui concerne l'automatisation des tests, le générateur de tests est implémenté sur la partie réseau et a deux objectifs principaux : la définition et le paramétrage d'une situation réseau pour réaliser les tests d'équités et la configuration des simulateurs réseau.

Automatisation des tests : La définition et le paramétrage d'un scénario de test sont réalisés par le générateur de tests. Pour cela, il se base sur un fichier de configuration afin de choisir un paramétrage réseau parmi tous ceux disponibles. À partir des informations choisies, il configure la partie réseau en se basant sur le simulateur réseau qu'il possède, puis réalise le routage des paquets à travers les différents simulateurs. Enfin, il se repose sur deux API réseau (présentes sur les clients et serveurs) utilisées pour le démarrage des serveurs et des clients tout en réalisant l'ensemble des configurations nécessaires.

Simulateur de réseau : La configuration des paramètres réseau est basée sur le programme *tc* de contrôle du trafic Linux, qui permet de modifier la configuration du réseau. Dans notre cas, *tc* est utilisé pour définir une latence, un taux d'erreur et les caractéristiques du points de saturation (queuing policies et taille du *buffer*).

Différents queueing policies sont implémentées par défaut sous Linux et donc utilisables sur la plateforme : *(i)* First In First Out (FIFO) est l'implémentation par défaut du *buffer* : le premier paquet arrivé est le premier à sortir. Lorsque la mémoire du *buffer* est pleine, les paquets sont automatiquement supprimés. *(ii)* fair queuing controlled delay (fq_codel) est un algorithme du *buffer* qui vise à corriger les problèmes d'iniquité sur le réseau. Il émule un *buffer* FIFO dédié pour chaque flux qui est identifié par adresse source, adresse destination, port source, port

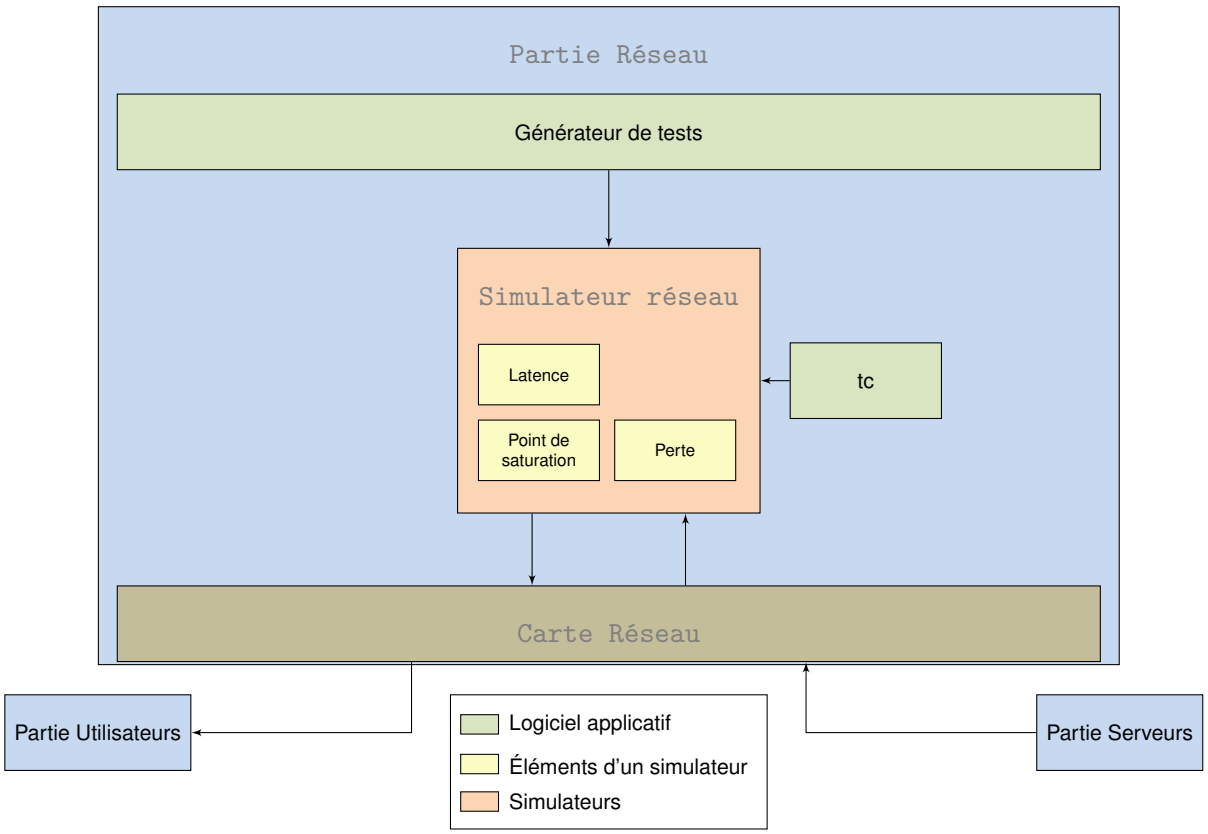


FIGURE 3.3 – Architecture logicielle de la partie réseau à débit constant.

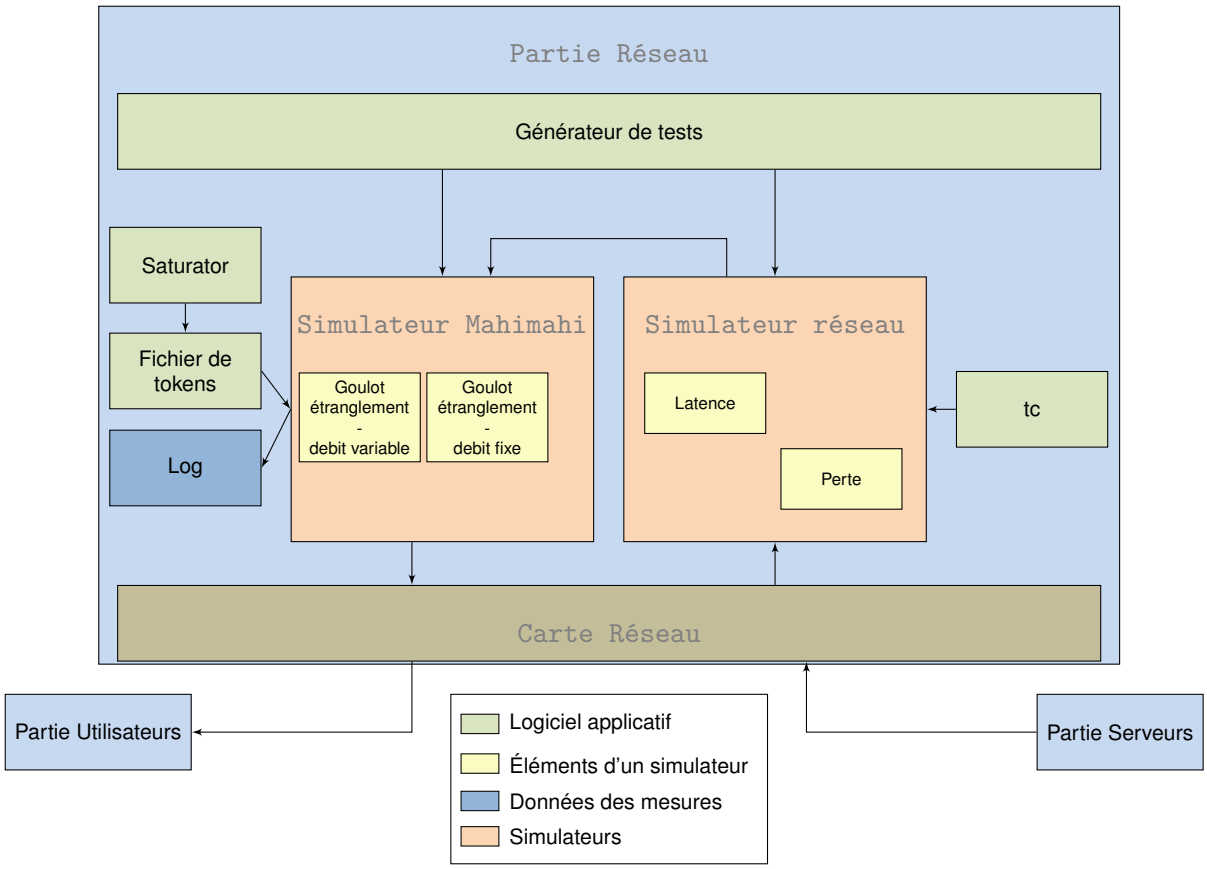


FIGURE 3.4 – Architecture logicielle de la partie réseau à débit constant et variable.

destination et type de protocole. L'algorithme Controlled Delay (CoDel) présent dans fq_codel évite l'accumulation de paquets dans les *buffer*s FIFO et élimine l'iniquité entre les différents flux. (iii) Random Early Detection (RED) supprime des paquets avec une probabilité qui augmente progressivement à mesure que le *buffer* se remplit pour limiter le débit des flux les plus gourmand en bande passante. Les valeurs de taux de pertes, la latence et la taille de buffers sont facilement paramétrables dans la plateforme.

Version 2 : Réseau à débit constant et variable

L'architecture de la deuxième version du simulateur est visible sur la figure 3.4. Ce simulateur permet de tester les différents algorithmes de contrôle de congestion et protocoles sur des réseaux à débit constant et à débit variable.

Le fonctionnement de cette version du simulateur réseau reste identique à la première version. On y retrouve entre autres le générateur de tests qui a les mêmes fonctionnalités que dans la première version ainsi que le simulateur de réseau qui simule une latence et une perte pour l'ensemble des flux. Ce simulateur de réseau ne simule plus directement le point de

saturation car celui-ci est présent dans le simulateur Mahimahi [50] qui permet une supervision du remplissage du *buffer* en temps réel. L'objectif du premier simulateur est de reproduire différentes situations réseau en faisant varier le taux de perte et de latence. Une fois que les paquets ont traversé ce simulateur, ils sont acheminés vers le deuxième.

Simulateur Mahimahi

Le second simulateur, nommé simulateur Mahimahi, est capable de simuler des réseaux à débit constant (par exemple une liaison ADSL) ou un réseau à débit variable (par exemple un réseau mobile 4G). Il est basé sur un programme open source nommé Mahimahi [50, 51]. La configuration de ce simulateur est aussi réalisée par le générateur de tests. Ainsi ce dernier démarre une instance de *link-shell* qui permet l'émulation d'un réseau d'accès en fonction des paramètres qui lui sont communiqués. Pour déterminer la caractéristique du débit, le *link shell* se base sur un fichier de *token*s. Ce fichier est composé d'informations temporelles permettant d'indiquer les opportunités de transmission. Une opportunité de transmission se traduit par la possibilité d'envoyer un paquet de 1500 octets en attente de transmission. Pour la caractérisation d'un réseau à débit constant, le fichier de *token*s autorise l'envoi de paquets à des intervalles de temps réguliers, tandis que pour la caractérisation d'un réseau à débit variable, le fichier de *token*s indique plus ou moins rapidement des opportunités de transmission. Dans ce dernier cas, les opportunités de transmission sont déterminées par le programme *open source* nommé *saturator* [59]. C'est un programme open source permettant de caractériser une cellule radio en opportunité de transmission pour Mahimahi.

Évolution du débit du goulot d'étranglement à débit variable : Le *link-shell* de *Mahimahi* utilise des fichiers de *token*s pour la livraison de paquets afin d'émuler les caractéristiques de liaisons montantes et descendantes d'un réseau Long Term Evolution (LTE) en termes de possibilités de transmission. Chaque ligne d'un fichier de *token*s représente le moment (à la milliseconde près) qui autorise la transmission d'un paquet de 1500 octets.

Les fichiers de *token*s sont acquis en utilisant l'outil saturateur [59] qui sature le chemin réseau entre le client saturateur et le serveur saturateur dans les directions de liaison montante et descendante et les temps de livraison des paquets. L'algorithme de l'outil assure que la file d'attente du goulot d'étranglement est suffisamment remplie. Chaque opportunité de transmission de paquets de 1500 octets fournie par le réseau est alors utilisée (et non gaspillée) car la file d'attente n'est jamais vide. Les horodatages portés par les paquets permettent à l'outil de garder la trace de chaque milliseconde où une ou plusieurs opportunités de transmission se sont produites.

Pour les tests rapportés dans le présent manuscrit, nous avons capturé une trace de livraison de paquets en conditions statiques (le *smartphone* LTE hébergeant le client était immobile), avec une bonne qualité de liaison radio (une moyenne Channel Quality Indicator (CQI) de 10

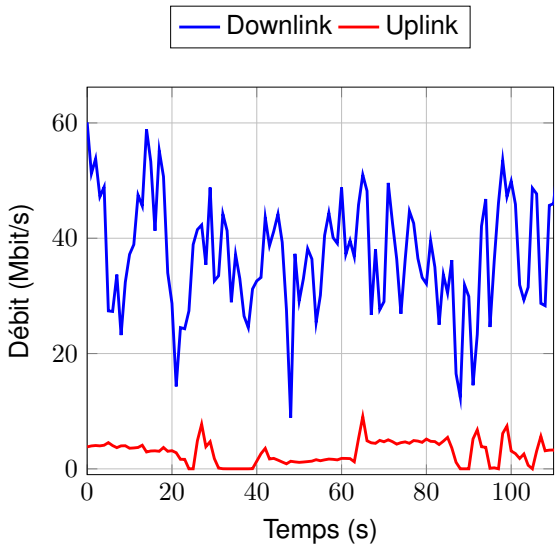


FIGURE 3.5 – Trace de débit de Mahimahi.

sur 15). Les variations de débit dans la trace de livraison des paquets apparaissent sur la figure 3.5. Les débits moyens en liaison descendante et en liaison montante sont respectivement de 34Mbit/s et 4 Mbit/s.

Supervision des informations du *buffer* : Mahimahi permet aussi une supervision en temps réel du remplissage du *buffer*, ainsi que de tous les évènements tels que l'arrivée d'un paquet sur l'interface mahimahi, l'ajout d'un paquet au *buffer*, les opportunités de transmission et enfin l'envoi d'un paquet. L'ensemble de ces informations est sauvegardé dans les fichiers de log.

3.1.3 Implémentation des serveurs

La partie serveur est implémentée sur un ordinateur basé sur le système d'exploitation Ubuntu Server 16.04 avec le noyau 4.4. Cette machine possède une RAM de 3Go, un processeur Intel Xenon et un disque dur d'une capacité de 68Go. L'ensemble des éléments logiciels de la partie serveur est visible sur la figure 3.6.

Protocoles et algorithmes de contrôle de congestion implémentés sur la partie serveur

Plusieurs protocoles et algorithmes de contrôle de congestion sont implémentés sur la partie serveur de la plateforme de tests et sont récapitulés dans le tableau 3.2.

Actuellement, la plateforme de tests implémente sur son serveur le protocole TCP avec trois algorithmes de contrôle de congestion différents, nommé CUBIC, RENO et BBR. L'implémentation du protocole TCP ainsi que les trois algorithmes de contrôle de congestion sont di-

Protocoles	Algorithmes de contrôle de congestion	Implémentions
QUIC	CUBIC	code QUIC-GO [23]
TCP	CUBIC	Linux Ubuntu
TCP	RENO	Linux Ubuntu
TCP	BBR	Linux Ubuntu
UDP (PCC)	PCC	code PCC [58]

Tableau 3.2 – Implémentations des algorithmes de contrôle de congestion et protocoles dans le serveur.

rectement présents dans le noyau de le système d'exploitation. QUIC et PCC sont deux autres protocoles de transport installés sur la plateforme de tests. Ces deux protocoles sont implémentés dans la partie *user-space* avec les programmes suivants : QUIC-GO [23] et PCC [58].

L'API réseau dans la partie serveur

L'API réseau permet de configurer correctement la partie serveur en fonction des paramètres reçus par le générateur de tests. Les principales fonctions de l'API sont : *(i)* la configuration des algorithmes de contrôle de congestion dans le noyau du système d'exploitation lorsque TCP est utilisé, *(ii)* la capture du trafic réseau sur l'interface réseau à la sortie du serveur et *(iii)* la gestion des serveurs pour l'envoi des données aux clients avec la création des contenus à transmettre.

La première fonctionnalité de l'API réseau est la configuration des algorithmes de contrôle de congestion utilisant le protocole de transport TCP. Cette fonctionnalité de l'API modifie directement la configuration du noyau de l'OS de façon à obtenir l'algorithme de contrôle de congestion souhaité sur le protocole TCP. Ces différents algorithmes (nommés RENO, BBR et CUBIC) sont déjà implémentés dans le noyau du système d'exploitation et ne nécessitent aucune installation supplémentaire. Les autres protocoles de transports comme QUIC et PCC sont installés dans l'*user-space* et ne nécessitent pas de configuration au préalable. En effet, les implémentations de PCC et QUIC ne possèdent qu'un seul algorithme de contrôle de congestion et sont directement ré-implémentés dans les programmes.

La seconde fonctionnalité de l'API est de gérer le programme *"tcpdump"* afin de réaliser les captures de trafic réseau au niveau de la carte réseau. Comme sur la partie utilisateur, ces captures sont sauvegardées sur le disque dur et seront analysées dans un second temps pour observer le trafic de chaque utilisateur tel qu'ils ont été émis sur le serveur.

Enfin, la dernière fonctionnalité de l'API est le démarrage des différents serveurs nécessaires pour réaliser les échanges de données entre le client et le serveur. Pour cela, l'API réseau génère l'ensemble des fichiers de données nécessaires aux serveurs pour simuler le téléchargement de contenus.

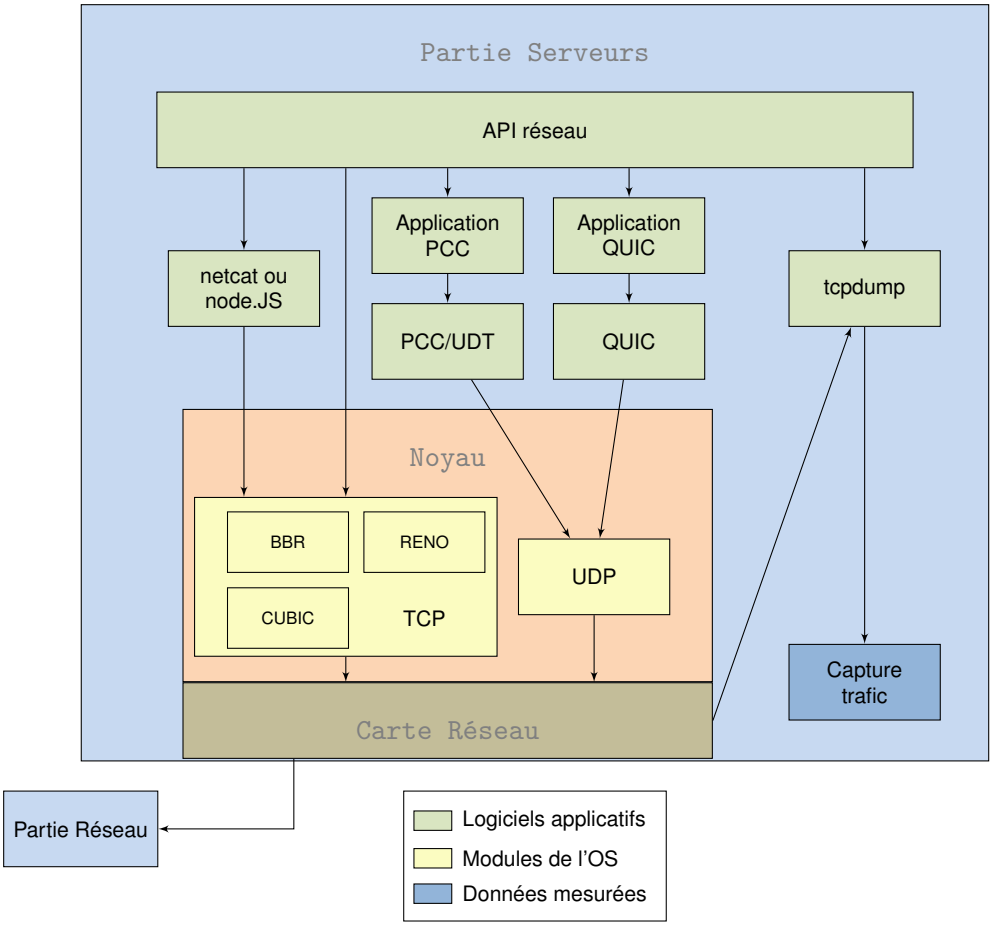


FIGURE 3.6 – Architecture logicielle de la partie serveur.

3.2 Mesures possible à partir de la plateforme

Un programme *open source* est mise à disposition sur le lien suivant (lien) et permet de mesurer différents paramètres réseau à partir des informations de la plateforme. Ce programme se base sur deux types de données pour mesurer les paramètres réseau. Le premier consiste à utiliser les fichiers capturant le trafic réseau avec *tcpdump* pour réaliser des mesures d'équité ou de performance avec le débit des différents utilisateurs. Le second est d'utiliser les informations de supervision du remplissage du buffer (point de saturation) à partir des fichiers de log de Mahimahi.

Mesure réalisable à partir du traitement des données des logs de Mahimahi : Ces traitements ne peuvent être fait que dans la seconde version de la partie réseau, car le logiciel Mahimahi n'est pas implémenté dans la première. Les données des fichiers de log récapitulent toutes les actions qui se sont produites dans le *buffer* durant l'exécution des tests : à savoir l'arrivée d'un nouveau paquet, la mise en mémoire d'un paquet dans le *buffer*, les opportunités de transmission, et l'envoi d'un paquet. Chacune de ces informations représente une nouvelle ligne dans le fichier de log. Chaque ligne comporte trois types d'informations : le "timestamp" qui indique le moment où s'est produit l'évènement, l'action réalisée et le flux concerné. A partir de ces informations, le programme de traitement des données fournit plusieurs statistiques comme :

— *Le taux de perte du buffer :* Le taux de perte du *buffer* correspond au nombre de paquets supprimés suite à un débordement de ce dernier. En d'autres termes, cela correspond au dépassement de la capacité du *buffer* causé par un nombre trop important de paquets arrivés dans un intervalle de temps trop court. Ainsi pour déterminer ce taux de perte, l'algorithme calcule la valeur de remplissage du *buffer* à chaque arrivée de nouveaux paquets. À partir de cette valeur calculée, l'algorithme détermine si le paquet peut être mémorisé ou non dans le *buffer*. En effet, la perte se produit si et uniquement si cette valeur est supérieure à la capacité maximale du *buffer*. L'algorithme est alors en mesure de déterminer le nombre de paquets supprimés et d'établir avec précision la valeur du taux de perte dans le *buffer*.

— *le taux de remplissage de la mémoire tampon :* Le taux de remplissage du *buffer* permet de déterminer le pourcentage du temps où celui-ci est rempli à plus de $x\%$. L'algorithme se base donc sur le calcul du taux de perte pour connaître son taux de remplissage. Par défaut la valeur de x est fixée à 70%.

— *le pourcentage d'utilisation des opportunités de transmission :* Le pourcentage d'utilisation des opportunités de transmission correspond au nombre d'opportunités utilisées par rapport à celles disponibles dans le fichier de *token*s de Mahimahi. Pour fixer cette valeur, nous déterminons le rapport entre le nombre de paquets émis par rapport aux nombres d'opportunités de transmission autorisées par le fichier de *token*s.

D'autre paramètres peuvent encore être analysé comme la répartition des flux dans le *buffer* (combien paquets de chaque flux est présent dans le buffer) pour déterminer quel flux participe le plus à la congestion dans le réseau, mais cette axe n'a pas été étudié durant cette thèse.

L'ensemble de ces paramètres est calculé à partir d'algorithmes de traitement des données qui ont été développés avec le langage de programmation nodeJS. Ce programme permet de calculer l'intégralité de ces valeurs pour chacun des tests et les sauvegarder dans un fichier "CSV" pour pouvoir les analyser ou les représenter sous forme de graphique.

Traitement des données issues de tcpdump : Cette sous-section détaille le traitement des fichiers de capture réalisés par *tcpdump* au niveau de la partie serveurs et utilisateurs. Les fichiers sont capturés sous la forme de fichiers de type *wireshark* (.pcap). Or pour calculer un indice d'équité, seul le débit nous intéresse. Donc la première étape est de pouvoir convertir des fichiers de type *wireshark* en fichiers de données représentant le débit de chacun des flux. Pour cela, le module *dpkt* de Python est utilisé afin de réaliser cette conversion et obtenir le nombre de bits transmis par tranche de $100ms$. Par la suite, les fichiers de débit obtenus sont traités avec une métrique d'équité. L'ensemble de cette procédure est détaillé dans le chapitre 4.

3.2.1 Bilan sur la plateforme de tests

Dans ce chapitre, nous avons présenté la plateforme de tests réalisant l'intégralité des mesures d'équité présentes dans cette thèse. Cette plateforme regroupe trois parties distinctes qui sont : la partie utilisateur, la partie réseau et la partie serveur. Elle est ainsi capable de générer le trafic sur différents protocoles de transport et algorithmes de contrôle de congestion dans un environnement complètement contrôlé. Si un jour, des tests sont souhaités sur Internet, il faudra adapter le routage des paquets au niveau de la partie réseau.

A partir des informations reçues de la plateforme lors des tests, nous pouvons caractériser plusieurs paramètres. Cette plateforme nous permet de caractériser le *buffer* du goulot d'étranglement et en déterminer plusieurs paramètres comme : le taux de perte ou le taux de remplissage de ce dernier. Enfin, une seconde partie présente la conversion des fichier traces réseau en fichier "CSV" pour obtenir le débit de chaque flux sur le réseau. Ces fichiers pourront par la suite être traités pour faire des calculs d'équité réseau.

Chapitre 4

Évaluation de l'équité : Métrique

Contents

Chaque service utilisé sur le réseau internet a des besoins spécifiques en terme de débit pour maximiser la satisfaction de la clientèle. Par exemple dans [5, 60], les auteurs ont établi une classification de plusieurs services proposés tout en caractérisant le débit souhaité pour optimiser les performances :

— **Débit constant au cours du temps :** application ayant des besoins temps réel

— **Importance du début de la session :** visionnage de vidéos en streaming, téléchargement de pages web.

— **Faibles valeurs de latence et de taux de pertes** : véhicules autonomes, application critique de la réalité virtuelle (médecine)

— **Non-critique** : Internet of Things (IoT), téléchargement de logiciels, mises à jour des OS.

Pour caractériser le partage de la bande passante, c'est-à-dire le débit d'un flux comparé aux autres, des mesures d'équité sont utilisées. Aujourd'hui, de nombreuses métriques existent pour réaliser ce type de mesures (voir section 2.3.3). C'est pourquoi nous avons voulu créer une méthode impartiale permettant de mesurer l'équité sur la totalité de la session d'un utilisateur et qui de plus tiendra compte des caractéristiques des différents services. Durant cette thèse nous nous sommes uniquement concentré sur la partie téléchargement de contenus entre un

serveur et un utilisateur. Des études plus approfondie devront être réalisé pour mesurer l'équité entre les différentes classes de services et faire le lien entre l'équité et la Quality of Experience (QoE) et la QoS.

Ce chapitre est organisé comme suit : dans un premier temps, nous argumentons sur la nécessité d'évaluer l'équité au cours du temps. Puis, nous proposons une première méthode pour mesurer l'équité sur une session entière (nommée Session Fairness Assessment (SFA)). Ensuite dans une dernière partie, nous présentons une modification de la première métrique qui s'appelle Weighted Session Fairness Assessment (WSFA). Cette deuxième méthode permet d'accentuer l'importance d'un moment de la session (dans notre cas le début de la session). Ainsi nous pouvons affiner la mesure d'équité pour le visionnage de vidéos ou le téléchargement de pages web.

4.1 Le besoin d'évaluer l'équité durant toute une session

Comme étudié dans la partie 2.3.2, l'ensemble des métriques présentes dans l'état de l'art n'évalue l'équité que sur un instant précis de la session. Or les utilisations des services en ligne sur le réseau ne se limitent plus au téléchargement de fichiers. Au lieu de cela, les interactions avec les utilisateurs exigent des échanges réseau plus complexes pendant toute la durée de la session. Or une baisse soudaine des performances sur une session est alors possible, soit par des évènements réseau comme détaillés dans la partie 2.4.2 ou alors par des comportements plus néfastes comme la personnalisation ou la suppression de l'algorithme de contrôle de congestion. Ces baisses de performance pendant une session peuvent avoir des répercussions négatives pour l'utilisateur (par exemple des interruptions de service, des phases de rebuffering des vidéos ou des comportements répréhensibles). Pendant ce temps, les fournisseurs de services sont tentés d'obtenir plus de ressources à un moment donné d'une session parce qu'une augmentation soudaine de la performance peut améliorer considérablement la qualité du réseau perçue par les utilisateurs (par exemple un délai de démarrage de la vidéo plus rapide ou encore l'accélération du téléchargement de ressources web). Les opérateurs de réseau visent à garantir un accès équitable à l'ensemble des ressources disponibles du réseau par rapport à tous les flux concurrents. Ils doivent alors gérer ces exigences contradictoires.

Dans ce contexte, les opérateurs de réseau sont intéressés par l'évaluation de l'équité pendant toute la période d'une session en concurrence avec d'autres flux (point qui est déjà réalisé lors de la supervision réseau). C'est pourquoi nous souhaitons introduire une notion d'équité durant toute la session d'un utilisateur. L'idée de "l'équité au niveau de la session" est que chaque flux bénéficie d'un partage équitable des ressources pendant la durée totale du téléchargement. En particulier, lorsqu'on observe une session précise, on dit que le comportement du réseau est juste au niveau de la session si ce flux réseau est équitable avec d'autres flux concurrents lorsque tous les moments de la session sont analysés. Notamment, une session qui obtient beaucoup plus de ressources que d'autres flux pour certains créneaux, mais com-

pense en ayant moins de ressources pour d'autres créneaux n'est pas optimale puisque ce comportement peut compromettre l'accès aux services pour les flux concurrents. Cependant l'équité est respectée sur la période de temps. Si nous voulions prendre en compte ce point, une métrique de stabilité de débit serait à prendre en considération.

4.2 Session Fairness Assessment

4.2.1 Définition de la méthode de mesure de l'équité sur une session

En principe, l'équité consiste à vérifier l'allocation des ressources du réseau tout en maximisant la satisfaction des utilisateurs en fonction de leurs besoins. Dans les protocoles de transport, l'équité est généralement considérée en comparant le débit affecté à chacun des flux. Dans cette sous-section, nous présentons une nouvelle méthode pour réaliser une mesure de l'équité, nommée Session Fairness Assessment (SFA), qui sera utilisée pour quantifier l'équité entre les flux de données simultanés au niveau d'un goulot d'étranglement.

