N° d'ordre : xx

Thèse

préparée au
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS
et à l'**École Nationale Supérieur d'Ingénieurs de Constructions Aéronautique**

en co-tutelle avec
l'**University of New South Wales, Sydney, Australie**
et **National ICT Australia, Australie**

Pour obtenir le titre de
DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE
École doctorale : Informatique et Télécommunications
Spécialité : Réseaux et Télécommunications

Par
M. Guillaume Jourjon

Toward a Versatile Transport Protocol

Soutenue le 23 Janvier 2008 devant le jury composé de :

Président :	Aruna	Seneviratne
Directeur de thèse :	Michel	Diaz
Co-Directeurs :	Patrick	Sénac
	Tim	Moors
	Emmanuel	Lochin
Rapporteurs :	Bjorn	Landfeldt
	Pascale	Vicat-Blanc Primet
	Chadi	Barakat
Membre :	Max	Ott
	Aruna	Seneviratne

Acknowledgement

Firstly, I would like to thank my four supervisors; Michel Diaz, Emmanuel Lochin, Patrick Sénac and Tim Moors for their support and their belief in my work. Tim accepted to supervise me in the context of a co-tutelle agreement which was sometimes difficult for both of us. Michel gave me some good advice for the latter sections of this thesis. Patrick gave me the opportunity to come to Australia in the first place, and always gave very interesting directions for all the articles we published together. Emmanuel was a strong support during these three years and we succeeded to build some good proposals during meetings organised in his "office".

The thesis would have never been possible without the help of Aruna Seneviratne. I am very grateful to him for accepting me into NICTA and for supporting me in obtaining the NICTA scholarship.

I would like to thank NICTA organisation for believing in me and supporting me financially during these three years. I have learnt a lot and was able to meet very interesting people inside and outside the organisation thanks to this support.

I would like to thank all the people of the NPC program (Seb, Max, Henrik, Thierry...) and all the people of the DMI/ENSICA (Tanguy, Jérome, Fabrice...). At the ENSICA I am very grateful to Laurent Dairaine who was my interim supervisor for one year.

I would like to give a special thanks to the people of Redfern (Fern, Jason, Jacques, Dave, Gersh...) who were constantly supportive, and helped me discover the inside of Australia.

Finally, I would like to thank my parents, my brothers and all my family who stayed in France instead of coming to Australia with me. I thank Yoann for all the time he refused to use my work and for the numerous hours he spent correcting some of my articles (especially the ten versions of "the optimisation"). I also thank Thea for correcting my English and simply being there.

Abstract

This thesis presents three main contributions that aim to improve the transport layer of the current networking architecture. The transport layer is nowadays dominated by the use of TCP and its congestion control. Recently new congestion control mechanisms have been proposed. Among them, TCP Friendly Rate Control (TFRC) appears to be one of the most complete. Nevertheless this congestion control mechanism, as with TCP, does not take into account either the evolution of the network in terms of Quality of Service and mobility or the evolution of the applications.

The first contribution of this thesis is a specialisation of TFRC congestion control to provide a QoS-aware Transport Protocol specifically designed to operate over QoS-enabled networks with bandwidth guarantee mechanisms. This protocol combines a QoS-aware congestion control, which takes into account network-level bandwidth reservations, with full ordered reliability mechanism to provide a transport service similar to TCP. As a result, we obtain the guaranteed throughput at the application level where TCP fails. This protocol is the first transport protocol compliant with bandwidth guaranteed networks.

At the same time the set of network services expands, new technologies have been proposed and deployed at the physical layer. These new technologies are mainly characterised by communications done without wire constraint and the mobility of the end-systems. Furthermore, these technologies are usually deployed on entities where the CPU power and memory storage are limited. The second contribution of this thesis is therefore to propose an adaptation of TFRC to these entities. This is accomplished with the proposition of a new sender-based version of TFRC. This version has been implemented, evaluated and its numerous contributions and advantages compare to usual TFRC version have been demonstrated.

Finally, we proposed an optimisation of actual implementations of TFRC. This optimisation first consists in the proposition of an algorithm based on a numerical analysis of the equation used in TFRC and the use of the Newton's algorithm. We furthermore give a first step, with the introduction of a new framework for TFRC, in order to better understand TFRC behaviour and to optimise the computation of the packet loss rate according to loss probability distributions.

Keywords: Transport Protocol, Congestion Control