Plusieurs solutions pour mesurer l'équité ont été proposées dans la littérature (voir section 2.3.3), le plus connu et le plus utilisé étant l'indice de Jain [42] ; en supposant qu'à un goulot d'étranglement donné, il y a la présence de n flux, l'indice de Jain est calculé comme suit :

$$J_{x_1,x_2,....,x_n} = \frac{(\sum_{k=1}^{n} x_k)^2}{n \sum_{k=1}^{n} x_k^2} \tag{4.1}$$

où x_i est le rapport entre le débit observé et le débit " juste " d'un flux i selon l'équité *max-min fairness* (voir section 2.3.2). La valeur de l'indice de Jain varie alors entre $1/n$ dans le pire cas et 1 dans le meilleur cas.

Pour intégrer cet indice d'équité sur une période de temps, un moyen simple consiste à obtenir un indicateur global "d'équité de la session" à partir des valeurs successives de l'indice de Jain calculées sur de petits intervalles de temps. Cependant, le calcul de la moyenne des indices de Jain successifs ne représente pas une indication fiable de l'équité de la session, car ce qui se produit à des intervalles successifs peut minimiser à tort l'équité en cas de variation de débit.

Compte tenu de ce qui précède, lorsqu'on évalue l'équité entre deux flux ($n = 2$) qui partagent un goulot d'étranglement donné, il est utile de définir le flux " dominant ". Il est déterminé à chaque intervalle de mesures et définit le flux qui obtient un débit supérieur par rapport à l'autre dans l'intervalle de temps considéré. Nous avons donc réparti le temps de la compé-

tition entre deux flux A et B en S créneaux de 100 millisecondes. Pour chaque intervalle de temps i, l'indice d'équité est donné par J_i et la dominance entre A et B pendant le créneau de temps i est caractérisé par d_i d'où

$$d_i = \begin{cases} -1, & \text{si } A \text{ est dominant} \\ 1, & \text{si } B \text{ est dominant} \end{cases} \tag{4.2}$$

Ensuite, une mesure globale de l'équité de session F est définie en considérant $J = \{J_1, ..., J_S\}$ et $d = \{d_1, ..., d_S\}$ comme suit :

$$F = \frac{\sum_{i=1}^{S} d_i * (1 - J_i)}{S} \tag{4.3}$$

Les valeurs de F sont comprises dans l'intervalle suivant : $[-0.5, 0.5]$. Le signe F détermine le flux "dominant" à l'échelle de la session. Lorsque B (respectivement A) est globalement dominant, F est positif (respectivement négatif). La valeur F indique également une équité au niveau de la session, c'est-à-dire que lorsque $|F|$ est proche de 0, l'équité de la session est atteinte alors qu'une injustice globale est caractérisée par des valeurs plus élevées de $|F|$.

4.2.2 Application de la métrique sur des exemples

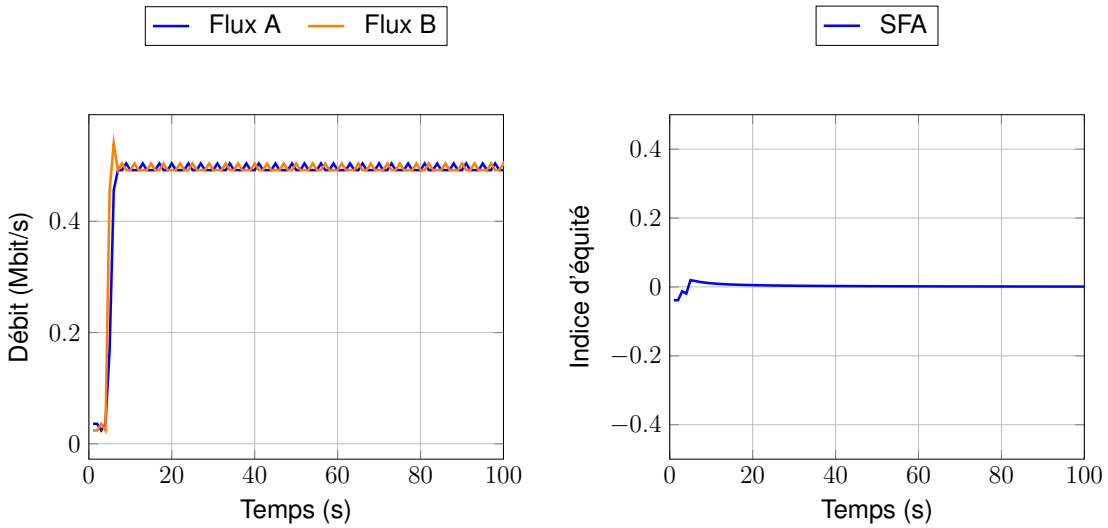
Pour illustrer le comportement de SFA, nous prenons trois situations réseau différentes avec deux flux en compétition partageant un même goulot d'étranglement avec un débit constant de 1Mbit/s. Le premier cas étudié est un partage de la bande passante équitable entre les deux flux ; le second représente un partage inéquitable entre les deux flux et le dernier est une alternance de dominance entre les flux (chaque flux obtient plus de débit à tour de rôle). Pour chaque compétition entre les deux flux, nous représentons deux courbes : l'une pour les débits de données mesurés pour les deux flux et l'autre pour l'évolution de l'indice d'équité avec la méthode SFA.

Cas équitable

Le premier cas correspond au cas équitable entre les deux flux, c'est-à-dire que chacun possède un débit proche de 500kbit/s. La mesure de l'équité est évaluée avec la méthode SFA et indique une valeur d'équité proche de 0 durant toute la période de concurrence. Ce résultat indique que les flux sont équitables entre eux.

Cas inéquitable

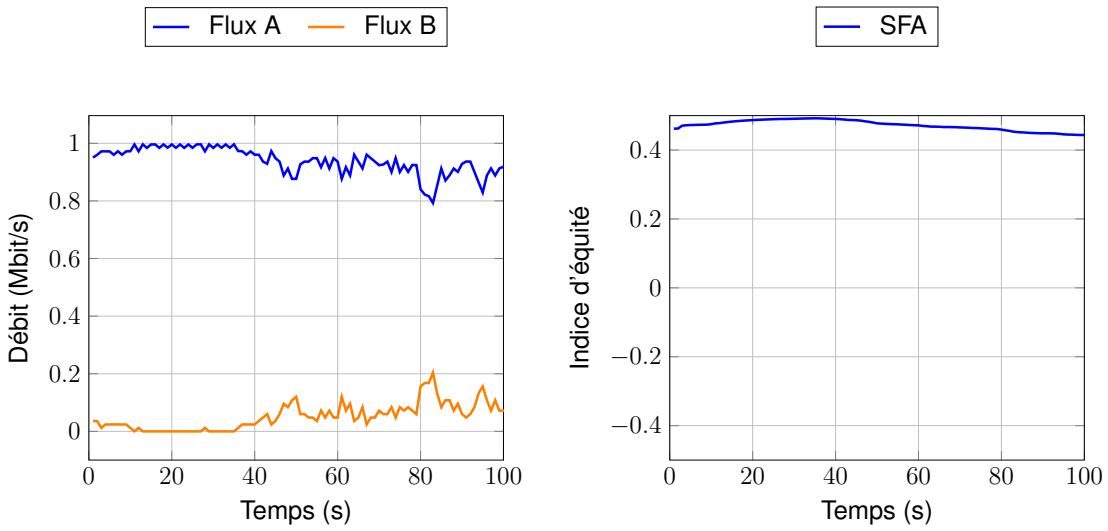
Le second cas est cette fois-ci inéquitable pendant toute la période de concurrence, c'est-à-dire que le flux A à un débit proche de 1Mbit/s et le flux B possède très peu de débit (quelque kbit/s). Ainsi, le flux A utilise presque l'intégralité de la bande passante au détriment du second.



(a) Évolution du débit

(b) Résultat de l'indice d'équité avec la méthode SFA

FIGURE 4.1 – Exemple de mesure d'équité pour un cas équitable.

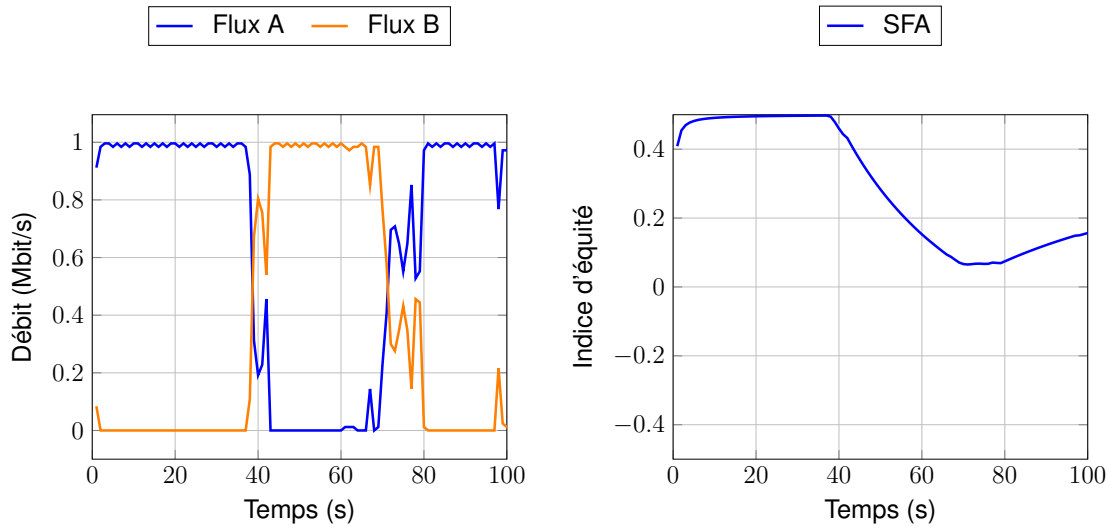


(a) Évolution du débit

(b) Résultat de l'indice d'équité avec la méthode SFA

FIGURE 4.2 – Exemple de mesure d'équité pour un cas inéquitable.

Lorsque nous appliquons la méthode SFA sur ce type de trafic, la valeur de l'indice d'équité est proche de 0.5 durant toute la période de concurrence, ce qui traduit une inéquité importante au cours du temps entre les flux. La valeur d'équité est positive au cours du temps, ce qui indique la dominance du flux A.



(a) Évolution du débit

(b) Résultat de l'indice d'équité avec la méthode SFA

FIGURE 4.3 – Exemple de mesure d'équité pour un cas d'alternance de domination.

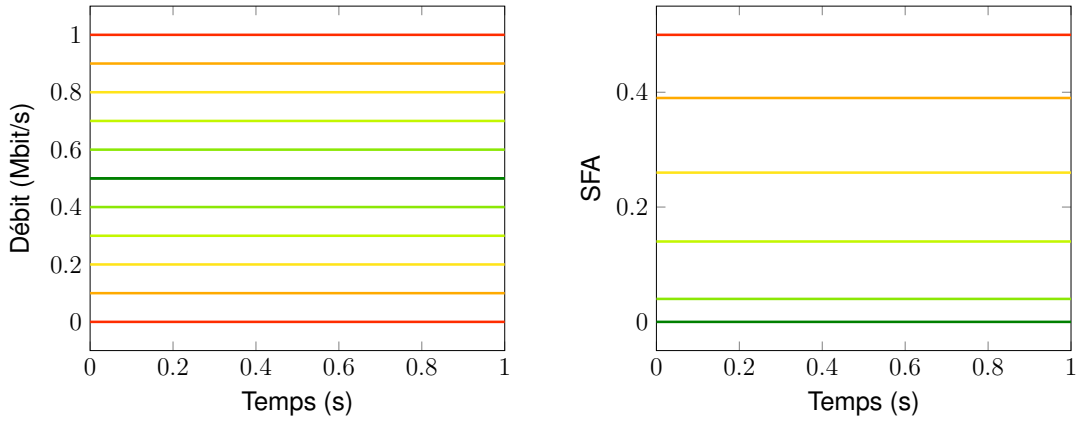
Cas d'alternance de domination

Le dernier cas est une alternance de la dominance des flux, c'est-à-dire que chaque flux obtient plus de débit pendant une période de temps. Dans notre exemple, trois périodes d'alternance sont visibles : *(i)* la première partie, comprise entre 0s et 40s, est une période très inéquitable en faveur du flux A. C'est donc naturellement que la méthode SFA indique une dominance nette du premier flux avec une valeur proche de 0.5. *(ii)* Dans la seconde période (de 40 à 70s), c'est une inversion de la dominance qui est notable, en effet, c'est le flux B qui obtient un débit largement supérieur à celui de l'autre. L'indice d'équité avec la méthode SFA se rapproche alors de la valeur de 0, car une compensation est réalisée du faîte du changement de dominance des flux. *(iii)* Enfin dans la dernière partie (après 70s), le flux A domine de nouveau. Donc une nouvelle augmentation de l'indice d'équité vers la valeur de 0.5 est à nouveau visible pendant le reste de la période de la session.

4.2.3 Caractérisation de l'évolution de l'indice d'équité avec la méthode SFA

Maintenant que l'indice d'équité a été défini au sein de la méthode de mesure SFA, nous réalisons une étude sur quelques exemples afin de pouvoir faire la relation entre les différentes valeurs d'équités et les écarts de débit entre deux flux. Les résultats de cette comparaison sont visibles sur la figure 4.4.

La première courbe 4.4a montre six combinaisons de deux flux à débit constant. Chaque paire de flux concurrencés est distinguable par une couleur. La paire de flux la plus inéquitable est représenté par la teinte la plus rouge, les flux partiellement équitables par les couleurs



(a) Évolution du débit

(b) Résultat de l'indice d'équité avec la méthode SFA

FIGURE 4.4 – Caractérisation du débit et de l'indice d'équité avec la méthode SFA.

jaunes et les plus équitables par les couleurs vertes. La méthode SFA est alors appliquée à l'ensemble de ces paires de flux et les résultats sont présentés dans la figure 4.4b. Les couleurs de chaque courbe d'équité représentent une combinaison de flux ayant la même couleur que les courbes de débit présentes dans la figure 4.4a. Par exemple, les deux flux orange sont en concurrence avec des débits de 0.1Mbit/s et 0.9Mbit/s avec un indice d'équité constant proche de 0.4.

La méthode SFA se base sur l'indice de Jain pour déterminer l'équité entre deux flux. Or la décroissance de l'indice de Jain ne se fait pas de façon linéaire. En effet, lorsque deux flux possèdent des débits proches des valeurs équitables, ils seront beaucoup moins pénalisés que des flux réellement inéquitables. Cette propriété est également vraie dans la méthode SFA où nous constatons que l'écart d'équité est beaucoup plus faible dans les combinaisons de flux les plus équitables (couleurs vertes) que dans les combinaisons les moins équitables (couleurs jaune et rouge).

4.3 Weighted Session Fairness Assessment

4.3.1 Besoin de pondérer une partie de la période de concurrence

La première métrique, nommée SFA, présente une mesure d'équité durant l'ensemble de la session en considérant chaque moment à égale importance. Intuitivement en lisant la littérature sur ce sujet, des moments de la session ont plus d'importance que d'autres. Par exemple dans la partie ci-dessous nous nous intéressons au début de session qui est primordiale pour diminuer le temps d'affichage pour les utilisateurs lors du téléchargement d'un contenu. Mais

des analyses plus précisent seraient à réaliser pour faire le lien entre cette métrique et la QoE d'un utilisateur. Nous posons ici uniquement un nouveau concept pour mesurer l'équité, tout en essayant de prendre en compte le plus possible les besoins des utilisateurs.

Nous nous concentrons donc sur des études qui démontrent que la QoE d'un service est fortement influencée par la quantité de données reçues au début de la session. C'est généralement le cas des services vidéo en streaming et des pages web, qui représentent respectivement 73% et 18% du trafic internet global [61]. Pour le trafic lié aux pages web, des études ont montré que 40% des personnes quittent un site web si elles doivent attendre plus de 3 secondes pour obtenir la première information [62]. Pour réduire le temps d'affichage, les navigateurs peuvent utiliser différentes connexions TCP pour faire plusieurs requêtes HTTP simultanément (par exemple, Firefox et Chrome utilisent au maximum 6 connexions TCP simultanément). En ce qui concerne, le trafic lié à la vidéo en streaming, d'autres études ont révélé que le taux d'abandon augmente rapidement après plus de deux secondes passées à attendre le démarrage de la vidéo (5.8% plus d'abandons par seconde après les deux premières secondes). Ensuite, après ces premiers créneaux critiques, les flux réseau pour les pages web et les services vidéo en streaming ne nécessitent plus autant de bande passante que dans les premières secondes. Généralement dans la vidéo en streaming, une fois le *buffer* rempli, les mécanismes d'adaptation du débit vidéo peuvent s'adapter à une légère réduction du débit [63].

En raison de la nécessité d'avoir davantage de données le plus tôt possible, les fournisseurs de services sont enclin à mettre en œuvre un algorithme de contrôle de congestion, qui est particulièrement agressif en tout début de session. En d'autres termes, l'algorithme permettrait d'obtenir plus de débit que les autres flux concurrents pendant les premiers échanges de données. Au contraire, les opérateurs de réseaux sont soucieux de maintenir un partage équitable du goulot d'étranglement tout au long de la session et en particulier dans les premiers créneaux de celle-ci. Dans cette section, nous abordons ce problème en introduisant une nouvelle mesure de l'équité, permettant de considérer de façon plus importante le début de l'échange plutôt que la fin.

4.3.2 Définition de la métrique

Pour évaluer l'équité au niveau d'une session entière en pondérant chaque moment de la session par des valeurs de poids différents, nous introduisons une nouvelle métrique appelée Weighted Session Fairness Assessment (WSFA). En effet, l'idée principale derrière WSFA est de toujours découper les sessions en plusieurs créneaux, tout en mesurant l'équité pendant chacun d'entre eux, mais aussi d'agréger toutes les mesures en appliquant un poids à chaque moment afin de calculer une équité globale au niveau de la session.

Cette mesure d'équité, nommée WSFA, reprend exactement le même principe que SFA durant les premières étapes. À savoir, que nous nous concentrons sur le cas de deux flux concurrents A et B. Nous découpons le temps de la session en S créneaux temporels de 100ms chacun. Pour chaque créneau temporel $i, 1 \leq i \leq S$, nous collectons deux mesures :

— L'équité J_i pendant le créneau i, en utilisant l'indice de Jain (voir equation 4.1).

— Le flux dominant d_i, qui indique quel flux reçoit la plus grande quantité de données pendant le créneau i (voir équation 4.2).

Par conséquent, nous étudions dans cette section une fonction particulière d'agrégation des indices de Jain dans laquelle plus d'importance est accordée au début de la session :

$$F = \frac{\sum_{i=1}^{S} d_i * (1 - J_i) * \alpha_i}{\sum_{i=1}^{S} \alpha_i} \tag{4.4}$$

où α_i est la valeur du poids de chaque créneau i. Puisque les premières étapes sont plus importantes que les suivantes, nous définissons α_i comme une fonction strictement décroissante en fonction de i :

$$\alpha_i = \frac{1}{i} \tag{4.5}$$

4.3.3 Application de la métrique sur des exemples

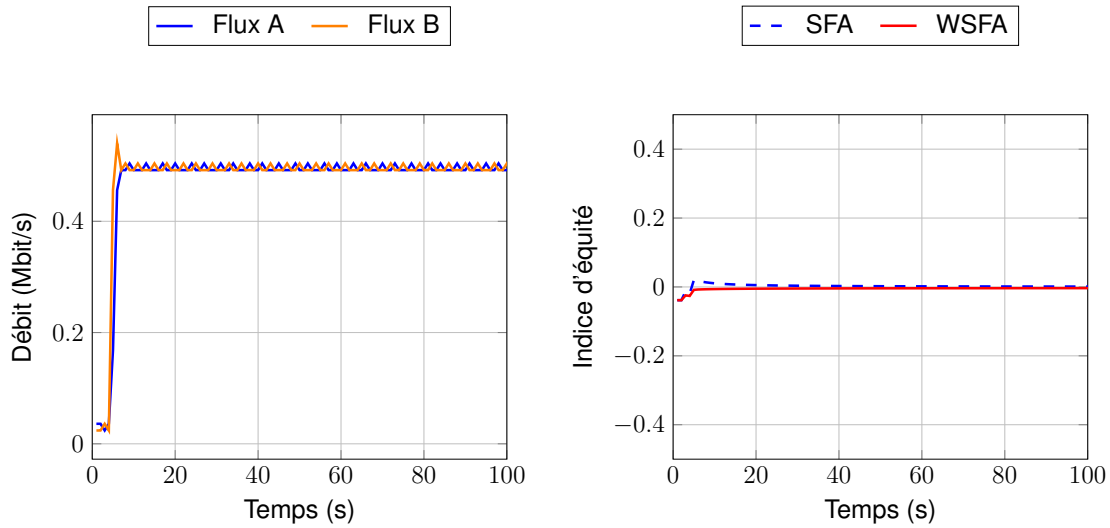
Pour illustrer cette deuxième méthode de mesure d'équité, nommée WSFA, nous nous basons sur les trois mêmes exemples que pour la méthode SFA. À partir de ces expériences, nous pouvons comparer les deux méthodes et mesurer l'impact de la pondération sur l'équité.

Cas équitable et inéquitable

Dans les deux cas, équitable (voir figure 4.5) et inéquitable (voir figure 4.6), nous observons des comportements similaires entre les deux méthodes de mesure. En effet, très peu de variations de débits sur l'ensemble des flux sont visibles sur les figures 4.5a et 4.6a, donc la prise en compte de la pondération des premiers instants de la session n'a que peu d'effet sur les courbes d'équité.

Cas d'alternance de domination

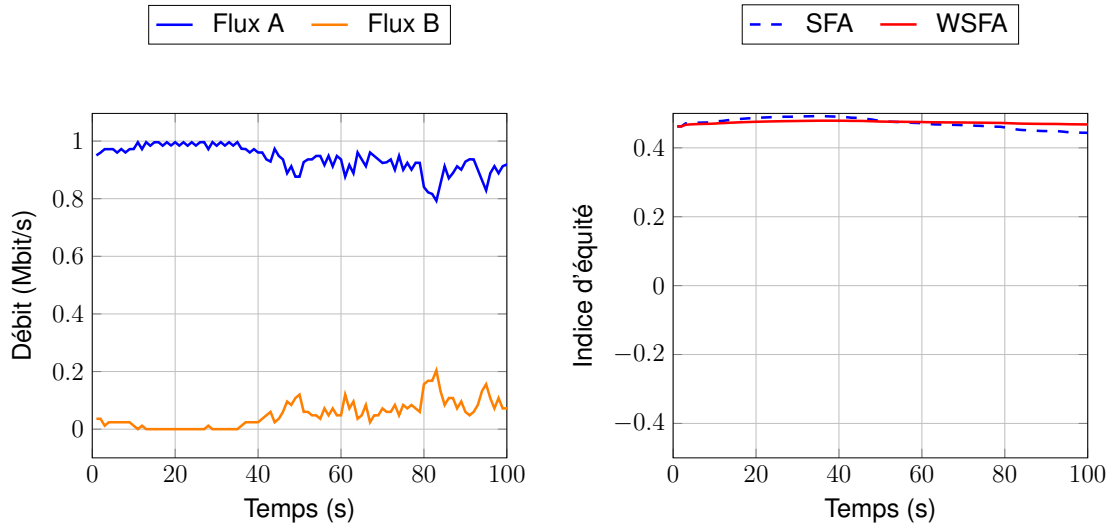
Dans le troisième cas, représenté dans la figure 4.7 qui représente le cas d'alternance des flux dominants. Sur la figure 4.7b nous constatons de réels écarts entre les deux courbes d'équité. Lors de la première période, entre 0s et 40s lorsque le flux A domine le flux B, les deux courbes d'équité tendent vers 0.5 ce qui traduit des flux inéquitables en faveur du flux A. Ceci s'explique par les mêmes raisons qu'avec le cas inéquitable. Puis, durant la seconde



(a) Évolution du débit

(b) Résultat de l'évolution de l'indice d'équité avec la méthode WSFA

FIGURE 4.5 – Exemple de mesure d'équité pour un cas équitable avec la méthode SFA et WSFA.

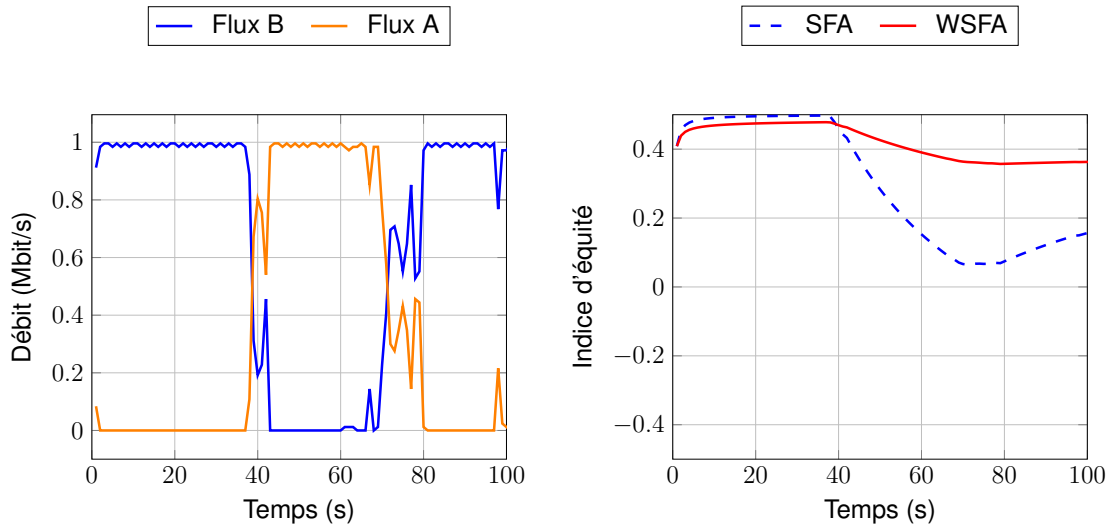


(a) Évolution du débit

(b) Résultat de l'évolution de l'indice d'équité avec la méthode WSFA

FIGURE 4.6 – Exemple de mesure d'équité pour un cas inéquitable avec la méthode SFA et WSFA.

période, entre 40s et 70s, l'inversion de la dominance a beaucoup moins d'impact sur l'équité avec la méthode WSFA que sur SFA. En effet, la méthode WSFA donne plus d'importance au début de session qu'à la fin de session avec la valeur de α_i donc la compensation causée par



(a) Évolution du débit

(b) Résultat de l'évolution de l'indice d'équité avec la méthode WSFA

FIGURE 4.7 – Exemple de mesure d'équité pour un cas d'alternance de domination avec la méthode SFA et WSFA

l'inversion du protocole dominant est beaucoup plus faible qu'avec la méthode SFA. En effet, nous vérifions à 70s qu'un écart d'équité d'environ 0.2 est visible entre les deux courbes. Enfin durant la dernière période, après les 70 premières secondes, nous constatons que la courbe de la méthode WSFA se stabilise autour de 0.38 car les coefficients α_i sont très faibles comparés aux premiers de la session. Tandis qu'avec la méthode SFA, nous remarquons une augmentation de l'inéquité, car cette méthode donne autant d'importance aux premiers échantillons qu'aux derniers.

4.4 Bilan des méthodes de mesure de l'équité

Dans ce chapitre, nous introduisons deux procédures impartiales de mesure de l'équité appelées SFA et WSFA. Ces nouvelles méthodes permettent de mesurer l'équité entre plusieurs flux en compétition sur un même goulot d'étranglement. Ces mesures proposées donnent une valeur quantitative de l'équité qui varie entre -0.5 et 0.5 avec les valeurs les plus équitables proches de 0. Une originalité clé de ce travail est que l'équité est évaluée durant toute la durée de la session. En effet, la première méthode SFA évalue l'équité sur l'intégralité de la session d'un utilisateur et considère chaque moment de la session de la même manière. La deuxième, WSFA pondère de façon plus importante le début de la session qui est un moment critique pour l'utilisateur. Aujourd'hui, cette deuxième métrique est paramétrée avec une pondération

plus importante en début de session, mais en modifiant les valeurs de α_i (voir équation 4.5) il serait possible de pondérer toute autre partie de la session ou évènement se produisant au cours du téléchargement des contenus.

Cette métrique a été conçue pour mesurer l'équité entre deux flux qui se partagent un point de saturation. Pour pouvoir étendre cette métrique à plus de flux, nous devons modifier la valeur de D_i calculée sur chaque échantillon. Une première solution est de la modifier et la passer dans l'ensemble des complexes pour pouvoir comparer jusqu'à 4 flux. Ainsi à partir de la partie réelle et complexe du nombre obtenu, nous sommes capables de déterminer quel est le flux dominant. Si nous souhaitons analyser encore plus de flux, nous devrons impérativement scinder la métrique en deux parties. Nous obtiendrons une première valeur traduisant l'équité moyenne entre les flux (basée sur l'indice de Jain qui lui, prend en considération N flux pour mesurer l'équité), puis une seconde valeur permettant de mesurer la dominance des flux. L'idée derrière cette seconde valeur est de pouvoir analyser sur chaque échantillon la dominance d'un flux et lui affecter un poids pour traduire cette dernière (plus la dominance d'un flux est importante, plus la valeur du poids sera importante). Par la suite, il ne restera plus qu'à agréger l'ensemble poids de chaque flux pour déterminer la dominance de chacun.

L'ensemble de ces deux méthodes de calcul d'équité a été implémenté dans un programme *open source*. Ce programme permet de récupérer les données de débit de la plateforme de tests pour indiquer une valeur d'équité entre les différents flux d'un scénarios testés (voir chapitre 3).

Dans les prochains chapitres, nous simulerons du trafic réel sur la plateforme de tests, entre différents utilisateurs et serveurs, avec plusieurs protocoles de transport (QUIC et TCP) et plusieurs algorithmes de contrôle de congestion (RENO, BBR, PCC). Ensuite, nous appliquerons ces deux méthodes pour déterminer l'équité entre ces différents flux.

Chapitre 5

Équité des protocoles de transport : QUIC versus TCP

Contents

Nous nous concentrons dans ce chapitre de thèse à l'évaluation des implémentations des protocoles de transport sur l'équité. Pour ce faire, nous étudions les protocoles QUIC et TCP, sur des réseaux à débit constant et à débit variable. Plusieurs situations réseau sont testées sur la plateforme de tests présentée dans le chapitre 3.

Dans un premier temps, nous présentons une évolution majeure dans les implémentations des algorithmes de contrôle de congestion dans le protocole QUIC, à savoir la simulation de plusieurs connexions TCP dans QUIC sur la taille de la fenêtre de congestion. Puis les scénarios et la configuration de la plateforme de tests pour réaliser l'évaluation de l'équité sont présentés dans la deuxième section. Enfin, dans la dernière partie, nous nous focalisons sur les résultats, en analysant les mesures d'équité entre les protocoles de transport TCP et QUIC. En d'autres termes, nous nous concentrons sur l'impact qu'ont les options et paramètres suivants sur l'équité : l'hystart présent dans les deux protocoles de transport, le nombre de connexions

TCP émulées, et la limitation de la taille de la fenêtre de congestion dans QUIC. Nous démontrons que l'ensemble de ces paramètres peut avoir des comportements néfastes sur l'équité entre les utilisateurs.

5.1 Analyse des implémentations de CUBIC dans les protocoles TCP et QUIC

CUBIC est un algorithme de contrôle de congestion largement utilisé sur Internet et directement implémenté dans le protocole TCP. Il est également présent dans de nouveaux protocoles de transport tels que QUIC. Mais les nouvelles implémentations de CUBIC dans QUIC ont introduit des paramètres supplémentaires afin d'émuler le comportement de plusieurs connexions TCP. En effet, le protocole QUIC multiplexe l'équivalent de plusieurs connections TCP sur une unique connexion de transport UDP. Pour plus d'informations, référez-vous à la section 2.1.4.

Pour essayer de rétablir une équité entre le protocole TCP et QUIC, une émulation de plusieurs connexions TCP est présente dans les implémentations de QUIC (notamment dans QUIC-GO [23], proto-QUIC [25] et Chromium [24]). Ce paramètre sera appelé N dans la suite de ce manuscrit. A notre connaissance, cette émulation de plusieurs connexions TCP dans CUBIC n'a pas encore été détaillée dans la littérature. En effet, cette fonctionnalité n'est documentée ni par l'IETF [8], ni dans les papiers traitant de QUIC.

Pour mieux comprendre pourquoi ce paramètre a été mis en place, il faut se rappeler que les navigateurs web n'ouvrent plus qu'une seule connexion QUIC pour télécharger les données d'un site web grâce au multiplexage. Au contraire, TCP (le protcole que remplace QUIC) ne possède pas de multiplexage, donc les navigateurs web ouvrent plusieurs connexions TCP pour télécharger les données d'un site web. Afin de ne pas pénaliser le protocole QUIC dans ce cas d'utilisation, QUIC émule plusieurs connexions TCP. Or dans les implémentations actuelles, cette valeur est toujours constante durant la durée d'une session et est généralement égale à 2. De plus, dans les implémentations nous n'avons trouvé aucun lien entre cette valeur (N) et le nombre de streams présents dans le protocole QUIC. Or ce paramètre (N) a un impact direct sur le comportement de la taille de la fenêtre de congestion qui détermine le débit du protocole de transport (c'est-à-dire le nombre d'octets qui peuvent être envoyés sans recevoir d'acquittement). La présente étude se concentre sur QUIC-GO [23] qui nous sert de référence pour analyser l'implémentation du nombre de connexions TCP émulées dans QUIC avec CUBIC comme algorithme de contrôle de congestion. À chaque étape, nous comparons cette implémentation avec celle de Linux qui nous sert de référence pour TCP.

Dans la plupart des programmes de QUIC analysés, la valeur de N est constante durant toute la session (par défaut, dans différentes implémentations, elle est égale à 2). Ce qui est plus surprenant, c'est que cette valeur ne prend pas en compte les caractéristiques du flux de données (par exemple le nombre de streams dans QUIC).

La présentation de l'implémentation de N se passe en trois étapes (le *slow-start*, la *congestion avoidance* et la diminution de débit en cas de perte) détaillées ci-dessous :

5.1.1 Slow-start

La première phase de l'algorithme de contrôle de congestion CUBIC est le *slow-start*. Cette phase est inchangée comparée à celle de RENO et est détaillé dans la partie 2.2.1. Elle est présente dans de nombreux algorithmes de contrôle de congestion. Dans le cas qui nous intéresse, elle est implémentée dans l'algorithme de contrôle de congestion CUBIC présent dans les deux protocoles de transport (QUIC dans QUIC-GO et TCP dans Linux). Pour rappel, cette phase se produit au début d'une connexion ou après un évènement majeur de congestion comme le *timeout*. Elle initialise une valeur de débit très faible pour l'utilisateur et l'augmente de façon exponentielle pour déterminer le plus rapidement possible le débit de l'utilisateur en fonction de la bande passante disponible dans le réseau. Dans la phase du *slow-start*, l'augmentation de la taille des fenêtres (proportionnelle au débit) est indiquée dans l'équation suivante :

$$cWS = cWS_{t-1} + MSS \tag{5.1}$$

où cWS (congestion Window Size) est la taille de la fenêtre de congestion, cWS_{t-1} la taille de la fenêtre de congestion avant l'acquittement et MSS (Maximum Segment Size) est la taille maximale d'un paquet. Cette formule recalcule la taille de la fenêtre de congestion à chaque réception d'un nouvel acquittement. Ce principe augmente alors la taille de la fenêtre de congestion de façon exponentielle pendant toute la durée du *slow-start*. Cet accroissement de la taille de la fenêtre de congestion permet alors une augmentation très rapide du débit de l'utilisateur. Cette phase de *slow-start* dépend donc uniquement de deux paramètres : *(i)* le moment où arrive l'acquittement d'un paquet et *(ii)* la taille maximale autorisée d'un paquet. Donc, dans cette phase, l'augmentation de la fenêtre de congestion ne dépend pas du nombre de connexions TCP émulées N.

5.1.2 Diminution du débit lors d'une perte de paquet sans timeout

Dans cette section, nous analysons uniquement la diminution de débit après une perte de paquets signalée par trois paquets "DUP ACK" dans CUBIC et non par un *timeout* dans les deux implémentations (TCP et QUIC). En effet, que ce soit pour QUIC ou pour TCP lorsqu'une perte se produit du fait d'un évènement *timeout*, l'algorithme de contrôle de congestion réduit la taille de la fenêtre à la valeur initiale qu'il avait en début de connexion, puis il commence une nouvelle phase de *slow-start*. Nous détaillons ci-dessous la diminution du débit après évènement de perte dans TCP-CUBIC et QUIC-CUBIC.

TCP-Linux : Pour analyser l'impact de N sur la diminution de la taille de la fenêtre de congestion après une perte de paquets signalée par un "DUP ACK", nous nous basons sur l'implémentation de Linux pour obtenir un point de référence et comparer les différences visibles dans l'implémentation de QUIC-GO. La diminution de la taille des fenêtres de congestion de CUBIC dans l'implémentation de Linux (cWS_{Linux}) est déterminée de la manière suivante :

$$cWS_{Linux} = \frac{B_{linux}}{BittcpBetaScale} \cdot cWS_{b-c} \tag{5.2}$$

où cWS_{b-c} est la taille de la fenêtre avant l'événement de congestion, $B_{linux}/BitcpBetaScale$ représente le coefficient de diminution de la taille de la fenêtre de congestion. Les valeurs de B_{linux} et $BitcpBetaScale$ sont respectivement égales à 717 et 1024, ce qui représente une diminution de 30% de sa fenêtre.

QUIC-QUICGO : Lors de l'analyse du code de QUIC-GO, nous remarquons que l'équation diffère de celle de TCP implémentée sur Linux. En effet, elle modifie la diminution de la taille de la fenêtre de congestion pour prendre en compte le nombre de connexions TCP émulées dans QUIC. La nouvelle équation pour la diminution de la taille de la fenêtre de congestion dans QUIC-GO (cWS_{QUIC}) après une perte sans *timeout* est la suivante :

$$cWS_{QUIC} = B_{QUIC} * cWS_{b-c} \tag{5.3}$$

Avec $B_{QUIC} = (N-1+\beta)/N$ et $\beta = 0.7$, donc B_{QUIC} est une fonction croissante en fonction de N, c'est-à-dire du nombre de connexions TCP émulées. Cela signifie qu'un événement de congestion entraîne une diminution plus faible de la taille de la fenêtre de congestion (du débit) lorsque N est élevé que lorsqu'il est faible. Maintenant, si nous posons $N = 1$, nous obtenons alors la même diminution de la taille de la fenêtre de congestion entre TCP et QUIC, soit $cWS_{Linux} = cWS_{QUIC},\ pour\ N = 1$.

L'évolution de la diminution de la taille de la fenêtre de congestion de QUIC est illustrée dans la figure 5.1 qui montre comment la taille de la fenêtre est modifiée par un événement de congestion pour N égal à $\{1, 2, 5, 10\}$. En supposant que l'événement de congestion se produit juste avant 350ms, la taille des fenêtres de congestion (c'est-à-dire le débit autorisé) diminue différemment, en fonction de N. En effet, nous observons que, lorsque l'algorithme de contrôle de congestion émule une seule connexion TCP, le débit autorisé est réduit de 30% alors qu'il est réduit d'à peine 3% lorsque l'algorithme de contrôle de congestion émule 10 connexions TCP.

5.1.3 Congestion avoidance : augmentation du débit

Dans cette dernière sous-section, nous analysons cette fois-ci l'augmentation de taille de la fenêtre de congestion durant la phase de *congestion avoidance*. Pour rappel, cette phase se produit après une phase de *slow-start* ou après une perte de paquets (sans apparition de

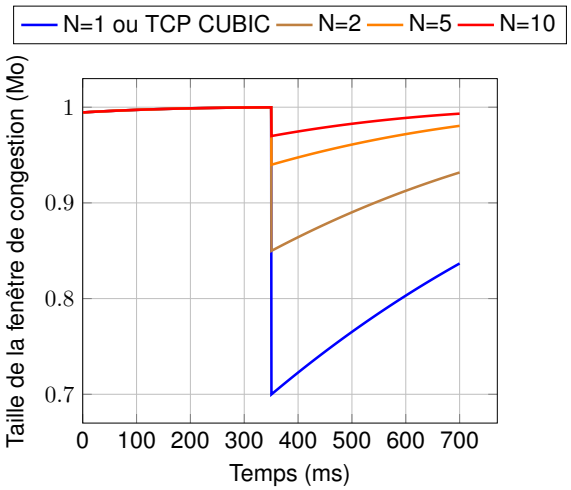


FIGURE 5.1 – Diminution de la taille de la fenêtre de congestion en fonction du nombre de connections TCP émulées dans QUIC (N).

timeout). Elle permet alors une augmentation plus lente de la taille de la fenêtre de congestion pour éviter d'augmenter le taux de perte pour l'utilisateur. Pour plus de détails, référez-vous à la section 2.2.4.

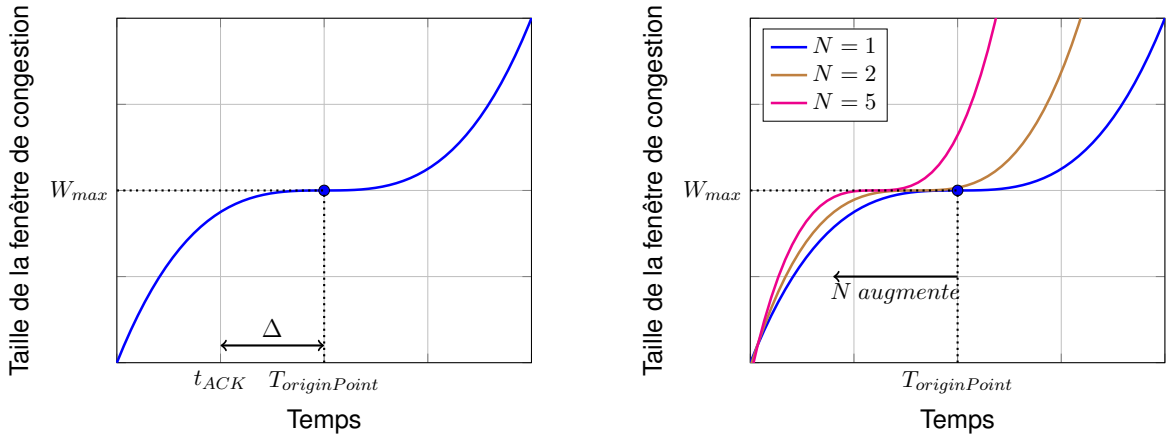
Pour analyser l'implémentation de QUIC-GO, nous faisons une analyse complète des quatre étapes nécessaires pour déterminer la nouvelle fenêtre de congestion après chaque acquittement reçu par l'utilisateur. Cette analyse compare toujours l'implémentation de CUBIC de TCP dans Linux et l'implémentation de QUIC-GO. Ceci permet de contrôler les modifications réalisées dans l'algorithme pour intégrer la valeur de N.

Forme CUBIC :

L'augmentation de la fenêtre de congestion est contrôlée par l'équation 2.2. Dans la plupart des implémentations de CUBIC, en particulier sur les deux qui nous intéressent, elles déterminent une valeur Δ correspondant à la différence de temps entre le moment de réception de l'acquittement (t_{ACK}) et $T_{originPoint}$ (voir figure 5.2a). $T_{originPoint}$ correspond au moment où la taille de la fenêtre de congestion est égale à W_{max} (W_{max} étant la valeur palier de la fonction CUBIC qui correspond généralement à la valeur de la taille de la fenêtre précédent le dernier événement de congestion).

TCP-Linux : Dans l'implémentation présente dans l'OS de Linux, le delta pour l'augmentation de la taille de la fenêtre de congestion (Δ_{QUIC}) correspond à l'équation suivante :

$$\Delta_{TCP} = CRS * \text{offset}^3 >> (10 + 3 + BICTCP) \tag{5.4}$$



(a) Augmentation du débit avec CUBIC.

(b) Impact de N sur la phase d'augmentation CUBIC.

FIGURE 5.2 – Évolution de la taille de la fenêtre de congestion avec l'algorithme CUBIC de QUIC-GO.

où CRS (Cube RTT Scale) et BICTCP sont des constantes respectivement égales à 410 et 10. La variable offset est donnée par offset $= f(T_{originPoint}, t_{ACK})$. $T_{originPoint}$ est calculé après chaque événement de congestion et peut prendre deux valeurs en fonction des conditions du réseau : 0 ou $\sqrt[3]{CF * (lastCWS_{max} - cWS_d)}$ où CF est le "Cube Factor", $lastCWS_{max}$ est la valeur maximale précédente de la taille de la fenêtre et cWS_d est la taille des fenêtres après un événement de congestion.

QUIC-QUICGO : QUIC-GO implémente le même type d'équation pour l'augmentation de la fenêtre de congestion, avec :

$$\Delta_{QUIC} = CCWS * \text{offset}^3 * MSS >> CS \tag{5.5}$$

où $CCWS$ est le "Cube Congestion Window Scale" et CS est le "Cube Scale". Ces deux valeurs (constantes) correspondent respectivement à CRS et $10 + 3 * BICTCP$ dans l'implémentation de CUBIC dans Linux et représentent les mêmes valeurs ($CRS = CCWS$ et $CS = 10 + *3BICTCP$). De plus, la valeur MSS est la taille d'un paquet émis sur le réseau et permet d'exprimer le résultat en octet et non en paquet, donc ce paramètre ne modifie pas le comportement entre les deux équations. Le calcul de l'offset est identique sauf pour le calcul de la valeur du $T_{originPoint}$. En effet, la valeur $T_{originPoint}$ dépend de cWS_{QUIC} qui est calculée dans l'équation 5.3 et qui dépend de la valeur de N. Ainsi, la valeur de $T_{originPoint}$ est déterminé en fonction du nombre de connexions TCP émulées N. Plus N est grand, plus la taille de la fenêtre de congestion atteint rapidement la valeur de W_{max} (voir figure 5.2b). L'implémentation QUIC-GO définit par défaut les valeurs suivantes : $CCWS = 410$, $CS = 40$, $MSS = 1460$ et $CF = 1 << CS/CCWS/MSS$.

De même que pour la diminution de débit, lorsque $N = 1$, l'augmentation de la taille de la fenêtre de congestion de QUIC est identique à celle de TCP ($\Delta_{QUIC} = \Delta_{TCP},\ pour\ N = 1$).

Limitation de l'augmentation de la fenêtre de congestion

Cette limitation est utilisée pour conserver l'équité entre CUBIC et les autres algorithmes de contrôle de congestion présents sur le réseau. Cette limitation se retrouve dans les deux implémentations CUBIC (TCP-Linux et QUIC-QUICGO).

TCP-Linux : Cette limitation dans l'implémentation de Linux se présente sous la forme du calcul suivant :

$$cWS = \max(cWS_{t-1} + \Delta_{TCP}, cWS_{t-1} + 2) \tag{5.6}$$

Ce calcul est appliqué tous les deux acquittements, ce qui entraîne une limitation maximale de la valeur d'accroissement de la taille de la fenêtre de congestion de 1.5 fois par RTT.

QUIC-QUICGO : Nous définissons m comme le nombre d'octets acquittés, alors l'implémentation de CUBIC dans QUIC-GO limite la taille de la fenêtre à $cWS_{threshold}$:

$$cWS_{threshold} = cWS_{t-1} + \frac{m}{2} \tag{5.7}$$

Cette limitation de l'incrémentation de la taille de la fenêtre de congestion dans le protocole QUIC-GO correspond à la même valeur sur l'algorithme CUBIC dans Linux. Tous deux sont limités à la moitié des octets acquittés par RTT.

Cette limitation prend en compte uniquement le nombre d'octets acquittés et la taille de la précédente fenêtre. Donc cette dernière ne prend pas en compte la valeur de N pour limiter l'augmentation de la fenêtre de congestion.

TCP-friendliness

Pour s'assurer que la taille de la fenêtre atteinte par CUBIC est au moins égale à celle que RENO pourrait atteindre, CUBIC fixe la nouvelle taille de fenêtre comme étant le maximum entre le cWS (CUBIC) actuel et la taille estimée de la fenêtre de congestion $eCWS$ de RENO, c'est-à-dire $cWS = max(cWS, eCWS)$. Cette vérification est à la fois présente dans CUBIC-Linux et dans QUIC-GO.

Linux : L'équation de TCP-friendliness prend la forme suivante [28] :

$$eCWS = cWS_{t-1} + \frac{2\beta}{2-\beta} \cdot \frac{m}{cwnd_{t-1}} \tag{5.8}$$

avec $\beta = 0.7$ correspondant à la constante de diminution de la taille de la fenêtre de congestion en cas de congestion et $cwnd_{t-1}$ la taille de la fenêtre de congestion avant l'augmentation du débit. Ce calcul permet d'estimer une augmentation de la taille de la fenêtre de congestion de l'algorithme de contrôle de congestion RENO pour vérifier que l'accroissement de CUBIC n'est pas plus faible que celui de RENO dans les mêmes conditions réseau.

QUIC-GO : La même vérification est faite dans QUIC-GO mais avec une équation différente pour prendre en compte le nombre de connexions TCP émulées dans QUIC. La taille estimée de la fenêtre de congestion de RENO $eCWS$ est alors donnée par :

$$eCWS = eCWS_{t-1} + \frac{m * \alpha * MSS}{eCWS_{t-1}} \tag{5.9}$$

où α est défini par $\alpha = 3 * N^2 * (1 - B)/(1 + B)$ où $B = (N - 1 + \beta)/N$. La formule ci-dessus montre que $eCWS$ dépend de N, et implique que le débit augmente plus rapidement pour des valeurs supérieures de N.

Limitation maximale de la taille de la fenêtre de congestion.

À notre connaissance, la limitation de la taille de la fenêtre de congestion est uniquement présente dans l'algorithme de congestion CUBIC qu'on retrouve dans plusieurs implémentations de QUIC, dont QUIC-GO. Elle permet de limiter l'accroissement de la taille maximale de la fenêtre à cWS_{max}. Cette valeur ne tient pas compte de la valeur de N avec :

$$cWS_{max} = MAX_p * MSS \tag{5.10}$$

où MAX_p est le nombre maximum de paquets dans la fenêtre de congestion. Dans l'implémentation de QUIC-GO MAX_p est fixé par défaut à 1000.

Évaluation de l'impact de N sur l'accroissement de la taille de la fenêtre de congestion

L'évolution de la taille de la fenêtre pendant l'évitement de la congestion dans QUIC-GO est illustrée dans la figure 5.3 pour diverses valeurs différentes de N. Un événement de congestion se produit juste avant 0ms. Nous supposons que la fenêtre de congestion précédente est égale à 150ko et que W_{max} est de 100ko. Nous admettons également qu'un acquittement est produit pour chaque segment et que le RTT est égal à 50ms.

La figure 5.3 présente trois étapes distinct : durant les 800 premières millisecondes, la taille de la fenêtre de congestion augmente selon la fonction cubique pour $N = 1$ et $N = 2$. Dans le même temps, pour des valeurs plus élevées de N, l'augmentation de la taille de la fenêtre RENO dépasse celle de la fenêtre CUBIC (du fait que CUBIC simule plusieurs connexions RENO) ; la taille de la fenêtre de congestion résultante présente donc une augmentation linéaire selon l'équation 5.9. Entre 800ms et 1500ms l'augmentation de la taille de la fenêtre de

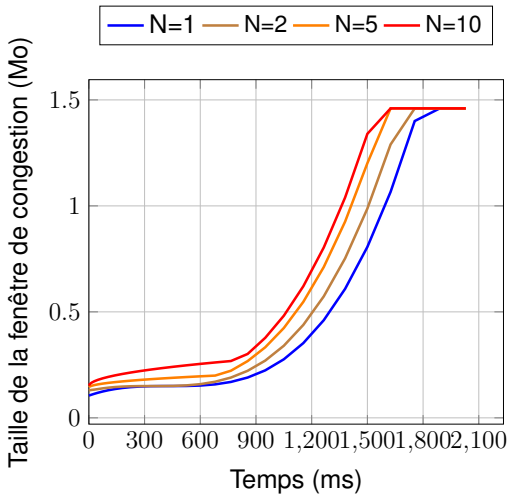


FIGURE 5.3 – Augmentation de la taille de la fenêtre de congestion en fonction du nombre de connections TCP émulées dans QUIC (N).

congestion est fixée à $cWS_{threshold}$ selon l'équation 5.7, indépendamment de N : ceci correspond à une augmentation linéaire avec la même pente pour tous les N. Enfin, après 1500ms, la taille de la fenêtre est fixée à la valeur maximale cWS_{max} spécifiée par QUIC dans l'équation 5.10. Évidemment, plus N est grand, plus la taille de la fenêtre augmente rapidement.

À noter que l'évolution de CUBIC dans TCP implémenté dans Linux aura la même évolution de croissance que $N = 1$, sauf pour la limitation maximale de la taille de la fenêtre de congestion absente dans Linux.

5.2 Scénarios de tests

Nous étudions l'impact du protocole QUIC et TCP sur l'équité dans les deux types de réseau implémentés sur la plateforme illustrée dans le chapitre 3, c'est-à-dire les réseaux à débit constant (par exemple : ADSL) et les réseaux à débit variable (comme les réseaux 4G). C'est la deuxième version de la "partie réseau" de la plateforme (qui permet de simuler un réseau à débit constant et à débit variable) qui est utilisée pour réaliser l'intégralité des tests de ce chapitre (voir section 3.1.2). Les services QUIC et TCP sont directement implémentés dans la plateforme de tests. Pour rappel, nous utilisons la pile TCP fournie par le noyau Linux implémentée dans Ubuntu 16-04 et le programme QUIC-GO [23] correspondant au protocole QUIC spécifié par l'IETF dans [64]. Dans notre configuration, les deux protocoles QUIC et TCP utilisent CUBIC comme algorithme de contrôle de congestion. De plus, plusieurs paramètres TCP par défaut sont modifiés, car ils ne sont pas actuellement implémentés dans QUIC qui est encore un protocole assez jeune dans les domaines de l'accélération et de l'optimisation. Premièrement, l'option de TCP Segmentation Offload (TSO) de TCP est désactivée. TSO vise à réduire

la charge Central Processing Unit (CPU), en déplaçant la segmentation des données du noyau vers la carte réseau. Cette segmentation des données permet d'envoyer les données sur un réseau et les découper à la taille idéale. La transposition dans QUIC de TSO, nommée Generic Segmentation Offload (GSO) [65], n'est pas encore supportée par les implémentations actuellement disponibles. Deuxièmement, le paramètre du noyau Linux *tcp_no_metric_save* (qui n'a pas d'équivalent dans les implémentations QUIC disponibles) est activé pour rendre les tests successifs indépendants les uns des autres. En effet, ce paramètre permet de sauvegarder les états des précédentes connexions, pour redéfinir les valeurs initiales de la nouvelle connexion.

Nous effectuons, sur la plateforme de tests, une combinaison exhaustive de scénarios réseau (ci-après dénommés tests) selon les paramètres indiqués dans le tableau 5.1. Les différents tests ont été obtenus en variant plusieurs caractéristiques du réseau et valeurs par défaut des protocoles, c'est-à-dire la latence, le taux de perte (pour les pertes qui ne sont pas dues aux files d'attente), la taille du *buffer*, le débit du point de saturation, le nombre de connexions TCP émulées dans QUIC (N) et la taille maximale de la fenêtre de congestion dans QUIC (CWS_{max}).

Tableau 5.1 – Caractéristiques du réseau et de la session.

Type de réseau	réseau à débit constant		réseau à débit variable
Débit (Mbit/s)	1	34	variable (moyen 34, max 60)
Latence (ms)	50 ; 300		
Taux de perte (%)	0 ; 0.1 ; 0.5		
Taille buffer (ko)	11.3 ; 62.3	425 ; 2550	750 ; 4500
N	1 ; 2 ; 5		
CWS$_{max}$	1000	1000 ; 10000	
Taille du fichier (Mo)	10	340	100

Les paramètres réseau (par exemple : la latence, le taux de perte) sont déterminés en fonction des valeurs trouvées sur les réseaux opérationnels Orange. Quant aux tailles du *buffer* (noté BS), elles sont fixées à deux fois le produit du BDP. Le BDP correspond à la multiplication du RTT par le débit lorsque celui-ci est constant ou alors par le débit maximum trouvé dans la trace de livraison des paquets pour les réseaux à débit variable. L'équation suivante montre comment est calculée la taille du *buffer* pour l'ensemble des configurations réseau possibles :

$$BS = 2 * BDP = 2 * L * D_{max} \tag{5.11}$$

où D correspond au débit maximal du point de saturation et L la latence sur le réseau. Dans notre cas, nous supposons que le RTT correspond environ à la valeur ajoutée de la latence. En ce qui concerne le nombre de connexions TCP émulées dans QUIC nous avons pris trois valeurs différentes : $1, 2, 5$. Pour rappel, la valeur par défaut dans les implémentations est de 2. Enfin, pour réaliser les mesures d'équité, les serveurs envoient des fichiers de différentes tailles pour que chaque mesure d'équité dure au minimum 60 secondes.

Chaque test est répété 10 fois pour observer l'évolution de l'équité sur une même configuration réseau. Sur chacun des tests, nous déterminons l'équité sur la période de concurrence des flux pour l'ensemble des scénarios décrits dans le tableau 5.1. Cette mesure utilise la méthode de SFA, c'est-à-dire qu'elle considère les différents moments de la période de concurrence de la même manière (le début de la période de concurrence a le même poids dans la métrique que la fin). Tous les résultats sont présentés sous forme de cartes de couleurs. L'axe des abscisses est le numéro du test et l'axe des ordonnées est le taux de perte simulée dans l'émulateur réseau. Ce paramètre a été représenté sur l'axe des ordonnées, car l'émulation de plusieurs connections TCP dans QUIC a un fort impact sur la taille de la fenêtre de congestion en cas de perte de paquets (voir section 5.1). Les couleurs indiquent l'équité atteinte entre les protocoles. Une couleur verte indique que les résultats sont équitables (métrique proche de 0) alors qu'une couleur rouge indique que les résultats sont injustes (métrique proche de $0,5$ ou $-0,5$). Pour chaque test, un carré rouge indique que TCP est dominant (obtenir plus de débit pendant la session), tandis qu'un cercle bleu indique que QUIC est dominant.

L'analyse de l'équité entre les protocoles de transport QUIC et TCP a été réalisé dans l'ensemble des configuration réseau présente dans le tableau 5.1. Après une analyse des résultats, nous démontrons que les trois paramètres suivant ont un impact significatif sur l'équité : l'option hystart, le nombre de connexion TCP émulées dans QUIC et la limitation maximale de la taille de la fenêtre de congestion. Chaque paramètre est détaillé dans une section ci-dessous.

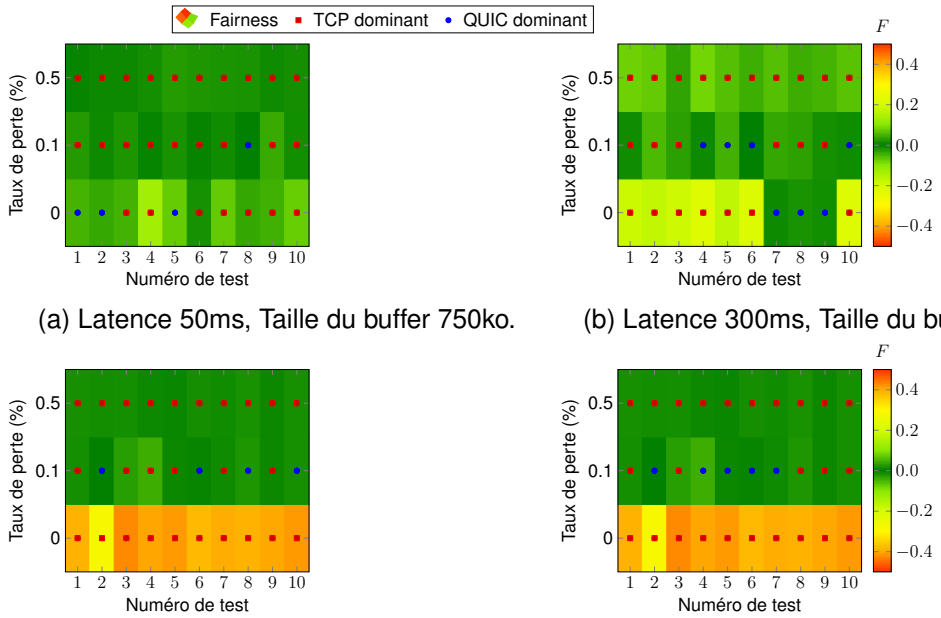
5.3 Analyse de l'impact sur l'équité du paramètre hystart

Cette section analyse l'équité obtenue entre un seul flux QUIC émulant un nombre de connexions TCP égal à 1 ($N = 1$) et un flux TCP unique. Les autres valeurs de N seront traitées dans la section 5.4. Nous commencerons par aborder le cas du réseau à débit variable puis nous étudirons les configurations à réseau fixe dans un second temps. Nous réalisons ce délai pour vérifier que le flux QUIC réussi à obtenir une part équivalente au flux TCP alors qui occupe l'intégralité de la bande passante.

5.3.1 Réseau à débit variable

La première série de mesures est effectuée sur un réseau à débit variable (réseau simulant un réseau mobile 4G). Pour vérifier que QUIC est capable d'obtenir une part équitable du débit lorsqu'un flux TCP est démarré en premier, le flux TCP commence la transmission des données 8 secondes avant le flux QUIC.

Les résultats d'équité sont présentés dans la figure 5.4. Pour les tests ayant un *buffer* important de 4.5Mo (voir les figures 5.4c et 5.4d), un problème d'équité est visible lorsque le taux de perte est fixé à 0% sur l'émulateur réseau. Durant ces tests, la moyenne des valeurs d'équité est proche de 0.40, ce qui indique une inéquité importante entre les flux.



(a) Latence 50ms, Taille du buffer 750ko.

(b) Latence 300ms, Taille du buffer 750ko.

(c) Latence 50ms, Taille du buffer 4.5Mo.

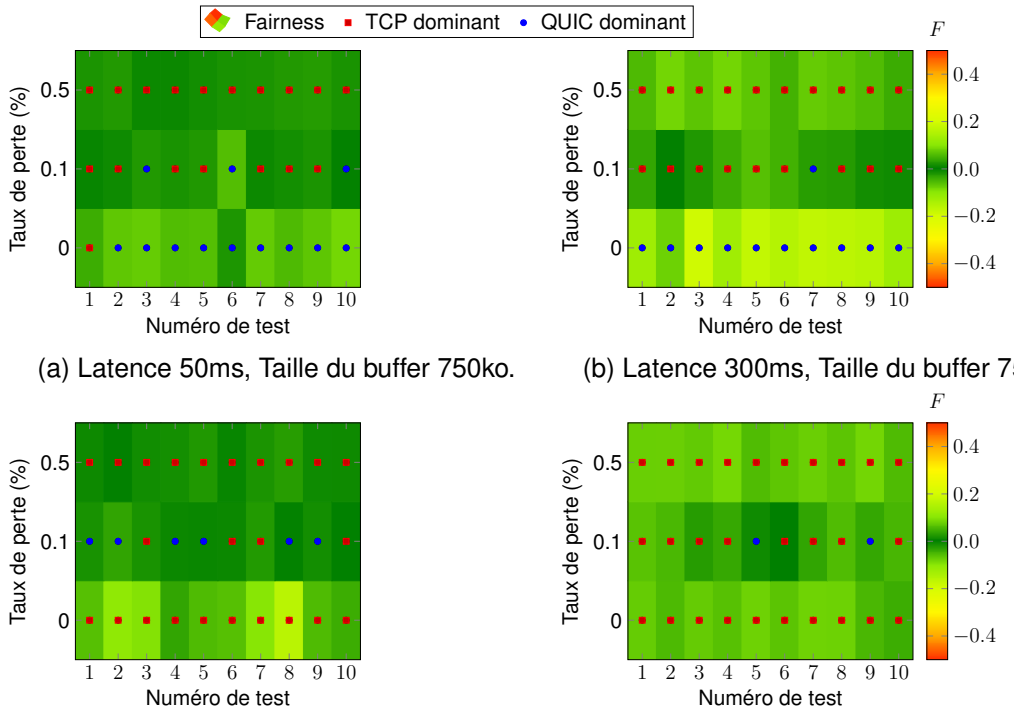
(d) Latence 300ms, Taille du buffer 4.5Mo.

FIGURE 5.4 – Évaluation de la fairness lorsqu'un flux TCP (hystart activée) démarre 8 secondes avant un flux QUIC ($N = 1$, hystart activée) avec un point de saturation à débit variable.

Cette injustice est illustrée dans la figure 5.6a qui montre comment les possibilités de transmission sont partagées lors d'un test représentatif de l'ensemble des configurations posant des problèmes d'équité. Pendant les 8 premières secondes, le flux TCP utilise toute la bande passante disponible, car il est seul sur le réseau. Après ces 8 premières secondes, le flux QUIC démarre l'émission de ces données. Cependant, la phase de *slow-start* prévue dans le protocole QUIC est absente, ce qui empêche une croissance rapide de la fenêtre de congestion QUIC. Le flux QUIC est capable d'utiliser une partie significative de la bande passante uniquement lorsque le flux TCP est terminé (à ≈ 50s).

Pour vérifier que c'est bien l'option hystart qui impacte l'équité entre les deux protocoles, nous la désactivons sur le flux QUIC. Les nouveaux résultats sans cette option sont indiqués sur la figure 5.5. Nous remarquons qu'il n'y a plus d'injustice entre QUIC et TCP. Ceci est également visible sur la figure 5.6b qui montre l'évolution du débit pour les deux flux avec les mêmes paramètres (aucune option hystart pour QUIC). On voit sur le début de la connexion QUIC, une phase de *slow-start* qui se traduit par une augmentation rapide du débit pour le flux QUIC. Cette augmentation plus importante du débit permet d'obtenir un partage équitable du débit entre les deux flux.

Afin de comprendre comment l'option hystart peut produire de l'injustice sur un réseau à débit variable, garder à l'esprit que la sortie du *slow-start* est causée par l'option hystart qui elle-même est due à une augmentation du RTT. Plusieurs phénomènes peuvent expliquer le déclenchement du hystart :



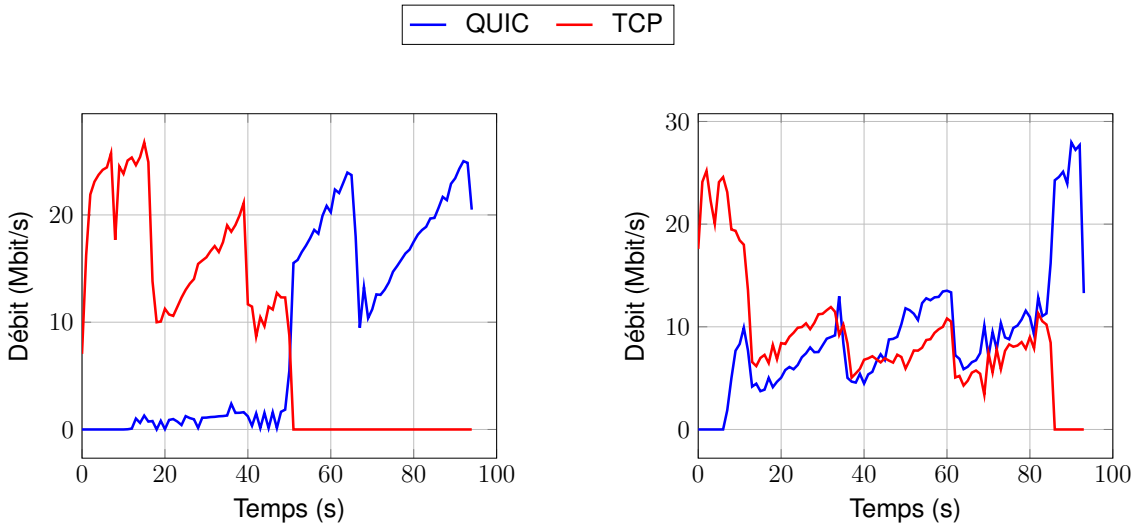
(a) Latence 50ms, Taille du buffer 750ko.

(b) Latence 300ms, Taille du buffer 750ko.

(c) Latence 50ms, Taille du buffer 4.5Mo.

(d) Latence 300ms, Taille du buffer 4.5Mo.

FIGURE 5.5 – Évaluation de l'équité lorsqu'un flux TCP (hystart activée) commence 8 secondes avant un flux QUIC ($N = 1$, hystart désactivée) avec un point de saturation à débit variable.



(a) 1 flux TCP et 1 flux QUIC $N = 1$, l'option hystart est activée sur les deux protocoles.

(b) 1 flux TCP et 1 flux QUIC $N = 1$, l'option hystart est désactivée sur les deux protocoles.

FIGURE 5.6 – Effets de l'option hystart sur le débit dans un réseau à débit variable.

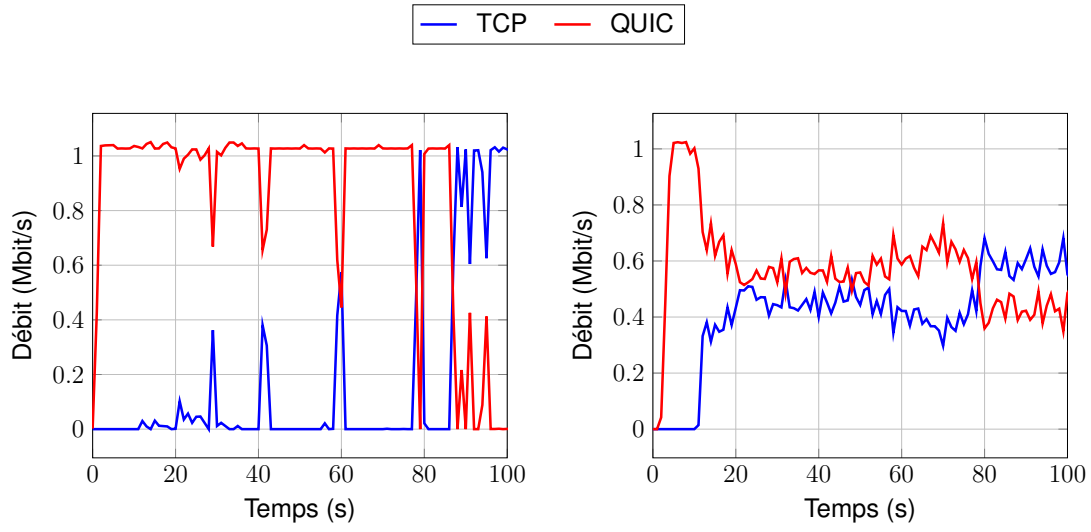
— *Remplissage en continu du buffer :* Pendant le démarrage du flux QUIC, le flux TCP continue d'augmenter son débit tant qu'aucune perte de paquets ne survient. Ayant très peu de pertes de paquets sur le réseau, cela a pour effet d'augmenter le remplissage du *buffer* et donc d'accroître le RTT. Ainsi, lorsque le flux QUIC est dans la phase du *slow-start*, le flux TCP provoque une augmentation du RTT sur flux QUIC.

— *Diminution du débit :* Le début du flux QUIC n'est pas parfaitement synchronisé avec le fichier de *token*s. Ainsi, le flux peut commencer soit pendant une phase d'augmentation de la capacité du réseau, soit pendant une phase de diminution de la capacité du réseau. Ainsi, lorsque le flux commence sur une phase de capacité réseau décroissante, ceci se traduit par une augmentation du RTT.

Aucun des phénomènes ci-dessus n'est en effet lié à une congestion naissante, bien que c'est ainsi que les deux soient (à tort) interprétés par hystart qui étrangle le flux QUIC.

5.3.2 Réseau à débit constant

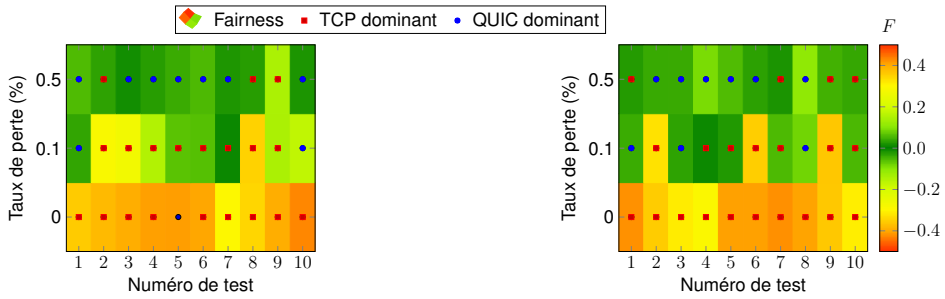
Durant la précédente sous-section, nous nous sommes focalisés sur le réseau à débit variable, maintenant nous nous concentrons sur des réseaux à débit constant. Pour exécuter ces nouveau tests, nous réutilisons les mêmes paramètres réseaux, à l'exception de la taille du *buffer* qui a été recalculée à deux fois la valeur du BDP. L'ensemble des paramètres de tests sont visibles dans le tableau 5.1. Le choix des débits est basé sur 1Mbit/s pour caractériser les réseaux à faible débit et à 34Mbit/s pour les réseaux avec des débits plus importants.



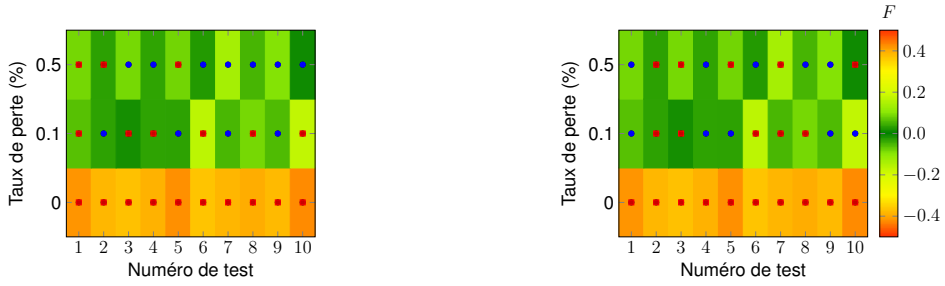
(a) 1 flux TCP et 1 flux QUIC $N = 1$, l'option hystart est activée sur les deux protocoles.

(b) 1 flux TCP et 1 flux QUIC $N = 1$, l'option hystart est désactivée sur les deux protocoles.

FIGURE 5.7 – Effets de l'option hystart sur les protocoles QUIC et TCP sur un réseau à débit constant de 1Mbit/s.

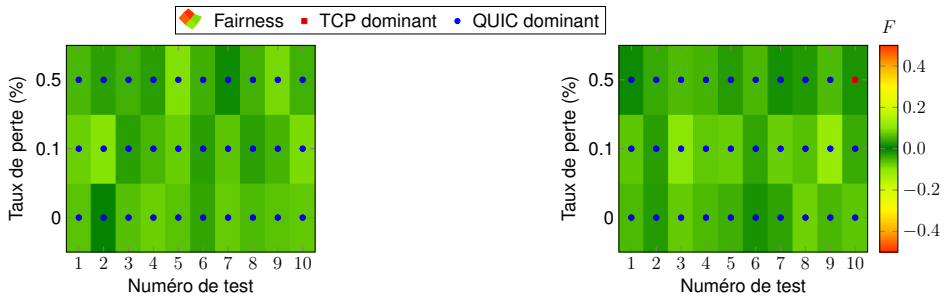


(a) Latence 50ms, Taille du buffer 11.25ko. (b) Latence 300ms, Taille du buffer 11.25ko.

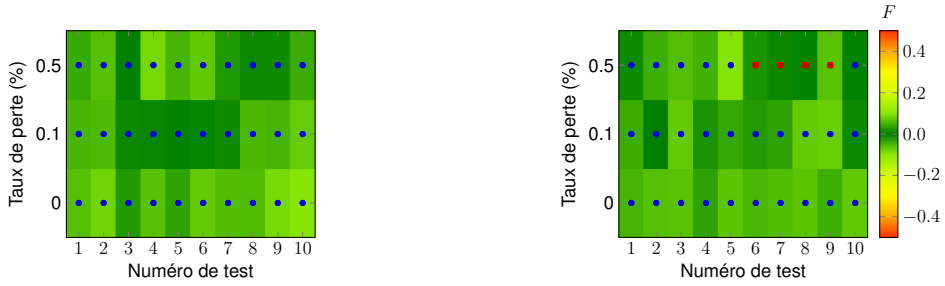


(c) Latence 50ms, Taille du buffer 62.25ko. (d) Latence 300ms, Taille du buffer 62.25ko.

FIGURE 5.8 – Évaluation de l'équité lorsqu'un flux TCP (hystart activée) démarre 8 secondes avant un flux QUIC ($N = 1$, hystart activée) et un débit du point de saturation égal à 1Mbit/s.



(a) Latence 50ms, Taille du buffer 11.25ko. (b) Latence 300ms, Taille du buffer 11.25ko.

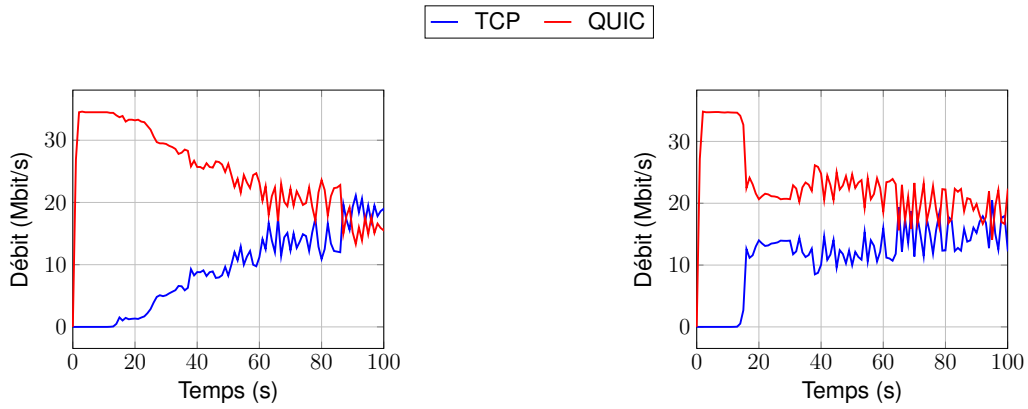


(c) Latence 50ms, Taille du buffer 62.25ko. (d) Latence 300ms, Taille du buffer 62.25ko.

FIGURE 5.9 – Évaluation de l'équité avec flux TCP (hystart activée) commence 8 secondes avant un flux QUIC ($N = 1$, hystart désactivée) et un débit du point de saturation égal à 1Mbit/s.

Dans un premier temps, nous nous consacrons à l'analyse de l'équité sur les réseaux à débit fixe de 1Mbit/s. Les résultats sont visibles sur les figures 5.8. Nous observons l'équité entre les deux protocoles lorsque l'option hystart est activée. Nous constatons un problème d'équité sur les faibles valeurs de taux de perte appliqué sur le réseau. Il est du même ordre de grandeur que sur les réseaux à débit variable. En effet, lorsqu'aucune perte n'est ajoutée, nous remarquons que les valeurs d'équité sont proches de 0.4, ce qui se traduit par une inéquité très forte entre les protocoles. Sur la figure 5.7a, nous représentons de nouveau l'évolution du débit au cours du temps. Nous faisons la même constatation que sur les réseaux à débit variable : lorsque le flux QUIC démarre, la phase de *slow-start* est absente. Pour vérifier que c'est toujours l'option hystart la cause du problème d'équité, nous la désactivons de nouveau sur le protocole QUIC et recommençons les tests. Les nouvelles mesures d'équité sont visibles sur la figure 5.9. Nous constatons maintenant que l'équité est respectée pour l'ensemble des taux de pertes testés. La figure 5.7b montre un test représentatif des faibles valeurs de taux de perte sans l'option hystart sur QUIC. Nous notons que la phase de *slow-start* est présente sur le protocole QUIC, ce qui rétablit l'équité entre les flux.

Maintenant, intéressons-nous à un réseau à débit constant de 34Mbit/s et réalisons de nouveau les mêmes tests. Les résultats sont visibles sur les figures 5.11.



(a) 1 flux TCP et 1 flux QUIC $N = 1$, l'option hystart est activée sur les deux protocoles.

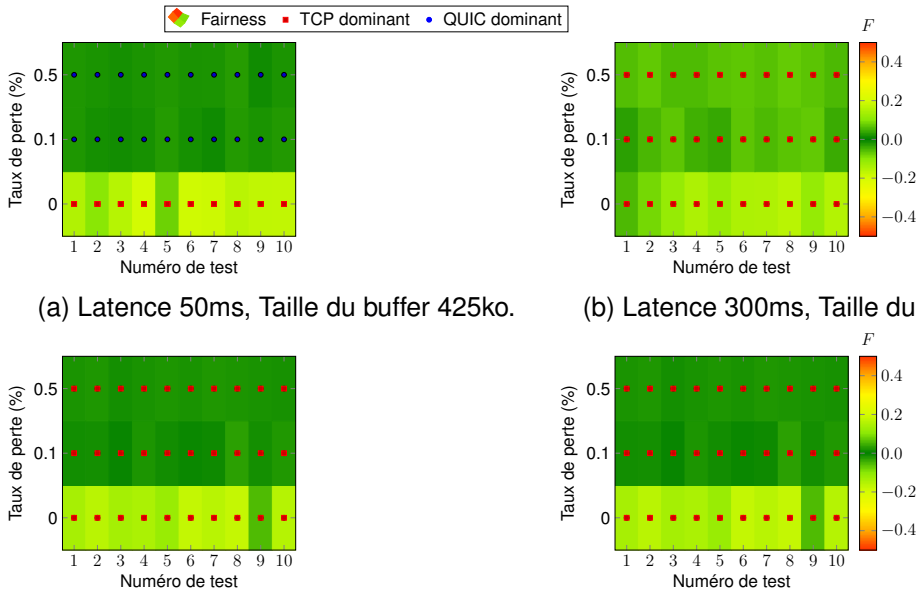
(b) 1 flux TCP et 1 flux QUIC $N = 1$, l'option hystart est désactivée sur les deux protocoles.

FIGURE 5.10 – Effets de l'option hystart sur les protocoles QUIC et TCP sur un réseau à débit constant de 34Mbit/s.

Nous constatons le même effet que sur les deux précédentes configurations de débit : un problème d'équité est présent sur les faibles taux de pertes (même s'il est nettement moins marqué que sur les précédents cas). La figure 5.10a montre l'évolution du débit et nous constatons de nouveau que la phase du *slow-start* est absente lorsque QUIC démarre son trafic, ce qui entraîne de l'inéquité entre les deux flux. Celle-ci est moins marquée, car le protocole QUIC parvient pendant la phase de *congestion avoidance* à augmenter son débit au même

niveau que celui de TCP, chose qui n'était pas possible dans les précédentes configurations. En effet, les deux protocoles obtiennent une bande passante équivalente à TCP au bout de 60 secondes.

Nous désactivons une nouvelle fois l'option hystart dans QUIC-GO (voir figure 5.12) et nous constatons que l'équité est optimale entre les deux flux. La figure 5.10b montre l'évolution du débit, mais cette fois, le flux QUIC obtient une bande passante équivalente à TCP beaucoup plus rapidement grâce à la phase du *slow-start* présente en début de flux.



(a) Latence 50ms, Taille du buffer 425ko.

(b) Latence 300ms, Taille du buffer 425ko.

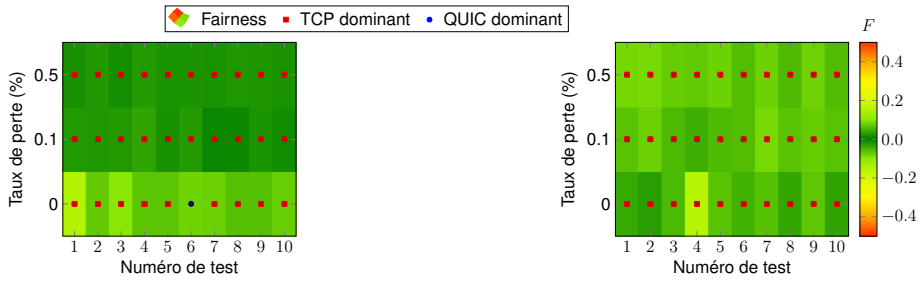
(c) Latence 50ms, Taille du buffer 2.55Mo.

(d) Latence 300ms, Taille du buffer 2.55Mo.

FIGURE 5.11 – Évaluation de l'équité lorsqu'un flux TCP (hystart activée) démarre 8 secondes avant un flux QUIC ($N = 1$, hystart activée) et un débit du point de saturation égal à 34Mbit/s.

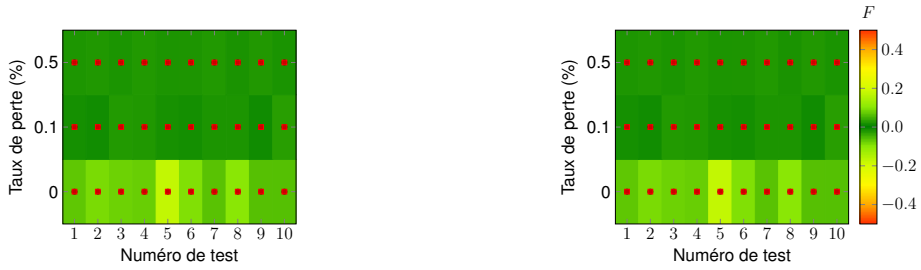
5.4 Analyse de l'impact sur l'équité du nombre de connexions TCP émulées dans QUIC

Dans cette sous-section, nous évaluons l'équité entre une connexion TCP et une connexion QUIC avec différentes valeurs de connexions TCP émulées dans QUIC (N). Le détail de l'implémentation du nombre de connexions TCP émulées dans QUIC est détaillé dans la section 5.1. De plus, nous avons démontré dans la section 5.3 que l'option hystart introduit des problèmes d'équité, c'est pourquoi cette option est désactivée sur les deux protocoles afin de mesurer uniquement l'impact des connections TCP émulées dans QUIC (N).



(a) Latence 50ms, Taille du buffer 425ko.

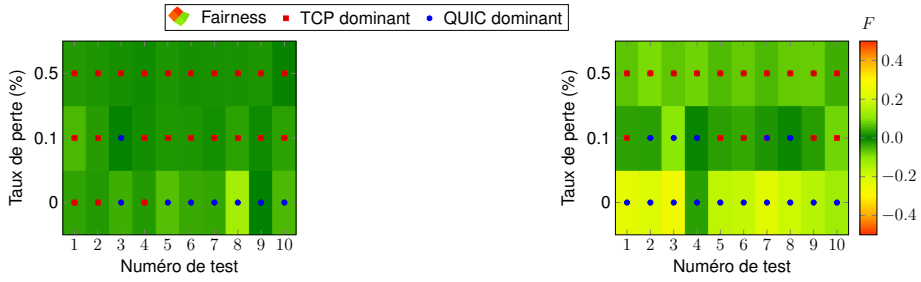
(b) Latence 300ms, Taille du buffer 425ko.



(c) Latence 50ms, Taille du buffer 2.55Mo.

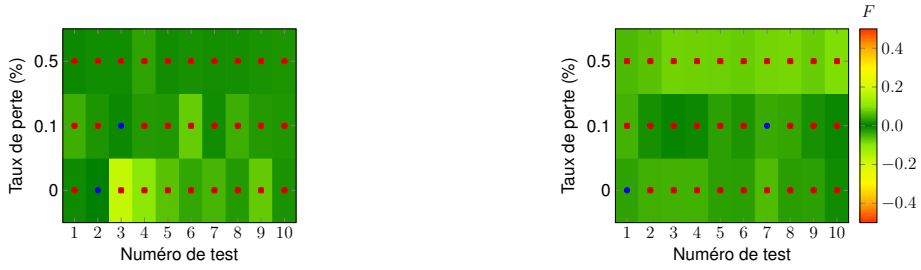
(d) Latence 300ms, Taille du buffer 2.55Mo.

FIGURE 5.12 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) démarre 8 secondes avant un flux QUIC ($N = 1$, hystart désactivée) et un débit du point de saturation égal à 34Mbit/s.



(a) Latence 50ms, Taille du buffer 750ko.

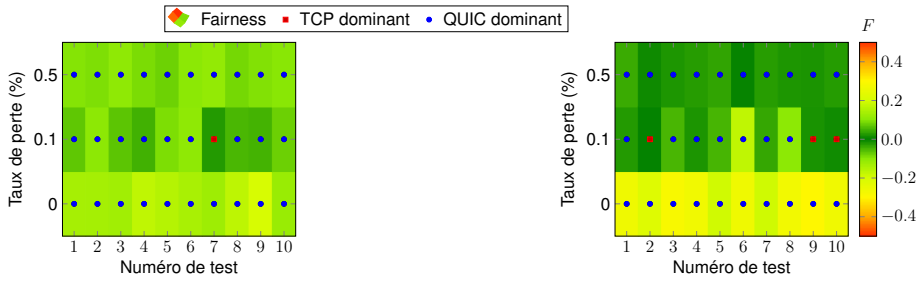
(b) Latence 300ms, Taille du buffer 750ko.



(c) Latence 50ms, Taille du buffer 4.5Mo.

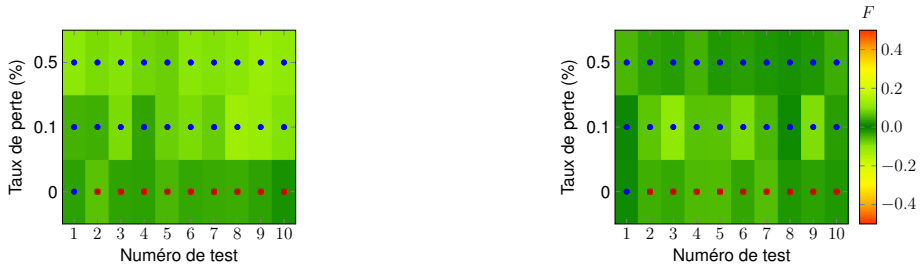
(d) Latence 300ms, Taille du buffer 4.5Mo.

FIGURE 5.13 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 1$, hystart désactivée) avec un point de saturation à débit variable.



(a) Latence 50ms, Taille du buffer 750ko.

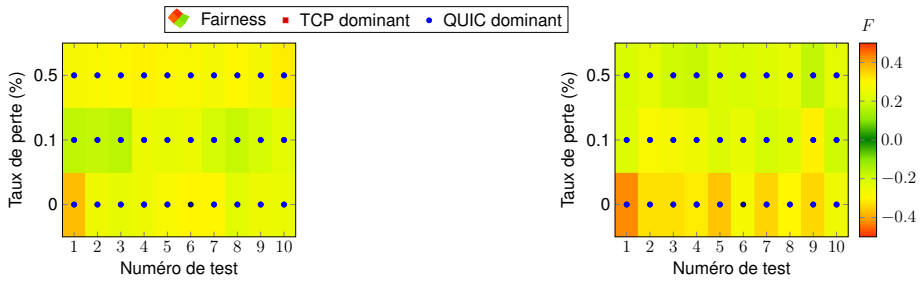
(b) Latence 300ms, Taille du buffer 750ko.



(c) Latence 50ms, Taille du buffer 4.5Mo.

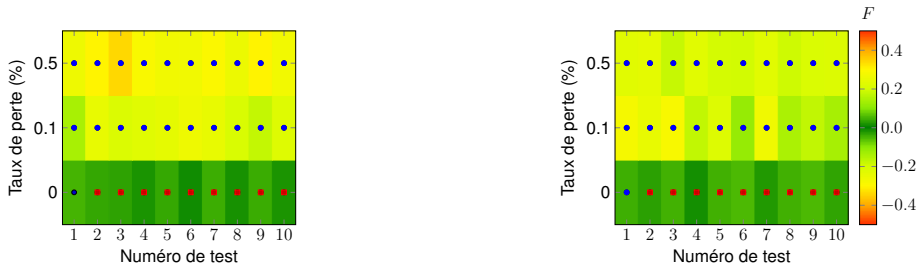
(d) Latence 300ms, Taille du buffer 4.5Mo.

FIGURE 5.14 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 2$, hystart désactivée) avec un point de saturation à débit variable.



(a) Latence 50ms, Taille du buffer 750ko.

(b) Latence 300ms, Taille du buffer 750ko.



(c) Latence 50ms, Taille du buffer 4.5Mo.

(d) Latence 300ms, Taille du buffer 4.5Mo.

FIGURE 5.15 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 5$, hystart désactivée) avec un point de saturation à débit variable.

5.4.1 Réseau à débit variable

Cette section traite des résultats sur un réseau à débit variable, simulant un réseau mobile 4G. La partie réseau à débit constant sera traitée dans la sous-section suivante. Les résultats des tests sont présentés dans les figures 5.13, 5.14 et 5.15.

Dans la figure 5.13, un flux TCP est en concurrence avec un flux QUIC émulant une seule connexion TCP. L'équité est atteinte puisque l'ensemble des indices d'équité est proche de 0. Dans les figures 5.14 et 5.15, lorsque N augmente, l'indice d'équité diminue. En effet, lorsque N est égal à 2, l'indice d'équité moyen est proche de 0.1, et lorsque la valeur de N est égale à 5, l'indice d'équité moyen est proche de 0.3, ce qui caractérise un degré élevé d'inéquité.

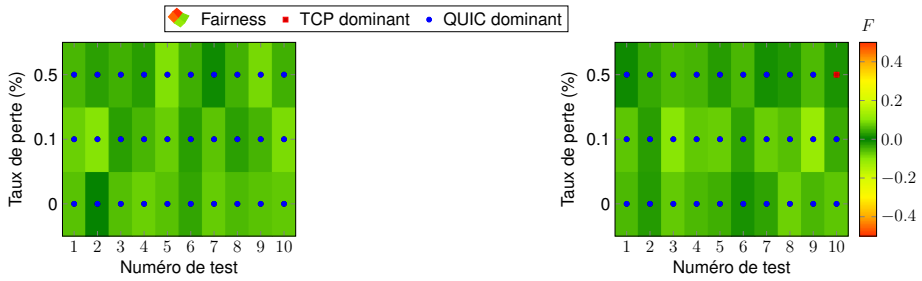
Dans cette section, nous ne prenons pas en compte la taille du *buffer* de 4.5Mo et un taux de perte nul. En effet, sur ces configurations réseau le protocole QUIC ($N = 5$) est équitable avec le protocole TCP. Cette équité, qui peut paraître surprenante, sera analysée dans la section 5.5.

L'inéquité constatée lorsque N augmente s'explique simplement par l'impact de N sur l'évolution de la taille de la fenêtre de congestion dans le temps. En effet, il a été démontré dans la section 5.1 que N influe principalement sur la diminution de la taille de la fenêtre de congestion après un événement de perte : en cas de perte de paquets lorsque $N = 1$ (respectivement $N = 5$) l'algorithme de contrôle de congestion diminue de 30% (respectivement 5%) la taille de cette fenêtre (voir figure 5.1). En d'autres termes, dans les scénarios de test avec des valeurs de N supérieures à 1, la réduction de la fenêtre de congestion due à la perte de paquets est moins sévère pour QUIC que pour TCP ; ce qui implique que QUIC peut maintenir un débit supérieur à TCP même si les deux flux sont soumis au même taux de perte.

5.4.2 Réseau à débit constant

Nous reproduisons la même expérience, mais cette fois-ci avec un réseau à débit constant, toujours sur les mêmes valeurs de débit que lors de l'évaluation de l'impact du hystart (1Mbit/s et 34Mbit/s). Les résultats des mesures d'équité avec les différentes valeurs de N pour un réseau à débit constant de 1Mbit/s et de 34Mbit/s sont respectivement visibles sur les figures 5.16, 5.17, 5.18 et 5.19, 5.20, 5.21.

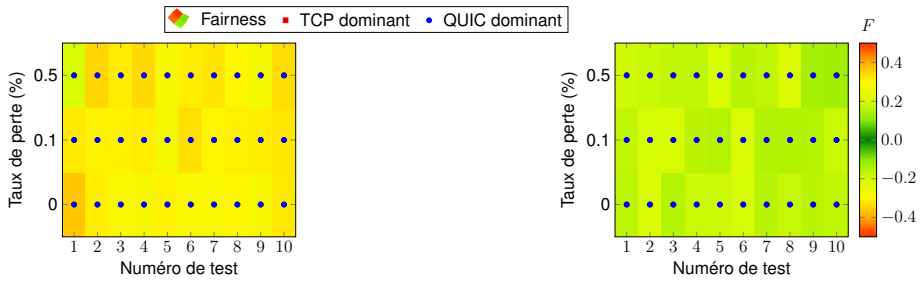
Nous obtenons exactement les mêmes conclusions sur les réseaux à débit constant que sur les réseaux à débit variable. En effet, lorsque la valeur de N augmente, l'équité diminue sur le réseau. Néanmoins, nous notons que pour les valeurs de $N = 2$ sur un réseau possédant un débit constant de 34Mbit/s (voir les figures 5.19 et 5.20), le problème d'inéquité est moins visible. En effet les valeurs d'équité sont proches de 0, mais nous remarquons un changement du protocole dominant donc un changement dans l'équité réseau.



(a) Latence 50ms, Taille du buffer 11.25ko. (b) Latence 300ms, Taille du buffer 11.25ko.

(c) Latence 50ms, Taille du buffer 62.25ko. (d) Latence 300ms, Taille du buffer 62.25ko.

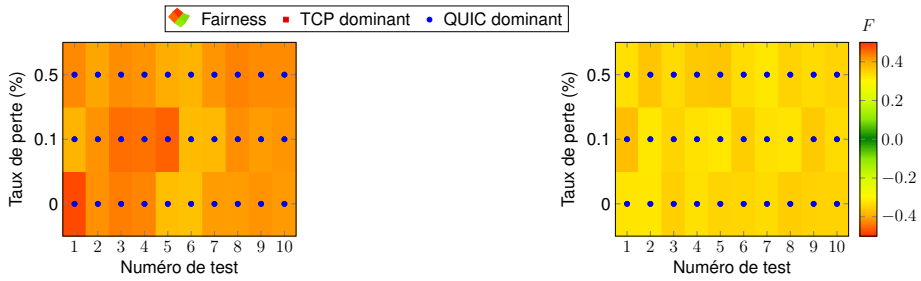
FIGURE 5.16 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 1$, hystart désactivée) avec un point de saturation à débit constant de 1Mbit/s.



(a) Latence 50ms, Taille du buffer 11.25ko. (b) Latence 300ms, Taille du buffer 11.25ko.

(c) Latence 50ms, Taille du buffer 62.25ko. (d) Latence 300ms, Taille du buffer 62.25ko.

FIGURE 5.17 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC ($N = 2$, hystart désactivée) avec un point de saturation à débit constant de 1Mbit/s.



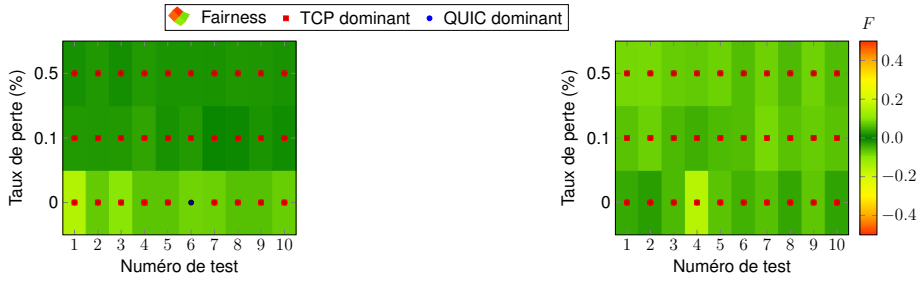
(a) Latence 50ms, Taille du buffer 11.25ko.

(b) Latence 300ms, Taille du buffer 11.25ko.

(c) Latence 50ms, Taille du buffer 62.25Mo.

(d) Latence 300ms, Taille du buffer 62.25ko.

FIGURE 5.18 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 5$, hystart désactivée) avec un point de saturation à débit constant de 1Mbit/s.



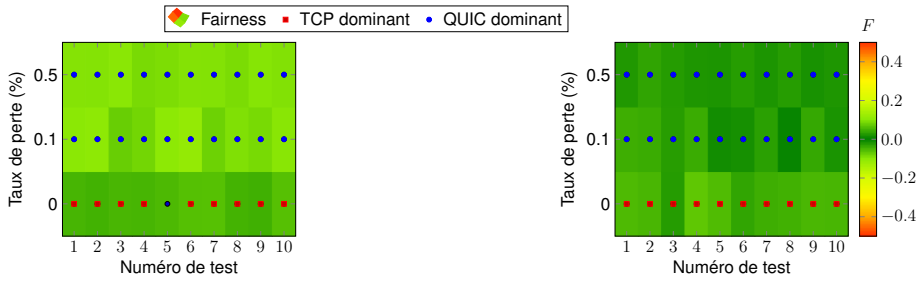
(a) Latence 50ms, Taille du buffer 425ko.

(b) Latence 300ms, Taille du buffer 425ko.

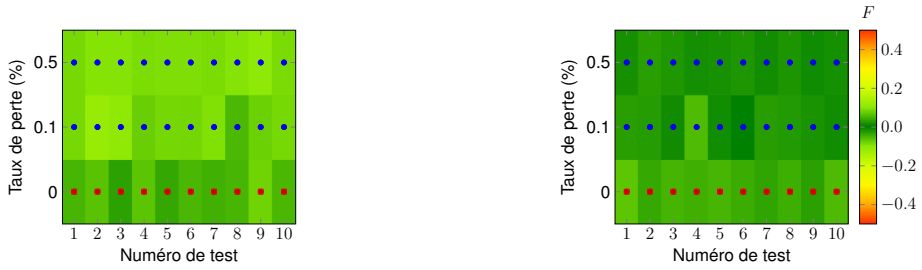
(c) Latence 50ms, Taille du buffer 2.55Mo.

(d) Latence 300ms, Taille du buffer 2.55Mo.

FIGURE 5.19 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 1$, hystart désactivée) avec un point de saturation à débit constant de 34Mbit/s.

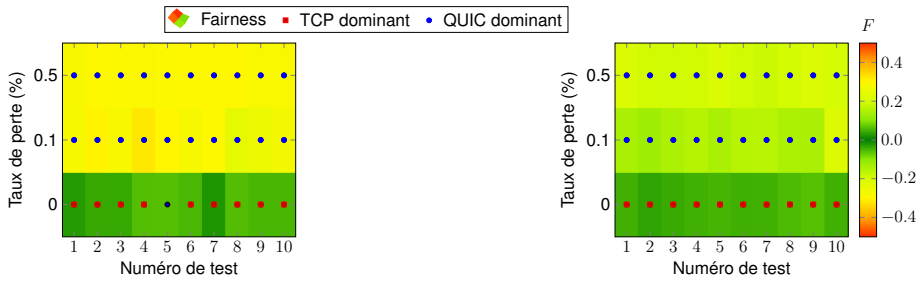


(a) Latence 50ms, Taille du buffer 425ko. (b) Latence 300ms, Taille du buffer 425ko.

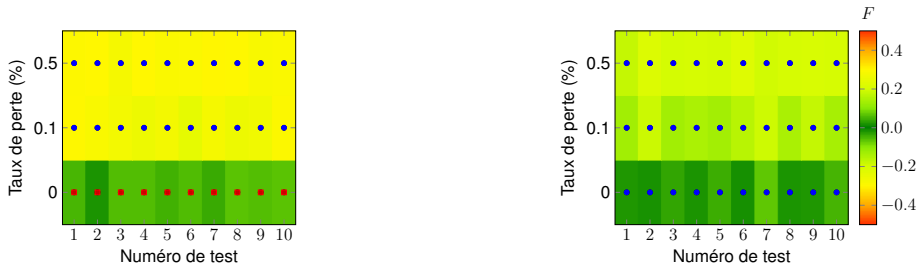


(c) Latence 50ms, Taille du buffer 2.55Mo. (d) Latence 300ms, Taille du buffer 2.55Mo.

FIGURE 5.20 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 2$, hystart désactivée) avec un point de saturation à débit constant de 34Mbit/s.



(a) Latence 50ms, Taille du buffer 425ko. (b) Latence 300ms, Taille du buffer 425ko.



(c) Latence 50ms, Taille du buffer 2.55Mo. (d) Latence 300ms, Taille du buffer 2.55Mo.

FIGURE 5.21 – Évaluation de l'équité lorsqu'un flux TCP (hystart désactivée) commence 8 secondes avant un flux QUIC avec ($N = 5$, hystart désactivée avec un point de saturation à débit constant de 34Mbit/s.

5.5 Analyse de l'impact sur l'équité de la limitation de la taille de la fenêtre de congestion dans QUIC

Dans cette section, nous soulignons l'impact sur l'équité d'un autre paramètre qui caractérise CUBIC : la limitation de la taille de la fenêtre de congestion. Nous nous concentrons sur les configurations réseau qui étaient équitables dans la section 5.4, c'est à dire les cas équitables visibles sur les figures 5.15c, 5.15d et 5.21. Ces différentes situations correspondent à un flux QUIC ($N = 5$) en concurrence avec un flux TCP.

Nous supposons que c'est un autre paramètre qui impacte l'équité sur le réseau : la limitation de la taille de la fenêtre de congestion. Cette limitation a été introduite dans la section 5.1 avec l'équation 5.5. Actuellement cette limite est fixée à 1000 et 2000 paquets dans les implémentations respectives de QUIC-GO et Chromium. Avant de détailler d'avantage l'analyse de l'équité, il est important de noter les points suivants : cette valeur ne change pas au cours de la session et ne dépend pas de la valeur de N. De plus, la limite supérieure de la taille de la fenêtre de congestion n'est pas liée au contrôle de flux. Ce contrôle limite le débit de la source afin de ne pas surcharger le récepteur. Comme il s'agit d'ordinateurs de grande capacité, l'étude actuelle n'est pas impactée par les limitations de débit dues au contrôle du flux.

Pour vérifier que c'est ce paramètre qui impacte bien l'équité, nous recommençons les mêmes tests mais en augmentant de 10 fois la valeur de limitation de la taille de la fenêtre de congestion (soit $MAX_p = 10000$). Les résultats de l'équité sont visibles dans la figure 5.22. Nous constatons que pour toutes les valeurs de taux de perte les deux flux sont inéquitables.

Néanmoins, nous observons que le partage inéquitable, même pour un taux de perte nul et $N = 5$, est visible dans certaines configurations réseau sans modifier la valeur de MAX_p. En effet, sur les figures 5.15a, 5.15b et 5.18 les valeurs sont inéquitables pour l'ensemble des valeurs de taux de pertes. Les conditions de test diffèrent uniquement en termes de taille de *buffer* du goulot d'étranglement. Afin de comprendre ce qui se passe lorsque la taille de la mémoire du *buffer* du goulot d'étranglement est importante (voir figures 5.15c, 5.15d et 5.21), une évolution représentative du débit dans ce cas est présentée à la figure 5.23a. Elle montre que TCP et QUIC obtiennent des débits similaires dus à la valeur par défaut (petite) CWS_{max}. La taille maximale de la fenêtre de congestion pour QUIC, est rapidement atteinte dans un réseau sans perte de paquets. Comme la taille du *buffer* dans les figures 5.15c, 5.15d et 5.21 est assez grande (2.55Mo ou 4.5Mo), alors que le débit pour QUIC est limité à la valeur correspondant à CWS_{max}, le flux TCP peut toujours augmenter sa propre fenêtre de congestion et donc son débit, tant que le goulot d'étranglement n'est pas saturé. Le débordement du *buffer*, donnant lieu à une inéquité, se produit plus rapidement pour un *buffer* de plus petit taille comme dans les figures 5.15a, 5.15b et 5.18.

Afin de confirmer que c'est la combinaison entre une grande taille de *buffer* et une petite valeur de CWS_{max} qui conduit à la situation observée, nous considérons les mêmes paramètres que ceux utilisés pour la figure 5.15c, 5.15d et 5.21, sauf pour CWS_{max} qui est maintenant réglé à une valeur dix fois supérieure ($CWS_{max} = 10000$). Les résultats en matière d'équité sont présentés à la figure 5.22. Dans cette nouvelle configuration de QUIC, le partage injuste entre TCP et QUIC est clairement visible même si aucune perte de paquets n'est simulée.

La figure 5.23b montre un exemple représentatif de l'évolution du débit lorsque le taux de perte est égal à 0% et la taille maximale de la fenêtre de congestion QUIC est égale à 10000. Il montre que le flux TCP est affamé par le flux QUIC qui peut maintenant augmenter significativement sa fenêtre de congestion.

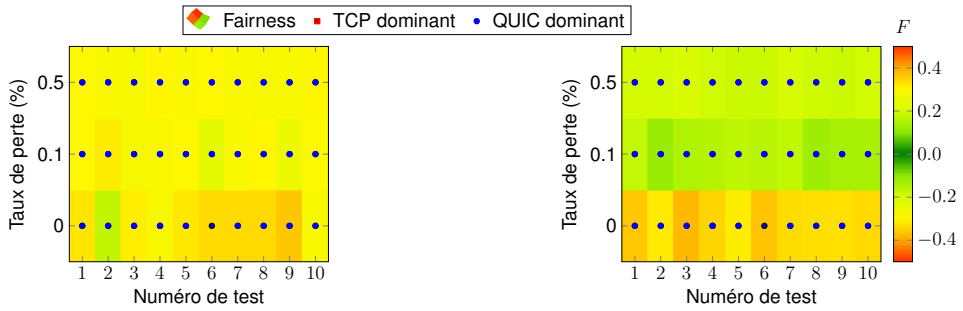
5.6 Bilan sur la comparaison de QUIC versus TCP

QUIC est actuellement déployé sur Internet et est destiné à remplacer TCP et TLS. Il adopte plusieurs options déjà implémentées dans TCP telles que hystart et CUBIC. Toutefois, sa mise en œuvre ne doit pas se faire au détriment de l'équité entre utilisateurs, ce qui est une préoccupation croissante pour les opérateurs, car leurs services à la clientèle sont en première ligne lorsque les performances d'une application internet régressent.

Dans le travail rapporté dans ce chapitre, nous avons analysé comment les implémentations de QUIC peuvent avoir un impact sur l'équité lorsqu'elles sont en concurrence avec un flux TCP sur la liaison sans fil d'un réseau mobile comme sur les réseaux à débit constant. Plusieurs aspects sont analysés au cours des mesures et nous amènent à souligner plusieurs questions d'équité. Premièrement, l'impact de hystart a été mesuré sur les différents flux dans diverses situations sur le réseau mobile. Ensuite, nous avons effectué une mesure d'équité avec un nombre variable (N) de connexions TCP émulées dans QUIC. Enfin, nous avons étudié l'impact sur l'équité de la limitation de la taille de la fenêtre de congestion dans QUIC pour des valeurs élevées de N.

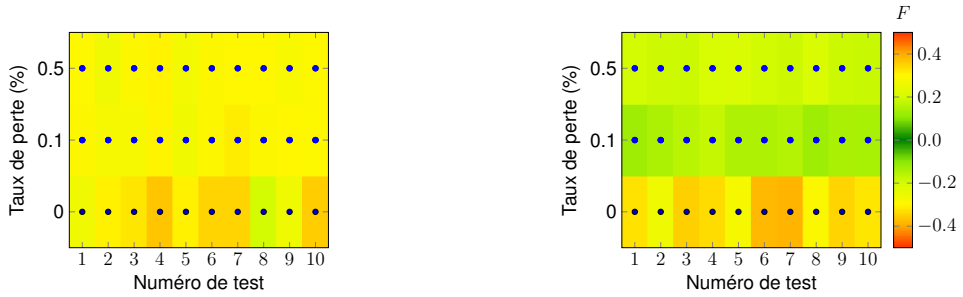
Option hystart

En ce qui concerne l'option hystart présente dans les implémentations TCP et QUIC, les résultats montrent clairement un problème important de partage de capacité réseau. Ce problème survient principalement lorsque les taux de pertes sur le réseau sont faibles. En effet, dans certaines configurations de réseau, cette option empêche le *slow-start* de fonctionner comme il se doit, c'est-à-dire en facilitant une montée en puissance rapide du débit soit au moment du lancement, soit après un *timeout* de la connexion. Le comportement de l'option hystart peut donc réduire considérablement le débit de l'utilisateur pendant toute la session et créer ainsi un réel problème en termes de QoE. Nous recommandons donc que cette option soit



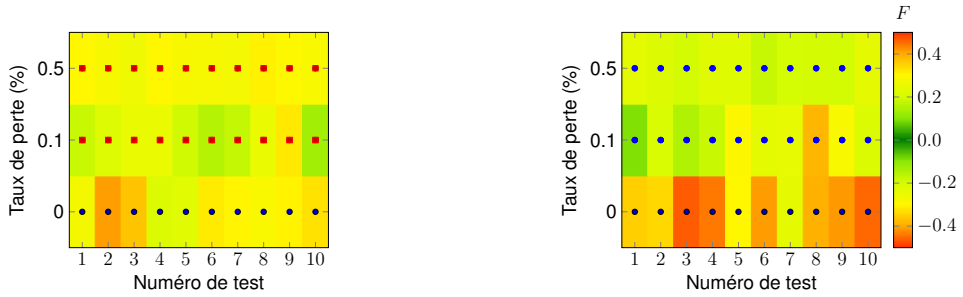
(a) Débit constant à 34Mbit/s, Latence 50ms, Taille du buffer 425ko.

(b) Débit constant à 34Mbit/s, Latence 300ms, Taille du buffer 425ko.



(c) Débit constant à 34Mbit/s, Latence 50ms, Taille du buffer 2.55Mo.

(d) Débit constant à 34Mbit/s, Latence 300ms, Taille du buffer 2.55Mo.

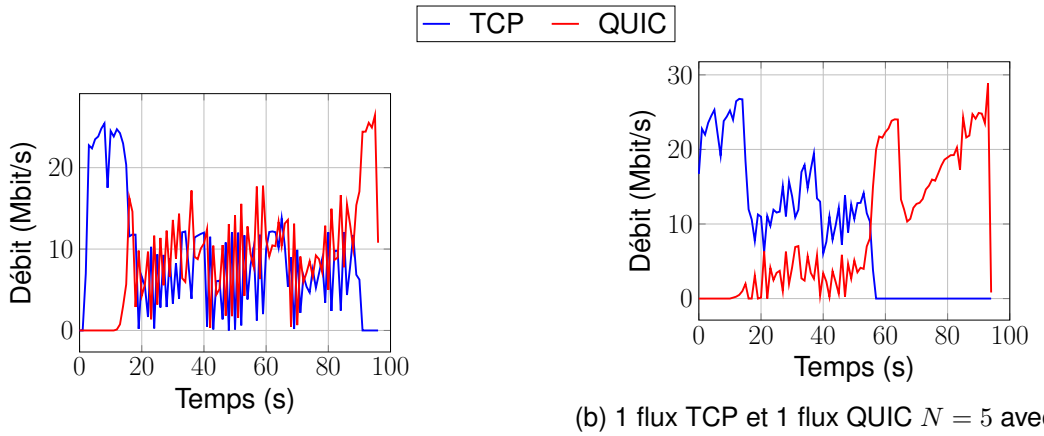


(e) Débit variable, Latence 50ms, Taille du buffer 4.5Mo.

(f) Débit variable, Latence 300ms, Taille du buffer 4.5Mo.

FIGURE 5.22 – Évaluation de l'équité lorsqu'un flux TCP commence 8 secondes avant un flux QUIC avec ($N = 5$), l'option hystart désactivée sur les deux protocoles et l'augmentation de la taille maximale de la fenêtre de congestion de QUIC à 10000 paquets ($CWS_{max} = 10000$).

désactivée ou significativement améliorée (en n'interprétant pas systématiquement l'augmentation du RTT comme une congestion naissante) sur QUIC et TCP, pour assurer une bonne utilisation et équité du réseau.



(a) 1 flux TCP et 1 flux QUIC $N = 5$ avec la valeur par défaut de CWS_{max} (1000 paquets).

(b) 1 flux TCP et 1 flux QUIC $N = 5$ avec la valeur de CWS_{max} supérieur de $10x$ à celle par défaut (10000 paquets).

FIGURE 5.23 – Effets de la limitation de la taille de la fenêtre de congestion, lorsque le taux de perte est égal à 0%.

Nombre de connexions TCP émulées dans QUIC

Ces résultats rapportés ici montrent que la valeur de N a un impact réel sur le comportement de l'algorithme de contrôle de congestion de QUIC et sur l'équité entre protocoles. Nous avons montré ici que des problèmes sont visibles sur les réseaux mobiles. Lorsque $N = 1$, les protocoles TCP et QUIC partagent équitablement la bande passante disponible dans toutes les configurations réseau testées. Mais quand la valeur de N augmente alors l'équité diminue considérablement en faveur de QUIC. C'est-à-dire que le protocole QUIC obtient une part de débit beaucoup plus élevée que le protocole TCP. Le paramètre N est actuellement statique dans les implémentations QUIC et est par défaut fixé à 2. Cependant, il peut être facilement modifié et incrémenté à des valeurs plus élevées dans les applications clientes et serveurs. Il est donc essentiel de pouvoir définir des règles d'ingénierie pour adapter dynamiquement cette valeur dans le temps en fonction des demandes des utilisateurs (par exemple le nombre de *stream*s dans QUIC) et des conditions réseau pour assurer un comportement équitable des protocoles sur le réseau. Nous sommes d'avis que des lignes directrices de mise en œuvre et la surveillance des usages seraient bénéfiques pour la majorité.

Limitation de l'augmentation de la taille de la fenêtre de congestion

Cette chapitre a également étudié l'impact de la limite de la taille de la fenêtre de congestion par l'algorithme de contrôle de congestion CUBIC dans QUIC. Pour QUIC, cette valeur est constante pendant la session et ne dépend pas de N. Nous montrons que lorsque les valeurs de N augmentent, la limitation de la taille de la fenêtre de congestion empêche le protocole

QUIC d'augmenter son débit. Cette limitation ne permet donc pas l'émulation de plusieurs connexions TCP lorsque le taux de perte sur le réseau est faible. En outre, elle entrave le principe du multiplexage de connexions d'applications multiples dans le protocole QUIC.

Chapitre 6

Équité des algorithmes de contrôle de congestion

Contents

Depuis l'essor d'Internet, les chercheurs ont mis au point des protocoles et des algorithmes pour accroître l'efficacité globale du réseau tout en préservant l'équité entre les utilisateurs ou les applications simultanées et en prévenant les saturations du réseau. Dans ce chapitre, nous sommes particulièrement intéressés par les algorithmes de contrôle de congestion, qui sont implémentés dans la couche transport de la pile du protocole d'Internet. Les plus populaires sont RENO [10] et CUBIC [11] qui identifient les congestions par la détection de perte de paquets. Ces algorithmes permettent d'ajuster le débit parmi les flux réseau simultanés ; cependant, les algorithmes de contrôle de congestion basés sur la perte de paquets ne répondent pas aux exigences de latence des services réseau entrants telles que les applications en temps réel, les *slice*s à faible latence sur les réseaux 5G, la réalité virtuelle, etc.

De récents efforts de recherche ont amené à de nouveaux algorithmes de contrôle de congestion, en particulier BBR [12, 13] et PCC [14]. Ces deux propositions visent à améliorer la performance du réseau en maximisant à la fois le débit du réseau et la "satisfaction" des utilisateurs. BBR et PCC diffèrent de RENO et CUBIC car ils ne comptent plus sur la détection des pertes pour détecter la congestion sur le réseau. BBR est basé sur l'évolution de la latence alors que PCC se fonde sur la prédiction de flux.

Dans un premier temps, nous présentons les scénarios de tests pour évaluer l'équité entre les différents algorithmes de contrôle de congestion. Dans la deuxième section, nous évaluons trois types d'algorithmes de contrôle de congestion (RENO, BBR et PCC). Enfin dans la dernière partie, nous cherchons à déterminer le nombre de tests nécessaire dans des situations réseau différentes pour déterminer les valeurs d'équité entre les algorithmes de contrôle de congestion.

6.1 Scénarios de tests

Dans cette section, nous présentons les différents paramètres de tests appliqués sur la plateforme décrite dans le chapitre 3. Dans ce chapitre, nous utilisons la première version de la partie réseau de la plateforme (voir section 3.1.2) pour réaliser les tests d'équité entre les différents algorithmes de contrôle de congestion. Ainsi, nous étudions l'équité entre différents algorithmes de contrôle de congestion, tel que RENO, BBR et PCC. Les deux premiers sont basés sur le protocole de transport TCP, tandis que le troisième utilise le protocole de transport UDP-based Data Transfer Protocol (UDT) basé sur UDP. L'ensemble de ces tests sont exécutés sur un réseau à débit constant.

Nous appliquons sur la plateforme, une combinaison exhaustive de scénarios de tests. L'ensemble de ces valeurs est présenté dans le tableau 6.1. Les différents tests ont été obtenus grâce à chaque combinaison des paramètres réseau, à savoir le taux de perte, la latence, la taille du *buffer* et les *queuing policies*.

Latence (ms)	0, 1, 50, 150, 300
Taux de pertes (%)	0, 1, 5, 10, 20
Taille du buffer (octets)	8750, 37500, 62500, 125000, 250000
Queuing policies	FIFO, FQ_CODEL, RED

Tableau 6.1 – Caractéristiques du réseau appliqué sur la plateforme de tests pour la comparaisons des algorithmes de contrôle de congestion.

Les paramètres réseau ont été obtenus en accord avec le draft IETF suivant [29]. De plus, nous définissons un point de saturation à 1Mbit/s et la taille des fichiers échangés par chaque protocole sont définis à 10Mo de façon à ce que la période de concurrence soit égale ou supérieure à 60 secondes. L'ensemble de ces tests représente un nombre important de configurations réseau à exécuter. C'est pourquoi, dans la dernière partie de ce chapitre, nous chercherons à savoir, si avec moins de tests, nous pouvons obtenir les mêmes résultats.

À partir des mesures obtenues sur la plateforme, nous appliquons les deux méthodes de mesures d'équité introduites dans le chapitre 4. Ces deux métriques nommées SFA et WSFA permettent de mesurer l'équité sur l'ensemble de la période de concurrence entre les différents

flux. Pour rappel, la principale différence entre les deux métriques est la pondération des moments de la session. Dans la première métrique, tous les moments sont pondérés de la même façon, tandis que pour la méthode WSFA le début de la session a beaucoup plus d'importance que la fin.

6.2 Études des algorithmes de contrôle de congestion avec eux-mêmes

Dans cette section, nous étudions l'équité lorsque l'on met des algorithmes de contrôle de congestion en compétition avec eux-même. Pour cela, nous les testons dans l'ensemble des configurations réseau présenté dans la section 6.1. Nous commençons d'abord par analyser l'équité avant de détailler l'impact des paramètres réseau sur celle-ci.

6.2.1 Études d'équité des algorithmes de contrôle de congestion avec eux-mêmes

Cette section permet de vérifier l'équité lorsque les algorithmes de contrôle de congestion sont en compétition avec eux-mêmes. Le tableau 6.2 représente le nombre de tests (un test correspond à une configuration réseau) de chaque combinaison des algorithmes de contrôle de congestion en fonction du protocole dominant et des plages de valeurs d'équité. Sur le tableau, nous notons sur la première ligne, le nom des algorithmes de contrôle de congestion testés. Par la suite, le tableau est divisé en deux parties : les premières lignes de résultats correspondent aux valeurs d'équité lorsque l'algorithme 1 domine (voir la première ligne de la première ligne du tableau pour connaître le nom de l'algorithme dominant) et la seconde partie correspond aux mesures d'équité lorsque l'algorithme 2 domine (se référer à la seconde ligne de la première ligne du tableau pour connaître le nom de l'algorithme dominant).

À partir de ces résultats, nous constatons une très forte équité quand les algorithmes de contrôle de congestion sont en concurrence avec eux-mêmes. Néanmoins, lorsque l'on mesure l'équité avec la méthode SFA, les valeurs inéquitables sont présentes lorsque plusieurs flux BBR ou flux PCC sont en concurrence avec eux-mêmes. Au contraire, avec la méthode de calcul de l'équité WSFA ces valeurs d'inéquité ne sont pas présentes. On peut donc conclure que les flux sont équitables en début de session et qu'un évènement vient perturber l'équité entre ces flux. Pour cela, nous représentons les données en fonction des valeurs de latence, de taux de perte, de taille de *buffer* et des *queuing policies*.

Algorithme 1 Algorithme 2		RENO RENO	BBR BBR	PCC PCC	RENO RENO	BBR BBR	PCC PCC
Equité		SFA			WSFA		
Algorithme 1	-0.5 à -0.4	0	1	3	0	0	0
	-0.4 à -0.3	0	0	0	0	0	0
	-0.3 à -0.2	0	4	0	0	0	3
	-0.2 à -0.1	0	18	18	15	22	9
	-0.1 à 0	197	168	160	181	160	115
Algorithme 2	0 à 0.1	177	162	170	165	171	232
	0.1 à 0.2	1	13	13	14	22	9
	0.2 à 0.3	0	7	2	0	0	7
	0.3 à 0.4	0	2	2	0	0	0
	0.4 à 0.5	0	0	7	0	0	0

Tableau 6.2 – Valeur d'équité lorsque des algorithmes de contrôle de congestion sont en concurrence avec eux-mêmes.

6.2.2 Analyse de l'impact des paramètres réseaux

Nous montrons sur les figures 6.1, 6.2, 6.3 et 6.4 l'évaluation de l'équité avec la méthode SFA en fonction des paramètres réseau. Nous représentons la latence, le taux de perte, la taille du *buffer* et les *queuing policies* sur l'axe des abscisses et le nombre d'occurrences des différentes plages des valeurs d'équité sur l'axe des ordonnées. L'axe des abscisses est divisé en deux parties : la partie gauche représente les valeurs d'équité dont le premier algorithme domine (obtient plus de débit), et la partie droite représente les valeurs d'équité lorsque le second algorithme est dominant.

En ce qui concerne les représentations de l'équité en fonction des configurations du réseau avec la méthode WSFA, elles sont représentées en annexe A car les flux sont intégralement équitables avec l'ensemble des paramètres réseau.

À partir des figures 6.1 et 6.2, nous déterminons que les problèmes d'équité sont essentiellement présents lorsque le taux de perte est important et que la latence est faible sur le réseau. En effet, les valeurs inéquitables sont présentes pour des taux de perte supérieurs à 5% et des latences inférieures à 50ms.

Pour conclure, nous pouvons déterminer que les algorithmes de contrôle de congestion testés sont équitables lorsqu'ils sont en compétition avec eux-mêmes. Dans la section suivante, nous analyserons l'équité lorsque nous mettons les algorithmes de contrôle de congestion en compétition entre-eux.

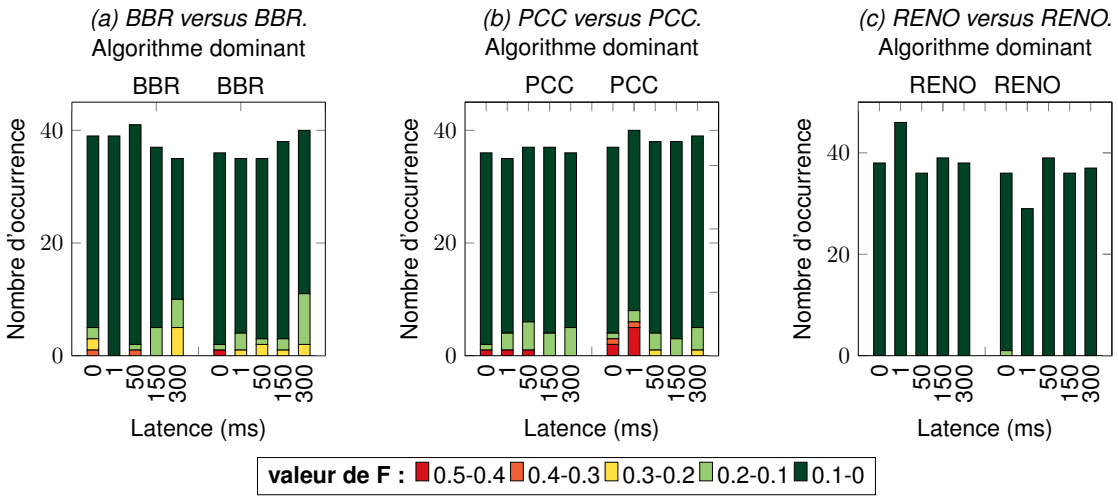


FIGURE 6.1 – Comparaison des algorithmes de contrôle de congestion avec eux-mêmes en les classant par latence. L'équité est évaluée avec la méthode SFA.

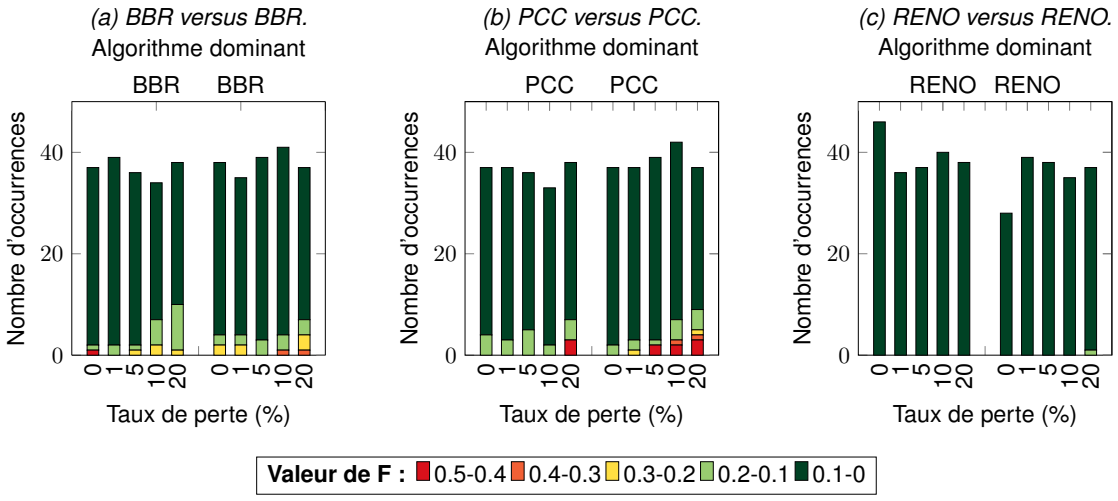


FIGURE 6.2 – Comparaison des algorithmes de contrôle de congestion avec eux même en les classant par taux de perte. La fairness est évalué avec la méthode SFA.

6.3 Études d'équité des algorithmes de contrôle de congestion entre-eux

Dans cette sous-section, nous nous concentrons sur l'analyse de l'équité lorsque différents algorithmes de contrôle de congestion sont en concurrence. Pour cela, nous les testons lorsqu'ils sont en compétitions entre-eux dans l'ensemble des configurations réseaux détaillées dans la section 6.1.

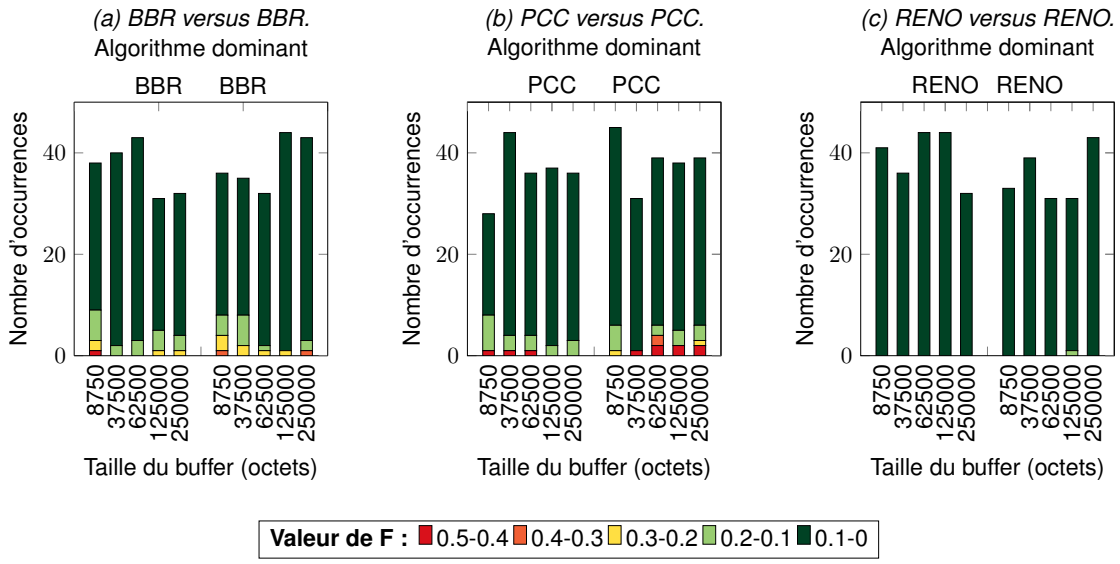


FIGURE 6.3 – Comparaison des algorithmes de contrôle de congestion avec eux-mêmes en les classant par taille de buffer. L'équité est évaluée avec la méthode SFA.

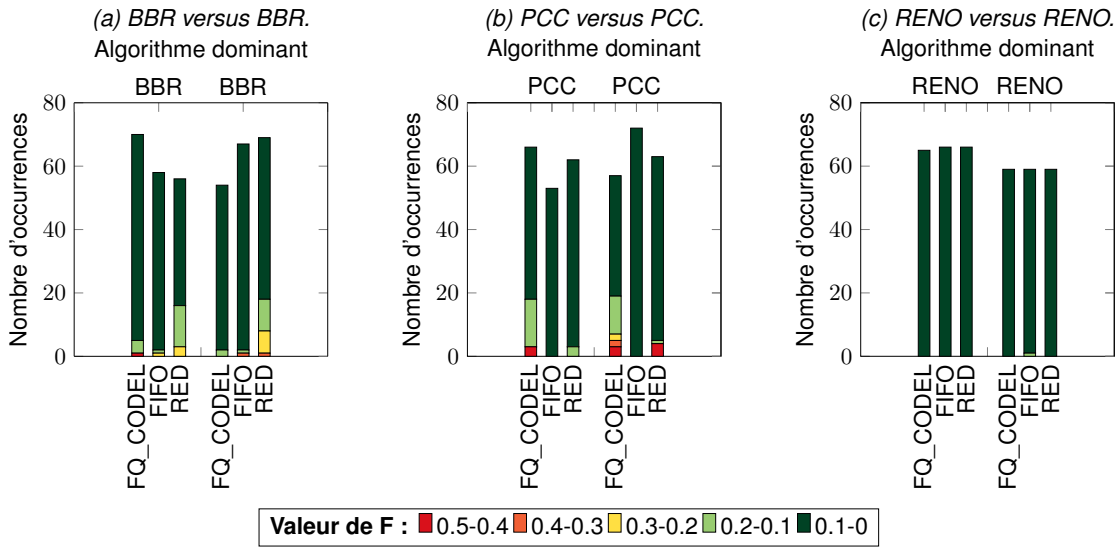


FIGURE 6.4 – Comparaison des algorithmes de contrôle de congestion avec eux-mêmes en les classant par queueing policies. L'équité est évaluée avec la méthode SFA.

6.3.1 Analyse des valeurs d'équité

Dans un premier temps, nous analysons les valeurs d'équité dans l'ensemble des situations réseau. Les valeurs d'équité sont présentées dans le tableau 6.3. Les résultats nous montrent une nette inéquité entre les différents algorithmes de contrôle de congestion avec la méthode d'évaluation d'équité SFA. En effet, lorsque l'algorithme de contrôle de congestion PCC est

utilisé, les indices d'équité nous démontrent qu'il a une tendance très nette à obtenir plus de débit que les deux autres (RENO et BBR). En effet, nous le remarquons avec les indices d'équité qui sont proche de la valeur limite ($|0.5|$) en faveur de l'algorithme PCC. En ce qui concerne les algorithmes de contrôle de congestion RENO versus BBR, l'inéquité entre les deux algorithmes est moins marquée que lorsque PCC est utilisé, mais elle est quand même bien présente entre les flux en faveur de BBR. En effet, beaucoup de valeurs d'équité sont comprises dans la plage intermédiaire de la métrique (comprise entre 0.1 et 0.4).

En ce qui concerne l'évaluation de l'équité avec la méthode WSFA, qui donne plus d'importance au début de la session qu'à la fin, les mêmes constats sur les algorithmes de contrôle de congestion dominant peuvent être réalisé qu'avec la méthode d'évaluation SFA. Nous remarquons néanmoins une amélioration de l'équité entre les deux flux avec des valeurs plus proches de 0.1 à 0.3. Ceci s'explique par le fait que les algorithmes de contrôle de congestion sont équitables durant la phase de *slow-start* et que l'inéquité arrive plus tard dans la connexion.

Algorithme 1 Algorithme 2		BBR RENO	PCC BBR	RENO PCC	BBR RENO	PCC BBR	RENO PCC
Fairness		SFA			WSFA		
Algorithme 1	-0.5 à -0.4	42	148	34	2	14	0
	-0.4 à -0.3	102	35	1	25	43	0
	-0.3 à -0.2	111	43	2	82	64	0
	-0.2 à -0.1	75	61	4	168	106	0
	-0.1 à 0	25	76	2	75	73	11
Algorithme 2	0 à 0.1	20	11	5	18	70	88
	0.1 à 0.2	0	0	36	5	5	111
	0.2 à 0.3	0	0	39	0	0	77
	0.3 à 0.4	0	1	86	0	0	74
	0.4 à 0.5	0	0	166	0	0	14

Tableau 6.3 – Valeur d'équité lorsque des algorithmes de contrôle de congestion sont en concurrence entre-eux.

Nous pouvons en conclure que les algorithmes RENO, PCC et BBR ne sont pas toujours équitables lorsqu'ils sont mis en concurrence entre-eux. Mais, nous démontrons qu'il est toujours possible de trouver des configurations réseau où ces protocoles sont équitables entre-eux, partiellement équitables ou complètement inéquitables. Ce qui démontre qu'un grand nombre de configurations réseau est nécessaire pour évaluer l'équité entre plusieurs algorithmes de contrôle de congestion. Nous aborderons ce point-là dans la section 6.4, qui déterminera le nombre de tests nécessaire pour évaluer l'équité réseau entre deux algorithmes de contrôle de congestion. Mais pour le moment, nous cherchons à comprendre l'impact des

paramètres réseau sur l'équité. Pour cela, nous réutilisons la même représentation graphique que dans la section 6.2 pour représenter les résultats en fonction de la latence, du taux de pertes, de la taille du *buffer* et des *queuing policies*.

6.3.2 Analyse en fonction des paramètres réseau

Dans cette sous-section, nous nous focalisons uniquement sur la méthode d'évaluation SFA pour évaluer l'équité sur le réseau. En ce qui concerne l'évaluation de l'équité avec la méthode WSFA, elles sont représentées en annexe B car elles possèdent exactement les mêmes conclusions que celles présentées ci-dessous sauf que l'on possède des valeurs plus équitables.

Évaluation de l'impact de la latence sur l'équité

Dans un premier temps, nous nous attachons à mesurer l'impact sur l'équité de la latence. La figure 6.5 nous montre les mesures d'équité en fonction de la latence représentée sur l'axe des abscisses.

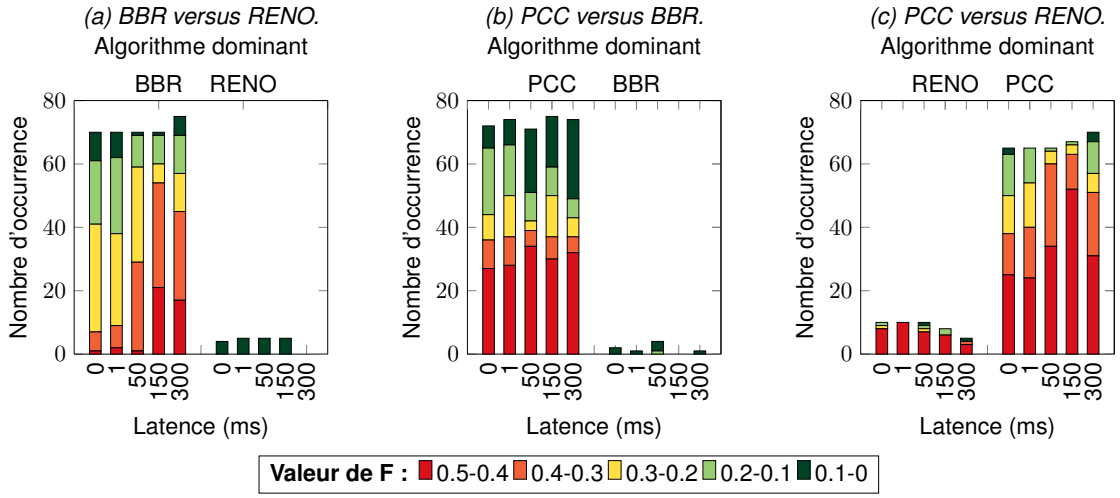


FIGURE 6.5 – Comparaison des algorithmes de contrôle de congestion entre-eux en les classant par latence. L'équité est évaluée avec la méthode SFA.

Nous constatons alors deux comportements distincts qui correspondent à la présence ou non de l'algorithme de contrôle de congestion PCC. Dans le cas où il est présent, nous constatons une forte dominance de celui-ci sur les deux autres algorithmes : RENO et BBR. Cette constatation est vraie pour l'ensemble des valeurs de latence. En d'autres termes, nous remarquons que les différentes valeurs de latences n'ont pas d'impact sur la répartition des valeurs d'équité lorsque PCC est utilisé sur le réseau. Ceci s'explique par le comportement de cet algorithme qui ne prend pas en compte la latence pour évaluer la taille de sa fenêtre de congestion

(débit). En ce qui concerne l'évaluation de la latence lorsque RENO et BBR sont en compétition, nous constatons une augmentation de l'inéquité lorsque la valeur de la latence augmente. En effet, lorsqu'une valeur de latence est importante sur le réseau, l'algorithme de contrôle de congestion a plus de difficulté pour détecter les pertes sur son flux et donc à déterminer précisément la bande passante disponible sur le réseau. Ceci explique pourquoi l'augmentation de la latence permet à BBR d'obtenir plus de débit que RENO. À noter que ce phénomène est moins visible lorsque PCC et RENO sont en concurrence, du fait que PCC laisse beaucoup moins de débit à RENO que BBR.

Évaluation de l'impact du taux de perte sur l'équité

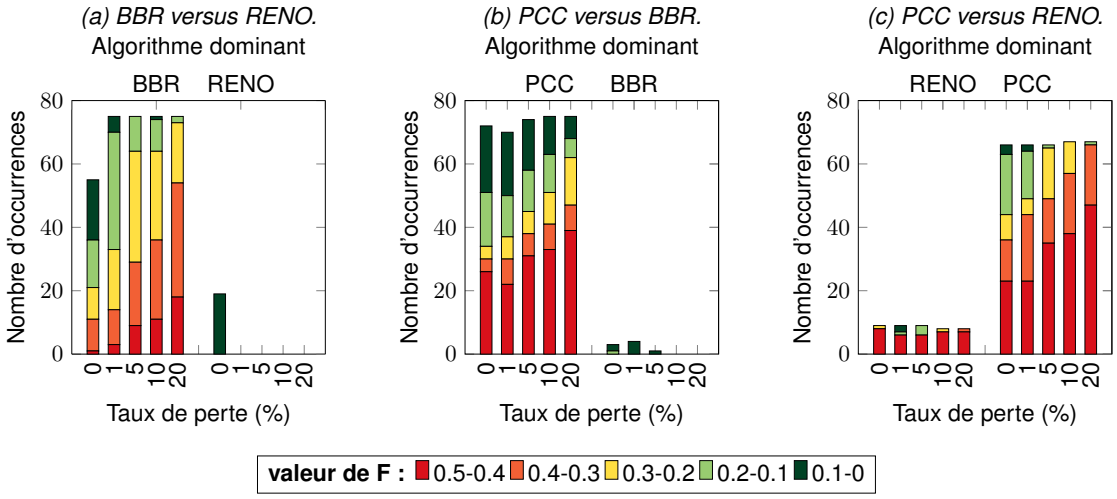


FIGURE 6.6 – Comparaison des algorithmes de contrôle de congestion entre eux en les classant par taux de perte. L'équité est évaluée avec la méthode SFA.

Nous étudions maintenant l'impact sur l'équité du taux de perte ajouté sur le réseau. Les résultats sont visibles sur la figure 6.6. Pour l'ensemble des combinaisons d'algorithme de contrôle de congestion testés, nous constatons que les valeurs d'équité sont de plus en plus inéquitables lorsque le taux de perte augmente. Donc l'équité est dégradée lorsque le pourcentage de pertes de paquets augmente sur le réseau. Lorsque l'algorithme de contrôle de congestion RENO est utilisé, il diminue systématiquement la taille de la fenêtre de congestion (débit) du flux à chaque perte de paquets. Donc lorsque le taux de perte augmente, le débit de RENO n'arrive pas à augmenter suffisamment pour être équitable avec les autres algorithmes de contrôle de congestion qui ne sont pas basés sur la perte de paquets pour détecter une congestion (dans notre cas BBR et PCC). Lorsque les algorithmes de contrôle de congestion PCC et BBR sont en concurrence, les résultats montrent toujours une iniquité croissante quand le taux de pertes augmente. Ceci s'explique du fait que PCC ne réagit pas aux pertes de pa-

quets sur le réseau et qu'il cherche à maximiser son propre débit. Par conséquent, lorsque le taux de perte augmente, PCC ne change pas son comportement tandis que BBR a de plus en plus de difficultés à estimer la bande passante disponible en raison de l'augmentation du RTT.

Évaluation de l'impact de la taille du buffer sur l'équité

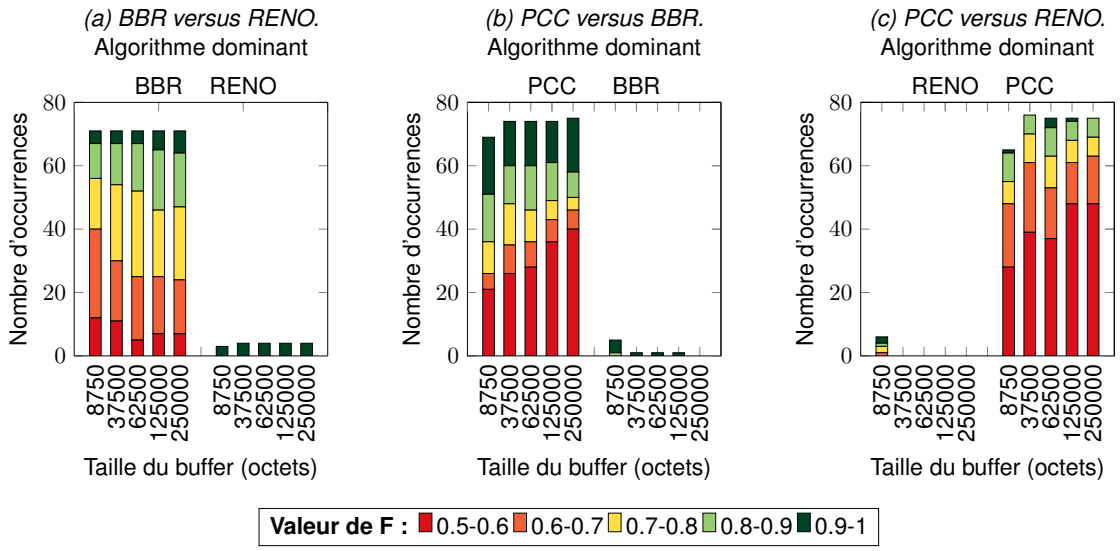


FIGURE 6.7 – Comparaison des algorithmes de contrôle de congestion entre eux en les classant par taille de buffer. L'équité est évaluée avec la méthode SFA.

Dans cette analyse, nous nous concentrons sur l'impact de la taille du *buffer* du point de saturation sur le réseau. Les résultats sont visibles sur la figure 6.7. Nous constatons sur l'ensemble des comparaisons d'algorithme de contrôle de congestion (BBR versus RENO, PCC versus BBR et PCC versus RENO) que l'équité n'est pas impactée par la taille du buffer. En effet, le même nombre de valeurs est présent dans les différentes plage d'équité sur l'ensemble des tailles de *buffer*.

Évaluation de l'impact des queuing policies sur l'équité

Dans cette analyse, nous nous focalisons sur l'analyse des règles de *queuing policies* du *buffer* du point de saturation sur le réseau. Pour cela, nous représentons les trois *queuing policies* sur l'axe des abscisses de la figure 6.8.

Nous constatons que les *queuing policies* sont l'un des paramètres à avoir un impact significatif sur les mesures d'équité. Toutes les interprétations effectuées ci-après sont valables pour l'ensemble des combinaisons des algorithmes de contrôle de congestion observé dans cette analyse. La *queuing policy* FIFO est la moins équitable des trois politiques de file d'attente testées. Ceci s'explique par le fait que FIFO ne priorise aucun des flux et délivre les paquets

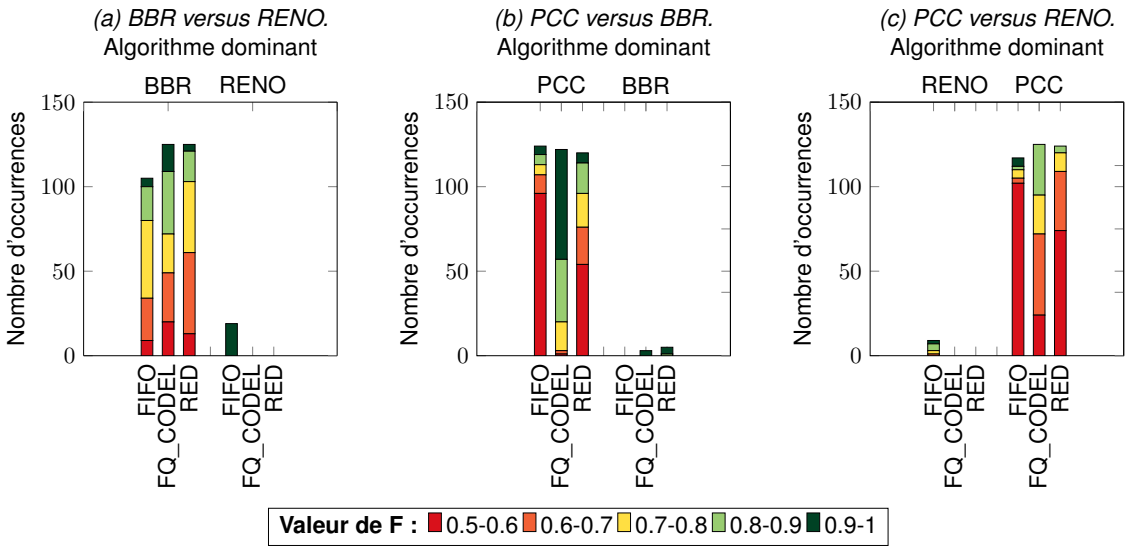


FIGURE 6.8 – Comparaison des algorithmes de contrôle de congestion entre eux en les classant par les différentes queuing policies. L'équité est évaluée avec la méthode SFA.

mis en attente par ordre d'arrivée. De plus, lorsque celui-ci est plein il supprime les paquets les plus récents sans tenir compte du flux auquel ils appartiennent. La *queuing policy* RED est plus équitable que FIFO car la stratégie de suppression des paquets n'est plus basée sur des choix aléatoires mais sur une probabilité. En effet, lorsque le *buffer* arrive en saturation, RED choisira de supprimer le paquet du flux qui a la probabilité la plus importante d'être le flux dominant. Cette solution n'est efficace que pour les algorithmes de contrôle de congestion qui sont basés sur la perte de paquets. Puisque PCC et BBR ignorent la perte de paquets, cette *queuing policy* n'est pas efficace. Enfin, la dernière *queuing policy* fq_codel est la plus juste. En effet, l'algorithme *CoDel* permet de mieux répartir les paquets des différents flux dans le *buffer* et permet alors de supprimer de façon plus efficace l'inéquité entre les différents flux.

6.4 Définition du nombre de tests pour l'évaluation de l'équité

Dans cette sous-section, nous nous intéressons maintenant à déterminer le nombre de tests, c'est-à-dire le nombre de configurations réseau différentes, nécessaires pour évaluer l'équité entre les flux concurrents avec des algorithmes de contrôle de congestion différents. Pour deux flux testés, nous avons réalisé 374 configurations réseau correspondant à chaque valeur de latence, de taux de perte, de taille de buffer et de queuing policies (voir tableau 6.1). La durée d'exécution de chaque test prend en moyenne 10 min. Ainsi, 2 jours et 14h sont nécessaires pour comparer deux algorithmes de contrôle de congestion. Puisqu'il y a six combinaisons possibles dans ce chapitre (avec trois algorithmes de contrôle de congestion considérés), la durée d'exécution totale est de plus de 41 jours. Mais le nombre d'algorithmes

Algorithmes	Valeurs d'équités avec SFA
RENO versus RENO	0.97
BBR versus BBR	0.90
PCC versus PCC	0.88
RENO versus BBR	0.60
RENO versus PCC	0.50
BBR versus PCC	0.51

Tableau 6.4 – 90ème percentile des valeurs de fairness obtenues avec la méthode SFA à partir de 374 configurations réseau.

de contrôle de congestion est beaucoup plus important dans la littérature, l'ajout de plus d'algorithmes augmenterait de façon considérable le nombre de tests. C'est pourquoi nous cherchons à caractériser le nombre de tests nécessaires pour diminuer le temps d'exécution tout en ayant un résultat d'équité représentatif de la réalité.

Nous prenons comme référence le 90ème percentile pour représenter un résultat final unique de toutes les valeurs K obtenues. En effet, c'est une bonne façon d'exprimer le résultat dans la plupart des configurations (à l'exception du 10ème percentile des valeurs aberrantes). Nous présentons les résultats des six combinaisons dans le tableau 6.1.

Comme la durée d'exécution est trop longue, même pour deux flux concurrents, nous sommes alors intéressés à obtenir une juste valeur représentative en utilisant un nombre limité de tests. Nous choisissons donc de façon aléatoire un plus petit nombre de configurations, allant de 20 à 200, et nous calculons le 90ème percentile obtenu à partir des résultats recueillis. Afin d'obtenir une valeur K représentative, nous répétons 1000 fois l'évaluation de l'équité pour chaque groupe de la configuration du réseau choisie au hasard. Ainsi, l'exactitude de l'équité peut être facilement perçue en fonction du nombre de tests. Les résultats de précision sont présentés à la figure 6.9.

La figure 6.9 permet de tirer les conclusions suivantes : *(i)* Un petit nombre de valeurs (notamment 20) peuvent être insuffisantes pour atteindre une précision représentative de la réalité, en particulier pour les tests suivants : BBR versus BBR, PCC versus PCC et RENO versus BBR. La différence entre la mesure exhaustive et la mesure estimée (c'est-à-dire lorsqu'on utilise des configurations de réseau choisies au hasard) peut atteindre 0.2, ce qui constitue une différence importante avec la réalité. Dans le pire des cas, la mesure estimée pour les configurations du groupe 20 de tests donne un 90ème percentile égal à 0.5 bien que la valeur exacte soit 0.88. *(ii)* Les expériences montrent que plus de 100 tests (soit moins d'un tiers de l'analyse exhaustive) suffisent à fournir une mesure précise et fiable de l'équité entre deux flux concurrents. L'exécution d'un plus grand nombre de tests (par exemple 200 tests) ne permet pas d'améliorer la précision.

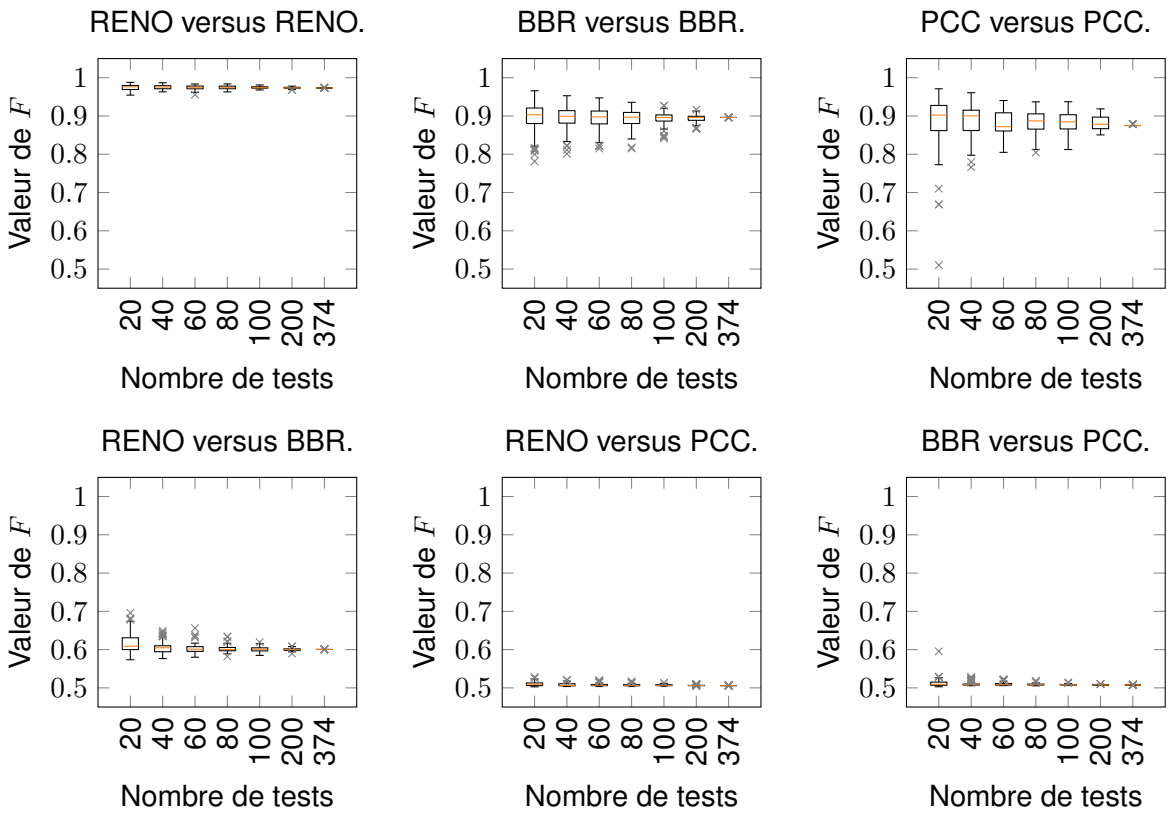


FIGURE 6.9 – L'évaluation de la précision de la méthode SFA en fonction du nombre d'essais.

6.5 Bilan sur l'évaluation de l'équité des algorithmes de contrôle de congestion

La mise en œuvre des algorithmes de contrôle de congestion sur Internet a récemment reçu une attention croissante de la part des chercheurs. Les nouveaux algorithmes de contrôle de congestion comme BBR et PCC ont été conçus pour améliorer l'efficacité de la performance du réseau en termes de débit pour la satisfaction de fournisseurs de services. De plus, la mise en œuvre de QUIC est la promesse d'une mise à jour et d'un déploiement plus rapides pour les améliorations de QUIC. Cet axe de recherche peut toutefois devenir problématique si les nouveaux algorithmes de contrôle de congestion ne sont pas suffisamment mis à l'essai pour déterminer l'équité sur le réseau. La question de la validation d'un algorithme de contrôle de congestion pour savoir s'il se comporte bien avec d'autres flux concurrents dans le réseau est ouvert.

Dans le présent chapitre, les procédures de mesures SFA et WSFA permettent d'évaluer l'équité entre plusieurs flux en compétition dans le réseau. Ces méthodes sont appliquées sur les algorithmes de contrôle de congestion : RENO, BBR et PCC. Les résultats démontrent que ces algorithmes sont équitables avec eux-mêmes, mais que lorsqu'ils sont mis en concur-

rence entre-eux des fortes inéquités peuvent apparaître. Dans l'ensemble des configurations réseaux testées, ils est toujours possible d'en déterminer où les algorithmes de contrôle de congestion sont équitables, partiellement équitables ou inéquitables. C'est pour cela que nous avons déterminé un nombre de tests requis (100) pour évaluer l'équité.

Chapitre 7

Conclusions

Contents

7.1 Conclusions, observations et propositions

Au fil du temps, la pile de protocoles d'Internet n'a cessé d'évoluer pour améliorer la performance des services réseau et des applications des utilisateurs. Par exemple, Quick UDP Internet Connections (QUIC) a été introduit par Google en 2012 pour remplacer le protocole Transmission Control Protocol (TCP) et Transport Layer Security (TLS), notamment pour les connexions web. Actuellement, l'Internet Engineering Task Force (IETF) l'étudie en vue d'une normalisation. Le protocole QUIC réutilise des fonctions de transport de l'état de l'art comme les protocoles TCP pour le transport des données, TLS pour la sécurisation des données, Stream Control Transmission Protocol (SCTP) lui est capable de multiplexer plusieurs sessions d'application (par exemple HyperText Transfer Protocol (HTTP)) sur une seule connexion de transport alors que TCP utiliserait plutôt plusieurs connexions pour éviter le Head-of-line (HOL) blocking. De plus, de nouveaux algorithmes de contrôle de congestion ont été conçus comme Bottleneck Bandwidth and Round-trip propagation time (BBR) et Performance-oriented Congestion Control (PCC) pour améliorer les performances des utilisateurs. Ils créent une rupture sur les principes de fonctionnement avec l'abandon du principe Additive Increase and Multiplicative Deacrease (AIMD) qui est actuellement utilisé par les algorithmes les plus couramment utilisés sur Internet comme RENO et CUBIC.

C'est dans ce contexte que cette thèse s'est déroulée en réalisant des mesures d'équités sur les nouvelles piles de transport. Nous commencerons par conclure sur la procédure de mesure avec la plateforme de tests et les nouvelles métriques introduites pour déterminer l'équité entre les flux. Puis nous aborderons les résultats d'équité obtenus sur le nouveau protocole de transport QUIC et nous terminerons par les conclusions sur l'évaluation de l'équité des nouveaux algorithmes de contrôle de congestion.

7.1.1 Procédure de mesures

Plateforme de mesures

Afin de pouvoir réaliser les mesures d'équité entre les différents protocoles et algorithmes de contrôle de congestion, nous nous sommes basés sur une plateforme de mesure permettant de reproduire des conditions réelles de réseau fixe ou mobile. Cette plateforme regroupe trois parties distinctes : la partie utilisateur, la partie réseau et la partie serveur. Elles permettent de reproduire des conditions réelles d'utilisation et de générer l'ensemble des flux nécessaires à l'évaluation de l'équité. Pour représenter exactement l'ensemble des configurations réseaux, nous pouvons déterminer sur la plateforme différents paramètres réseau comme la latence, le taux de perte, la taille du *buffer* et la *queuing policy*. À partir des données évaluées sur la plateforme, plusieurs statistiques sont mesurables pour étudier le comportement des flux tel que par exemple le débit des différents flux, le remplissage et le taux de perte du *buffer* correspondant au point de saturation.

Mesure de l'équité

L'axe d'étude de la thèse est d'évaluer l'équité sur le réseau. Lors de notre état de l'art, nous avons fait le constat que l'équité est mesurée uniquement à un instant précis, or un utilisateur évalue la qualité d'un service sur l'ensemble de la session. C'est pourquoi nous avons décidé de proposer une métrique, nommée Session Fairness Assessment (SFA), qui permet de mesurer l'équité d'une session complète. La mesure proposée donne une valeur quantitative de l'équité qui varie de $|0.5|$ (injuste) à 0 (juste). Le signe de cette métrique nous indique quel est le flux dominant.

Une deuxième version de la métrique, reprenant les mêmes principes que la méthode SFA, a été développée. Cette dernière, nommée Weighted Session Fairness Assessment (WSFA), tient en compte le fait que tous les moments d'une session n'ont pas la même importance. Dans notre exemple, nous considérons que le début d'une session est un moment critique où des politiques trop agressives peuvent nuire au partage équitable des ressources du réseau. C'est pourquoi la version de WSFA proposée accorde plus d'importance au début de la ses-

sion. Mais d'autres types de pondération sur la métrique WSFA pourraient être réalisés (par exemple, donner plus d'importance aux inéquités importantes de la session ce qui entraînerait une dégradation majeur de la Quality of Experience (QoE) comme une interruption de service).

7.1.2 Mesure d'équité du protocole QUIC

Bilan des résultats

QUIC est actuellement en cours de déploiement sur Internet et est destiné à remplacer TCP et TLS. Il adopte plusieurs algorithmes déjà implémentées dans TCP telle que hybrid start (hystart) et CUBIC. Toutefois, sa mise en œuvre ne doit pas se faire au détriment de l'équité pour l'utilisateur, ceci demeure une préoccupation croissante pour les opérateurs, car leurs services à la clientèle sont en première ligne lorsqu'une application internet perd en performance.

Dans le travail dont il est question dans le présent document, nous avons analysé comment la mise en œuvre de QUIC peut avoir une incidence sur l'équité lorsqu'il est en concurrence avec un flux TCP. Pour réaliser cette mesure de l'équité, les deux protocoles utilisent le même algorithme de contrôle de congestion (CUBIC) afin d'isoler l'incidence des implémentations des protocoles sur l'équité. Nous avons utilisé la plateforme de mesures pour analyser le comportement de chacun des flux TCP et QUIC. Plusieurs aspects sont analysés au cours du processus de mesures et nous soulignons plusieurs problèmes d'équité. Tout d'abord, l'impact de l'option hystart a été mesuré sur les différents flux dans des situations variées sur le réseau. Ensuite, nous avons considéré une mesure de l'équité du nombre de connexions TCP émulées dans QUIC (N). Enfin, nous avons étudié l'impact sur l'équité de la limitation de la taille de la fenêtre de congestion dans QUIC pour de grandes valeurs de N.

En ce qui concerne l'option hystart présente dans les implémentations TCP et QUIC, les résultats montrent clairement un problème important de partage de capacité réseau. Ce problème survient principalement lorsque les taux de pertes sur le réseau sont faibles. En effet, dans certaines configurations de réseau, cette option empêche le *slow-start* de fonctionner comme il se doit, c'est-à-dire qu'elle facilite une montée en débit rapide du flux soit au moment de son lancement, soit après un évènement de *timeout*. Le comportement de l'option hystart peut donc réduire considérablement le débit de l'utilisateur pendant toute la durée de la session et créer ainsi un réel problème en termes de QoE. Nous recommandons donc de désactiver cette option ou de l'améliorer de manière significative (en n'interprétant pas systématiquement l'augmentation du Round-Trip Time (RTT) comme une congestion naissante) sur QUIC et TCP, afin d'assurer une bonne qualité de service et donc l'équité du réseau.

Nous avons aussi déterminé que la valeur de N a un réel impact sur le comportement de l'algorithme de contrôle de congestion de QUIC et sur l'équité entre les protocoles. Quand $N = 1$ les protocoles TCP et QUIC partagent équitablement la bande passante disponible dans toutes les configurations réseau testées. Mais quand la valeur de N augmente alors l'équité diminue considérablement en faveur de QUIC. En d'autres termes, le protocole QUIC obtient

un débit beaucoup plus élevé que le protocole TCP. La mise en place de N est actuellement statique dans les implémentations de QUIC et est par défaut égale à 2. Cependant, la valeur de N peut être facilement modifiée et incrémentée dans les applications clients et serveurs. Il est donc indispensable de pouvoir définir des règles d'ingénierie pour dynamiquement adapter cette valeur dans le temps en fonction des demandes des utilisateurs (par exemple, le nombre de *stream*s dans QUIC) et de l'état du réseau pour garantir un comportement équitable des protocoles sur le réseau. Nous sommes d'avis que des lignes directrices de mise en œuvre et la surveillance des usages seraient bénéfiques pour la majorité des utilisateurs.

Enfin, le dernier paramètre étudié est la limite de la taille de la fenêtre de congestion par l'algorithme de contrôle de congestion CUBIC dans QUIC. Pour QUIC, cette valeur est constante pendant la session et ne dépend pas de la valeur de N. Nous montrons que lorsque la valeur de N augmente, la limitation de la taille de la fenêtre de congestion empêche le protocole QUIC d'accroître son débit. Cette limitation ne permet donc pas l'émulation de plusieurs connexions TCP lorsque le taux de perte sur le réseau est faible. Cette limitation entrave le principe du multiplexage de plusieurs connexions d'applications dans le protocole QUIC.

La loi du plus fort pour obtenir le plus de bande passante du réseau

Comme nous l'avons vu, de nombreuse options des implémentations de QUIC peuvent avoir un impact sur l'équité. Or les développeurs des implémentations open source QUIC [23, 24] ont leurs propres vues sur le partage équitable des capacités réseau avec un flux TCP. En ce qui concerne le déploiement, il est difficile d'inverser la politique d'équité suivie par les terminaux QUIC et, à notre connaissance, ces politiques ne sont pas documentées par les développeurs, notamment Google avec sa pile QUIC côté serveur (le code et les réglages spécifiques du serveur QUIC de Google sont actuellement tenus secrets [9]). De plus, aujourd'hui, ces questions sont largement ignorées lors de la standardisation de QUIC [8].

QUIC est conçu pour être plus facile à faire évoluer et à redéployer que son prédécesseur TCP. Cela favorise l'émergence rapide des variantes de QUIC. De plus, le déploiement d'une variante QUIC est accessible aux éditeurs d'applications (par opposition aux éditeurs de systèmes d'exploitation pour TCP). C'est un terrain favorable pour voir davantage de variantes de QUIC déployées de façon significative contrairement à celles qui l'ont été dans l'histoire de TCP.

S'agit-il d'ouvrir une nouvelle ère de l'Internet, où les politiques de partage équitable des capacités des réseaux ne résultent plus d'une convention collective et où une multitude de choix individuels différents seront faits ?

De notre avis, s'il est concevable qu'une certaine équité (telle que mesurée par la métrique SFA) puisse être échangée pour une meilleure utilisation des capacités du réseau ou pour une meilleure expérience utilisateur (ces deux aspects étant hors du cadre de cette thèse), ces compromis doivent cependant être le résultat de discussions ouvertes et de conventions collectives dans la communauté internet (notamment par la normalisation) plutôt que le résultat

du droit des plus forts. Nous préconisons ici que la surveillance et la documentation des nouvelles pratiques en matière de partage de la capacité des réseaux entre flux simultanés doivent alimenter ces discussions et générer des directives de mise en œuvre par les organismes de normalisation.

7.1.3 Équité des algorithmes de contrôle de congestion

Mesure de l'équité des algorithmes de contrôle de congestion

Durant ces dernière années, l'attention des chercheurs pour améliorer les performances du réseau s'est portée sur les algorithmes de contrôle de congestion. Dans ce but, de nouveaux algorithmes de contrôle de congestion ont été conçus comme PCC et BBR pour accroître le débit des flux. De plus, la mise en place du protocole de transport dans les applications clientes et serveurs, avec le protocole QUIC, permettent de mettre à jour plus rapidement les algorithmes de contrôle de congestion et donc de multiplier les versions sur Internet. La multiplication des versions d'algorithmes de contrôle de congestion peut toutefois devenir problématique si les nouveaux algorithmes ne sont pas suffisamment testés au niveau de l'équité.

Dans le présent document, nous présentons une approche générique pour mesurer l'équité entre les algorithmes de contrôle de congestion. Pour cela, nous nous basons sur une plateforme de mesure en laboratoire et les procédures de mesure de l'équité nommée SFA et WSFA. Nous appliquons cette procédure sur trois algorithmes de contrôle de congestion, à savoir RENO, BBR et PCC, sur plus de 300 configurations réseau différentes. A partir des résultats obtenus nous pouvons en conclure que les algorithmes de contrôle de congestion sont équitables avec eux-mêmes. Mais lorsque les algorithmes de contrôle de congestion sont mis en concurrence entre-eux, alors l'équité dépend des situation réseau (taux de perte, latence et *queuing policies*). Il est alors parfaitement possible de déterminer des situations réseau où les algorithmes seront équitables, partiellement équitables ou inéquitables. C'est pour ces raisons que nous avons cherché à quantifier le nombre de situations réseau nécessaires pour obtenir une valeur représentative de la réalité. Nous avons déterminé qu'à partir de 100 situations réseau différentes nous l'obtenons.

La performance et l'équité

Lors de la proposition de nouveaux algorithmes de contrôle de congestion, ce sont essentiellement des tests de performances qui sont mis en avant. En effet, pour BBR et PCC, les auteurs montrent des chiffres-chocs sur l'amélioration du débit des utilisateurs. Effectivement, l'étude de performance de l'algorithme de BBR montre que cette solution peut atteindre des débits jusqu'à 25 fois supérieurs à l'algorithme le plus utilisé sur le réseau internet (CUBIC). À partir de cette constatation, nous pouvons alors penser que les différents acteurs (fournisseurs de contenus, opérateurs et utilisateurs) ne peuvent être que satisfaits de ces évolutions. De fait,

leurs débits pourront alors être considérablement améliorés sans apporter de modifications à l'infrastructure du réseau. Cette affirmation est particulièrement vraie lorsque les flux n'arrivent pas à atteindre le débit maximal du réseau, comme ça a été le cas avec l'apparition de l'algorithme CUBIC qui a remplacé progressivement RENO sur le réseau. Réellement, RENO possède de très fortes lacunes sur des réseaux à débit important, à cause d'une croissance très lente du débit.

Maintenant, si nous considérons que l'ensemble des flux utilise l'intégralité de la bande passante du réseau, des questions de partage de capacité et donc d'équité se posent. Cette situation de partage de ressources réseau, par exemple lors d'un téléchargement de contenus, peut alors se produire avec différents algorithmes de contrôle de congestion (par exemple, avec des algorithmes largement déployés (comme CUBIC) ou de nouveaux algorithmes en cours de déploiement comme BBR). Même si BBR peut prétendre à terme remplacer CUBIC, comme ça a été le cas entre RENO et CUBIC, une période de transition pouvant durer plusieurs années est nécessaire. Donc durant ce laps de temps, la probabilité que ces deux algorithmes de contrôle de congestion soient en concurrence est accrue. Or nous avons démontré que dans certaines configurations réseau les deux algorithmes de contrôle de congestion ont des problèmes majeurs d'équité. Donc la cohabitation de ces protocoles sur le réseau peut entraîner des difficultés pour certains fournisseurs de services. De plus, les nouveaux algorithmes de contrôle de congestion doivent pouvoir garantir la bonne interopérabilité avec l'ensemble des algorithmes déjà présents et ne doit pas viser à les remplacer (non étudié pendant cette thèse). Prenons l'exemple de TCP Friendly Rate Control (TFRC) qui est spécialisé dans les communications en temps-réel : il faut vérifier que tous nouveaux algorithmes déployés sur le réseau ne nuisent pas à son bon fonctionnement.

Dans ces deux exemples, une étude approfondie de l'équité doit alors certifier que celle-ci est respectée sur le réseau et qu'un flux n'obtient pas abusivement plus de bande passante qu'il ne le devrait. Or généralement, lors de la présentation de nouveaux algorithmes de contrôle de congestion, ce problème n'est que rarement abordé ou insuffisamment testé.

À partir de ces constats, une question se pose alors : faut-il alors prioriser la performance des algorithmes de contrôle de congestion au détriment de l'équité du réseau ?

De notre point de vue, les recherches qui sont actuellement en cours sur les algorithmes de contrôle de congestion sont primordiales pour pouvoir améliorer les performances des utilisateurs, par exemple en utilisant l'intégralité de la bande passante du réseau, tout en minimisant le remplissage des *buffer*s qui introduit de la latence supplémentaire pour les utilisateurs. Mais toute nouvelle proposition devrait être testée dans différentes situations réseau pour connaître l'ensemble des impacts sur les autres flux afin de s'assurer que l'équité réseau n'est pas dégradée de façon trop importante. L'équité ne doit pas être un frein à l'évolution des algorithmes de contrôle de congestion, mais doit permettre la vérification qu'aucun comportement abusif n'existe sur le réseau.

7.2 Contributions majeures

L'évaluation de la mesure de l'équité des algorithmes de contrôle de congestion et des protocoles de transport a constitué le cœur des activités de recherches. Les principales contributions de la thèse sont les suivantes :

- *Mesure de l'équité sur une session.* Dans cette thèse, nous proposons une nouvelle approche pour évaluer l'équité réseau. Pour cela, nous ne nous basons plus sur un moment précis de la session, mais sur l'intégralité de celle-ci pour obtenir une vision représentative de l'équité telle que la perçoit un utilisateur.

- *Mesure pondéré de l'équité sur une session.* Nous avons repris le même principe que la mesure de l'équité d'une session, mais nous avons voulu réaliser une métrique plus proche de la QoE des utilisateurs. Pour cela, nous avons donc décidé de pondérer de façon plus important certain moments ou évènements de la session. Dans notre exemple, nous avons décidé de prendre d'avantage en considération le début de la session.

- *Plateforme de mesures disponible en open-source.* La plateforme de tests qui a permis de réaliser l'ensemble des mesures de cette thèse est disponible en open-source et est réutilisable pour réaliser des mesures de débit de flux ou le remplissage du *buffer* dans différentes situations réseau (réseau fixe ou mobile, latence, taux de perte, taille et *queuing policies* du *buffer*).

- *Logiciel de traitement des données.* Nous proposons aussi un programme open-source pour réaliser le traitement des données et appliquer les mesures d'équité sur une session.

- *Analyse de l'implémentation de l'algorithme de contrôle de congestion CUBIC.* Une analyse précise des différences d'implémentation de l'algorithme de contrôle de congestion CUBIC présent dans QUIC et dans TCP a été effectuée.

- *Une analyse de l'équité du nouveau protocole de transport QUIC.* Une analyse de l'équité réseau entre TCP et QUIC a été effectuée. Elle permet de mettre à jour les différents paramètres et options qui ont un impact négatif sur l'équité entre les flux. Nous avons déterminé que l'option hystart, le nombre de connexions TCP émulées dans QUIC et la limitation de la taille de la fenêtre de congestion dans QUIC ont un impact important sur l'équité.

- *Une analyse de l'équité des algorithmes de contrôle de congestion.* Une analyse sur l'équité des nouveaux algorithmes de contrôle de congestion (BBR et PCC) a été réalisée. Nous avons démontré que ces algorithmes de contrôle de congestion sont équitables entre-eux. Mais lorsqu'ils sont mis en concurrence deux à deux, il est toujours

possible de pouvoir déterminer des configuration réseau où il sont inéquitables, partiellement équitables, ou équitables. Ce qui implique que les algorithmes de contrôle de congestion doivent être testés dans plusieurs situations réseau pour obtenir un résultat représentatif de la réalité. Avec la méthode de mesure SFA, nous avons démontré qu'il fallait réaliser une centaine de tests pour mesurer l'équité entre plusieurs algorithmes de contrôle de congestion.

7.3 Perspectives de recherches

L'analyse de l'équité sur l'évolution des protocoles de transport, réalisée durant cette thèse, ouvre de nouvelles perspectives de recherche, notamment :

- *Points de peering :* Durant cette thèse, nous avons étudié l'équité uniquement sur les réseaux d'accès d'un opérateur. Une perspective de recherches serait d'étendre les mesures d'équité sur les points de peering entre les différents acteurs de l'Internet.

- *Métrique d'équité sur la session et la QoE :* au cours du chapitre 4, nous avons défini deux procédures de mesure de l'équité pour évaluer celle-ci sur la session d'un flux. Cette métrique est aujourd'hui opérationnelle pour mesurer l'équité entre plusieurs flux, mais le lien entre cette métrique et la QoE reste à réaliser.

- *Simulation de plusieurs connexions TCP dans QUIC :* durant le chapitre 5, nous avons évalué que l'équité était fortement impactée par un paramètre, nommé N, qui permet la simulation de plusieurs connexions TCP dans le protocole de transport QUIC. Or, ce paramètre est statique durant toute la session et n'est pas optimisé tel qu'il est implémenté aujourd'hui dans les piles QUIC. Des études plus approfondies pourront être menées pour l'améliorer et rétablir de l'équité entre le protocole QUIC et TCP.

- *Amélioration de l'option hystart :* dans le chapitre 5, nous avons aussi déterminé que l'option hystart posait des problèmes d'équité et de performance sur le réseau. En effet, dans certaine configurations réseau (augmentation du RTT), l'option hystart présente dans le protocole TCP et QUIC détecte des congestions à tort. Une autre piste de travail consisterait à améliorer cette option pour qu'elle soit optimale dans l'ensemble des configurations réseau possibles.

- *Réalisation de mesures sur Internet :* afin de confirmer l'intégralité des tests effectués sur la plateforme de simulation, des tests sur réseau réel pourraient être réalisés, afin de confirmer l'ensemble des résultats obtenus sur la plateforme de mesures.

– *Une publication de Draft IETF ou IRTF.* L'ensemble de ces résultats abordent un sujet nouveau et absent à l'IETF ou Internet Research Task Force (IRTF) : l'analyse de l'équité sur les implémentations des protocoles de transport. Il est néanmoins primordiale de pouvoir remonter de tels résultats en normalisation afin de permettre une prise de conscience des différents acteurs au sujet des impacts qu'ont les paramètres de QUIC sur le débit et donc sur l'équité. Le but final est d'aboutir à la normalisation d'algorithmes (par exemple sur le nombre de connexions TCP émulées dans QUIC) qui garantiraient l'équité de QUIC telle qu'elle existait avec TCP.

– *Un observatoire de l'équité.* Dans les versions des protocoles de transport tels que TCP et User Datagram Protocol (UDP), la question de l'équité entre les différentes implémentations de ces protocoles est moins d'actualité qu'avec le protocole QUIC. En effet, elles présentent très peu de version existent sur les réseaux du fait que ces protocoles sont directement implémenté dans les Operating System (OS) des machines réseau. Au contraire, le protocole QUIC permet une plus grande diversification des implémentations. Aujourd'hui nous voyons des acteurs comme Facebook, Google, Akamai déployer leurs propres versions du protocole QUIC et aussi potentiellement leurs propres algorithmes de contrôle de congestion. Nous sommes persuadés que de nombreux autres acteurs vont très rapidement réaliser le même type d'implémentations avec des solutions propriétaires. Afin de pouvoir mesurer l'équité entre les différents acteurs, nous préconisons la création d'un observatoire de l'équité sur le réseau. Cet observatoire est complètement envisageable, du fait, que seuls les débits des différents flux sont nécessaires à la mesure de l'équité sur le réseau Internet.

7.4 Publications

Conférences internationales :

– R. Corbel, A. Braud, X. Marjou, G. Simon, A. Gravey. "Fairness Measurement Procedure for the Evaluation of Congestion Control Algorithms", 11th International Conference on Computer Science and Information Technology (ICCSIT 2018), 2018, Paris, France.

– R. Corbel, S. Tuffin, A. Gravey, X. Marjou, and A. Braud, "Assessing the impact of QUIC on network fairness," in 2nd International Conference on Communication and Network Protocol (ICCNP 2019), 2019, Nice, France.

– R. Corbel, S. Tuffin, A. Gravey, A. Braud and X. Marjou, "Impact of QUIC on fairness in mobile networks" in 10th International conference on the Network of the Future (NOF'19), 2019, Rome, Italie.

Brevet :

– R. Corbel, E. Stephan, “Procédé de préservation d’un débit d’émission de données d’un terminal dans un réseau de communications”, 2018, Ref : 201653.

Contribution open-source :

– Plateforme de simulation de réseau fixe et mobile.

– Logiciel de traitement des données pour la fairness.

Prix et distinctions :

– Prix de la meilleur présentation de la conférence, 11th International Conference on Computer Science and Information Technology (ICCSIT 2018), 2018, Paris, France.

Annexes

Annexe A

Évaluation des paramètres réseau avec des algorithmes de contrôle de congestion en compétition avec eux-mêmes en utilisant la méthode WSFA

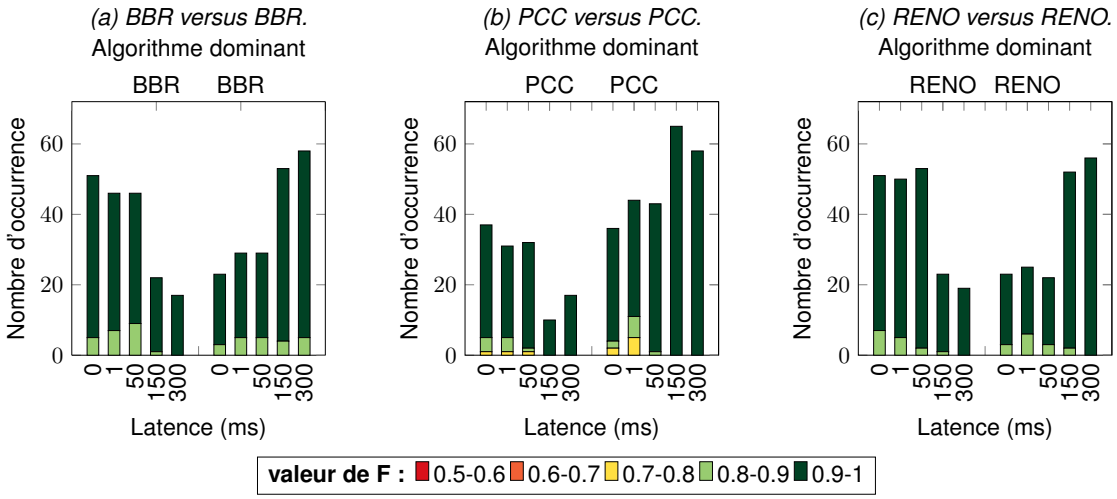


FIGURE A.1 – Comparaison des algorithmes de contrôle de congestion avec eux même en les classant par latence. La fairness est évalué avec la méthode WSFA.

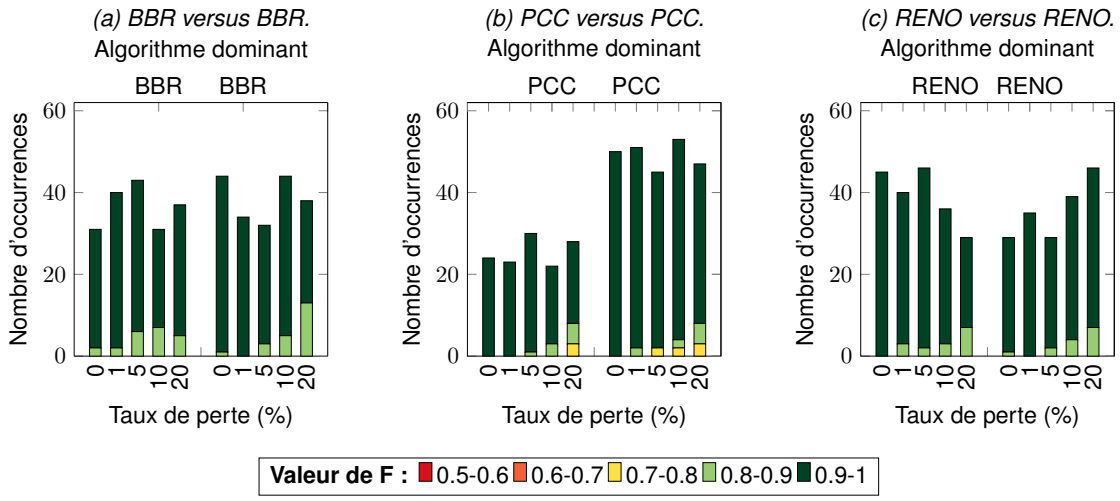


FIGURE A.2 – Comparaison des algorithmes de contrôle de congestion avec eux même en les classant par taux de perte. La fairness est évalué avec la méthode WSFA.

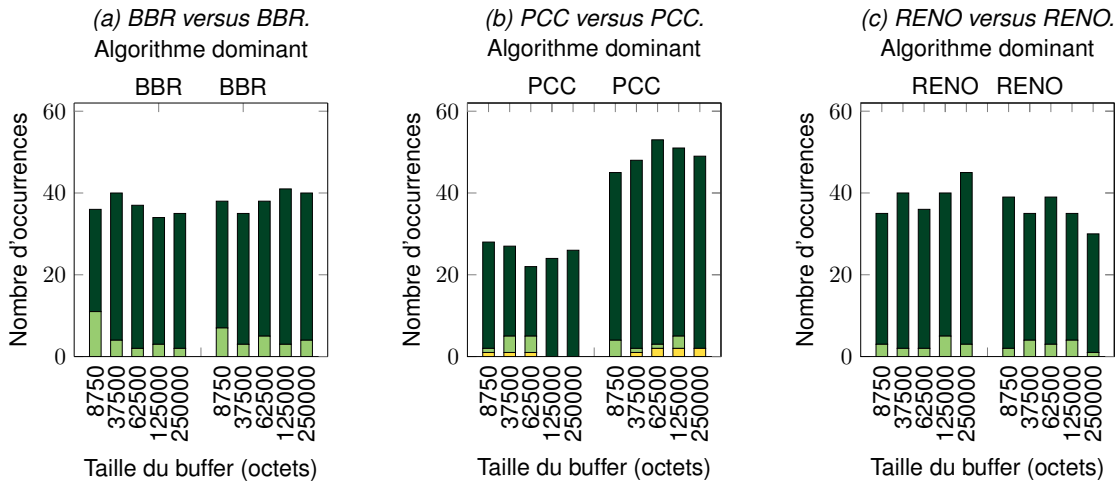


FIGURE A.3 – Comparaison des algorithmes de contrôle de congestion avec eux même en les classant par taille de buffer. La fairness est évalué avec la méthode WSFA.

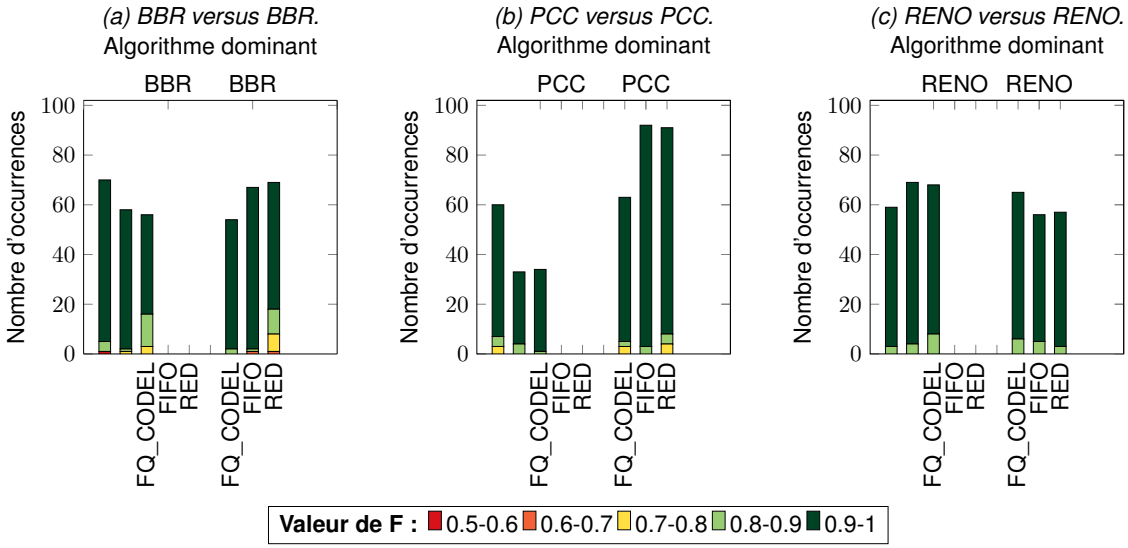


FIGURE A.4 – Comparaison des algorithmes de contrôle de congestion avec eux même en les classant par queueing policies. La fairness est évalué avec la méthode WSFA.

Annexe B

Évaluation des paramètres réseau avec des algorithmes de contrôle de congestion en compétition entre-eux en utilisant la méthode WSFA

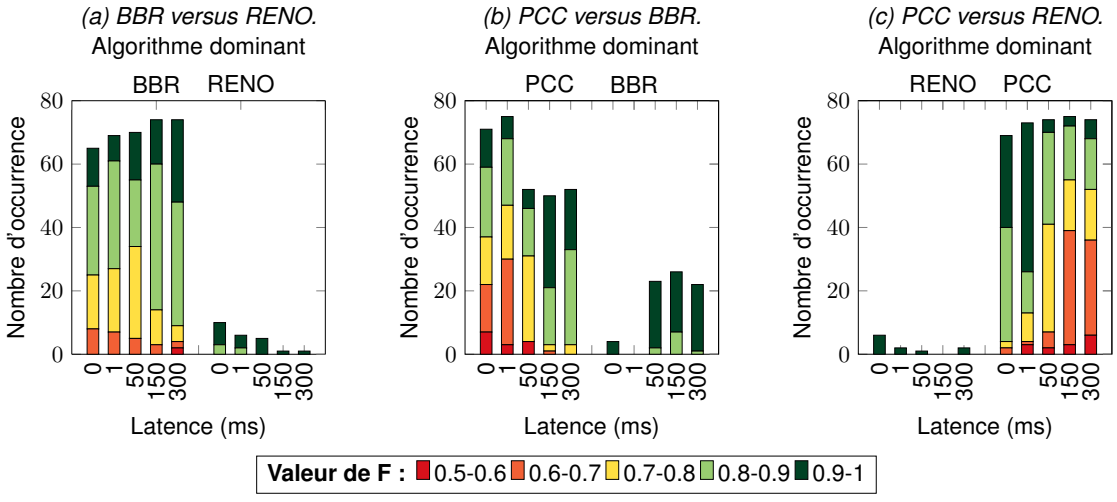


FIGURE B.1 – Comparaison des algorithmes de contrôle de congestion entre eux en les classant par latence. La fairness est évalué avec la méthode WSFA.

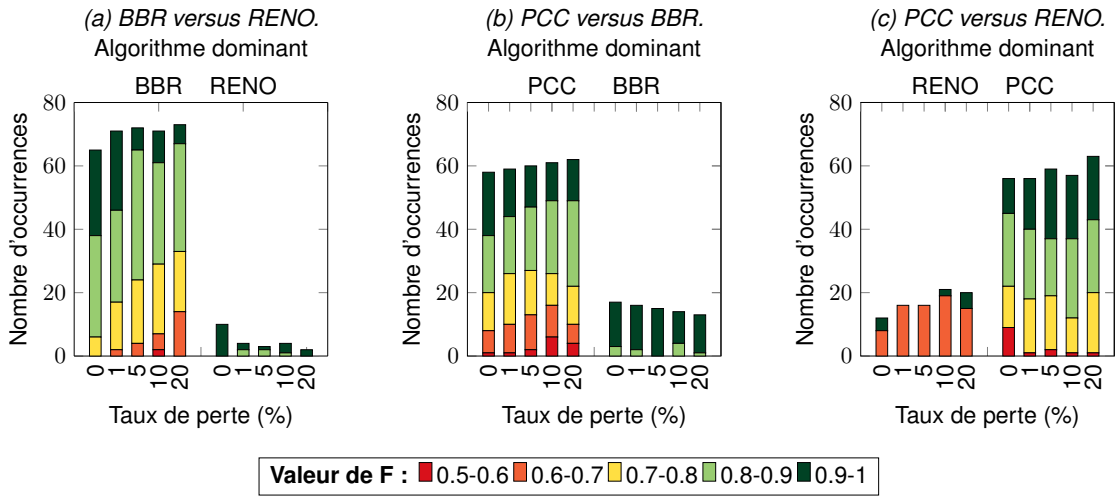


FIGURE B.2 – Comparaison des algorithmes de contrôle de congestion entre eux en les classant par taux de perte. La fairness est évalué avec la méthode WSFA.

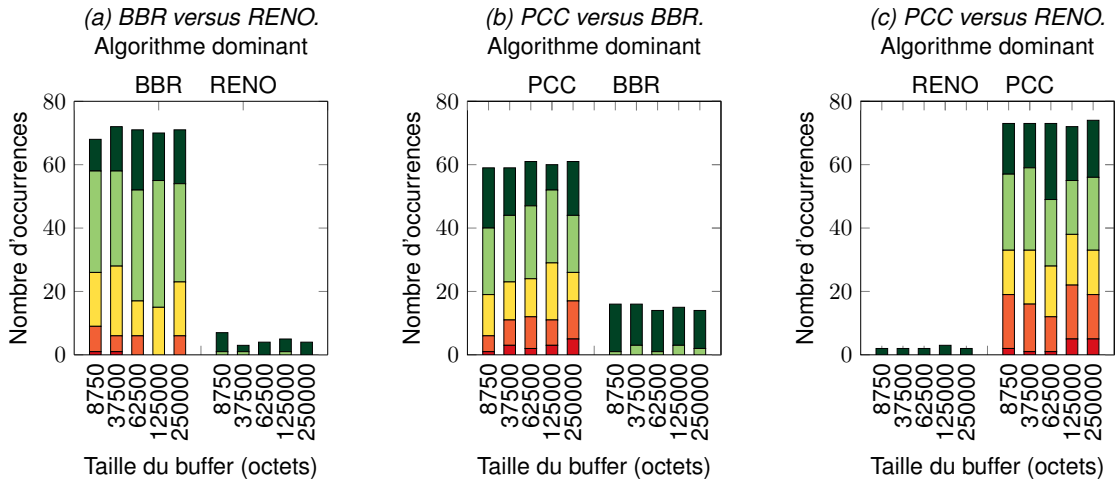


FIGURE B.3 – Comparaison des algorithmes de contrôle de congestion entre eux en les classant par taille de buffer. La fairness est évalué avec la méthode WSFA.

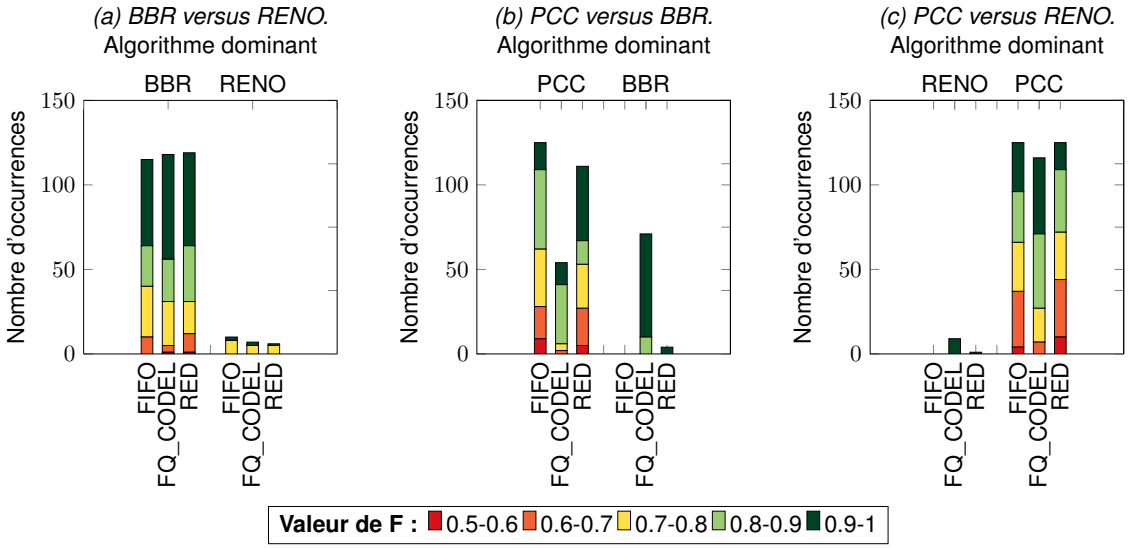


FIGURE B.4 – Comparaison des algorithmes de contrôle de congestion entre eux en les classant par la queuing policies. La fairness est évalué avec la méthode WSFA.

Bibliographie

[1] Benoit Miscopein, Jean-Baptiste Doré, Emilio Calvanese Strinati, Dimitri Kténas, and Sergio Barbarossa. Air Interface Challenges and Solutions for future 6G Networks. working paper or preprint, January 2019.

[2] Patrick Faure, David Perreau, Laurent Gou, Joël Lapraye, Patrice Laigo, and Marc. Prunier. De la mesure à la QoE, 2015.

[3] Yanjiao Chen, Kaishun Wu, and Qian Zhang. From QoS to QoE : A Tutorial on Video Quality Assessment. *IEEE Communications Surveys Tutorials*, 17(2) :1126–1165, Secondquarter 2015.

[4] Mamadou Alpha Barry, James K. Tamgno, Claude Lishou, and Modou Bamba Cissé. QoS impact on multimedia traffic load (IPTV, RoIP, VoIP) in best effort mode. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 694–700, Feb 2018.

[5] ITU-R. IMT Vision Framework and overall objectives of the future development of IMT for 2020 and beyond, 2015.

[6] Akhil Gupta and Rakesh Kumar Jha. A Survey of 5G Network : Architecture and Emerging Technologies. *IEEE Access*, 3 :1206–1232, 2015.

[7] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The QUIC Transport Protocol : Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 183–196. ACM, 2017.

[8] Jana Iyengar and Martin Thomson. QUIC : A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-20, Internet Engineering Task Force, April 2019. Work in Progress.

[9] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at QUIC : an approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference*, pages 290–303. ACM, 2017.

[10] Andrei Gurtov, Tom Henderson, Sally Floyd, and Yoshifumi Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582, April 2012.

[11] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. CUBIC for Fast Long-Distance Networks. RFC 8312, February 2018.

[12] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR : Congestion-Based Congestion Control. *Queue*, 14(5) :50 :20–50 :53, October 2016.

[13] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, and Van Jacobson. BBR Congestion Control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, Internet Engineering Task Force, July 2017. Work in Progress.

[14] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. {PCC} : Re-architecting Congestion Control for Consistent High Performance. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 395–408, 2015.

[15] Transmission Control Protocol. RFC 793, September 1981.

[16] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[17] Henrik Frystyk Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

[18] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, May 2015.

[19] User Datagram Protocol. RFC 768, August 1980.

[20] Florian Gratzer. QUIC - Quick UDP Internet Connections. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, 39, 2016.

[21] w3techs. Usage of QUIC for websites. `https://w3techs.com/technologies/details/ce-quic/all/all`.

[22] Randall R. Stewart. Stream Control Transmission Protocol. RFC 4960, September 2007.

[23] QUIC-GO. `https://github.com/lucas-clemente/quic-go`.

[24] Chromium. `https://github.com/chromium/chromium`.

[25] proto-QUIC. `https://github.com/google/proto-quic`.

[26] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *ACM SIGCOMM Computer Communication Review*, 26(3) :5–21, 1996.

[27] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. TCP Congestion Control. RFC 5681, September 2009.

[28] 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.

[29] Varun Singh, Joerg Ott, and Stefan Holmer. Evaluating Congestion Control for Interactive Real-time Media. Internet-Draft draft-ietf-rmcat-eval-criteria-08, Internet Engineering Task Force, November 2018. Work in Progress.

[30] Sangtae Ha and Injong Rhee. Taming the elephants : New TCP slow start. *Computer Networks*, 55(9) :2092–2110, 2011.

[31] Sangtae Ha and Injong Rhee. Hybrid slow start for high-bandwidth and long-distance networks. In *Proc. PFLDnet*, pages 1–6, 2008.

[32] fastCC. `https://github.com/private-octopus/picoquic/blob/master/picoquic/fastcc.c`.

[33] Venkat Arun and Hari Balakrishnan. Copa : Practical delay-based congestion control for the internet. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 329–342, 2018.

[34] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace : Online-learning congestion control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 343–356, 2018.

[35] Daniel B. Grossman. New Terminology and Clarifications for Diffserv. RFC 3260, April 2002.

[36] Fred Baker, Jozef Babiarz, and Kwok Ho Chan. Configuration Guidelines for DiffServ Service Classes. RFC 4594, August 2006.

[37] Ivan Marsic. Computer networks : Performance and quality of service, 2010.

[38] Jean-Yves Boudec. Rate adaptation, congestion control and fairness : A tutorial, 2000.

[39] Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks : shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3) :237–252, 1998.

[40] Patrick Poullie, Thomas Bocek, and Burkhard Stiller. A survey of the state-of-the-art in fair multi-resource allocations for data centers. *IEEE Transactions on Network and Service Management*, 15(1) :169–183, 2018.

[41] Sally Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166, March 2008.

[42] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, Digital Equipment Corporation, September 1984.

[43] Yueping Zhang, Seong-Ryong Kang, and Dmitri Loguinov. Delayed stability and performance of distributed congestion control. *SIGCOMM Comput. Commun. Rev.*, 34(4) :307–318, August 2004.

[44] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon : the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, Boston, MA, 2018. USENIX Association.

[45] Amit Srivastava. Performance evaluation of quic protocol under network congestion. 2017.

[46] Eneko Atxutegi, Åke Arvidsson, Fidel Liberal, Karl-Johan Grinnemo, and Anna Brunstrom. *TCP Performance over Current Cellular Access : A Comprehensive Analysis*, pages 371–400. Springer International Publishing, 2018.

[47] Stefan Bouckaert, Wim Vandenberghe, Bart Jooris, Ingrid Moerman, and Piet Demeester. The w-iLab. t testbed. In *International Conference on Testbeds and Research Infrastructures*, pages 145–154. Springer, 2010.

[48] Fengyuan Ren and Chuang Lin. Modeling and Improving TCP Performance over Cellular Link with Variable Bandwidth. *IEEE Transactions on Mobile Computing*, 10(8) :1057–1070, Aug 2011.

[49] Feng Li, Jae Won Chung, Xiaoxiao Jiang, and Mark Claypool. *TCP CUBIC versus BBR on the Highway*, pages 269–280. 03 2018.

[50] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi : Accurate record-and-replay for {HTTP}. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 417–429, 2015.

[51] Mahimahi. `https://github.com/ravinet/mahimahi`.

[52] Binh Nguyen, Arijit Banerjee, Vijay Gopalakrishnan, Sneha Kasera, Seungjoon Lee, Aman Shaikh, and Jacobus Van der Merwe. Towards understanding tcp performance on lte/epc mobile networks. In *Proceedings of the 4th workshop on All things cellular : operations, applications, & challenges*, pages 41–46. ACM, 2014.

[53] Pande Deb, Modiano and et al. NSF Workshop on Ultra-Low Latency Wireless Networks, March 2015.

[54] Haiqing Jiang, Zeyu Liu, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Understanding bufferbloat in cellular networks. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks : operations, challenges, and future design*, pages 1–6. ACM, 2012.

[55] Stefan Alfredsson, Giacomo Del Giudice, Johan Garcia, Anna Brunstrom, Luca De Cicco, and Saverio Mascolo. Impact of TCP congestion control on bufferbloat in cellular networks. In *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 1–7. IEEE, 2013.

[56] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab : An Overlay Testbed for Broad-coverage Services. *SIGCOMM Comput. Commun. Rev.*, 33(3) :3–12, July 2003.

[57] Brian White, Jay Lepreau, and Shashi Guruprasad. Lowering the Barrier to Wireless and Mobile Experimentation. *SIGCOMM Comput. Commun. Rev.*, 33(1) :47–52, January 2003.

[58] PCC. `https://github.com/modong/pcc`.

[59] Saturator. `https://github.com/keithw/multisend/blob/master/sender/saturatr.cc`.

[60] Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. Nf-tcp : A network friendly tcp variant for background delay-insensitive applications. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, pages 342–355, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[61] CISCO. The zettabyte era : Trends and analysis. `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html`, 2017.

[62] Martin Varela, Lea Skorin-Kapov, Toni Mäki, and Tobias Hoßfeld. QoE in the Web : A dance of design and performance. In *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 1–7, May 2015.

[63] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation : Evidence from a large video streaming service. 08 2014.

[64] Jana Iyengar and Martin Thomson. QUIC : A UDP-Based Multiplexed and Secure Transport. Internet-draft, IETF, 2018.

[65] Willem de Bruijn and Eric Dumazet. Optimizing UDP for content delivery : GSO, pacing and zerocopy.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Title: Evaluation of the fairness of a transport session

Keywords: QUIC, TCP, protocol, congestion control algorithm, fairness, hystart

Abstract: This thesis is in the context of measuring congestion on the network and the evolution of transport protocols. Changes are continually being made to meet the needs of users and new services. Congestion is one of the most critical issues because it has an impact on the performance of Internet networks, hence the need for congestion control algorithms to prevent or remove it. Today, no algorithm perfectly meets the expected requirements, and a lot of research is underway. Nevertheless, these new algorithms can affect network fairness since the behaviour of the transport protocol can change radically depending on the congestion control algorithm used in the endpoints. In addition, in recent years, transport protocols have undergone major changes. A recent significant example is Quick UDP Internet Connections (QUIC), a protocol introduced by Google, which aims to replace two widely used transport and security protocols, Transmission Control Protocol (TCP) and Transport Layer Security (TLS). QUIC is implemented in user applications (rather than in the operating system kernel). It is designed to be resistant to ossification and therefore more versatile. This makes content providers, such as Google, hegemonic about the data rate of their users.

Due to the progressive development of congestion control algorithms and the evolving nature of transport protocols, new challenges arise in fairness management. This is why, in this thesis, we focused on the development of a test platform to measure network fairness based on the flow rate of the different flows. In addition, in order to characterize fairness as perceived by a user, we focused on determining an impartial procedure for assessing fainress during an entire session of a transport flow (called Session Fairness Assessment (SFA) and Weighted Session Fairness Assessment (WSFA)). Based on these elements, we specifically analyzed the fairness of the protocols when TCP and QUIC flows coexist on a fixed and mobile network. In our fairness assessments, we identified the impact of aspects of QUIC implementation such as: emulating multiple TCP connections, limiting the size of congestion windows and using the hystart option. The results show that these mechanisms have a strong influence on fairness on both fixed and mobile networks. Indeed, a wrong setting of the default parameters of these mechanisms or the activation of the hystart option can affect the performance of transport protocols and therefore fainress. With regard to the evaluation of congestion control algorithms, the results show that the fainress between two different algorithms depends on the network configuration. This conclusion demonstrates that a measurement procedure, such as the one presented in this thesis, is relevant to conducting the fairness assessment.

In this thesis we can conclude that the lack of standardization, for example of emulating multiple TCP connections in QUIC, leads us to question more broadly how QUIC's design philosophy takes fairness into account. In addition, the results obtained on the evaluation of the fainress of congestion control algorithms allow us to question the fainress evaluation of several contributions when it is not tested in enough network configurations.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Évaluation de l'équité d'une session de transport

Mot clés : QUIC, TCP, protocole, algorithmes de contrôle de congestion, Équité, hystart

Résumé : Cette thèse s'inscrit dans le cadre de la mesure de la congestion sur le réseau et sur l'évolution des protocoles de transport. Des changements sont apportés continuellement afin de répondre aux besoins des utilisateurs et des nouveaux services. La congestion est l'un des problèmes les plus critiques car elle a un impact sur la performance des réseaux Internet, d'où la nécessité pour les algorithmes de contrôle de congestion de la prévenir ou de la supprimer. Aujourd'hui, aucun algorithme ne répond parfaitement aux exigences attendues, et de nombreux travaux de recherches sont en cours. Néanmoins, ces nouveaux algorithmes peuvent affecter l'équité du réseau étant donné que le comportement du protocole de transport peut changer radicalement en fonction de l'algorithme de contrôle de congestion utilisé dans les points finaux. De plus, durant ces dernières années, les protocoles de transport ont subi des évolutions majeures. Un exemple significatif récent est celui de Quick UDP Internet Connections (QUIC), un protocole introduit par Google, qui vise à remplacer deux protocoles de transport et de sécurité largement utilisés, à savoir Transmission Control Protocol (TCP) et Transport Layer Security (TLS). QUIC est implémenté dans les applications utilisateurs (plutôt que dans le noyau du système d'exploitation). Il se veut résistant à l'ossification et donc de ce fait il est plus versatile. Ceci rend alors les fournisseurs de contenus, comme Google, hégémoniques sur le débit de ses utilisateurs.

En raison du développement progressif des algorithmes de contrôle de congestion et de la nature évolutive des protocoles de transport, de nouveaux défis se posent en matière de gestion de l'équité. C'est pourquoi, dans cette thèse nous nous sommes orientés sur le développement d'une plateforme de tests pour mesurer l'équité réseau à partir du débit des différents flux. De plus, afin de caractériser l'équité telle que la perçoit un utilisateur, nous nous sommes concentrés sur la détermination d'une procédure impartiale d'évaluation de l'équité durant toute une session d'un flux de transport (nommée Session Fairness Assessment (SFA) et Weighted Session Fairness Assessment (WSFA)). A partir de ces éléments, nous avons analysé spécifiquement l'équité des protocoles lorsque les flux TCP et QUIC coexistent sur un réseau fixe et mobile. Lors de nos évaluations de l'équité, nous avons identifié l'impact des aspects de la mise en œuvre de QUIC tels que : l'émulation de connexions TCP multiples, la limitation de la taille des fenêtres de congestion et l'utilisation de l'option hybrid start (hystart). Les résultats montrent que ces mécanismes ont une forte influence sur l'équité que ce soit sur réseau fixe ou réseau mobile. En effet, un mauvais réglage des paramètres par défaut de ces mécanismes ou l'activation de l'option hystart peut affecter la performance des protocoles de transport et par conséquent l'équité. En ce qui concerne l'évaluation des algorithmes de contrôle de congestion, les résultats montrent que l'équité entre deux algorithmes différents dépend de la configuration du réseau. Cette conclusion démontre qu'une procédure de mesures, telle que celle qui a été présentée dans cette thèse, est pertinente pour réaliser l'évaluation de l'équité.

Dans cette thèse nous pouvons conclure que le manque de standardisation, par exemple de l'émulation de connexions TCP multiples dans QUIC nous amène à nous interroger plus largement sur la manière dont la philosophie de conception de QUIC tient compte de l'équité. De plus, les résultats obtenus sur l'évaluation de l'équité des algorithmes de contrôle de congestion, nous permet de remettre en cause l'évaluation de l'équité de plusieurs contributions lorsqu'elle n'est pas testée dans suffisamment de configurations réseau.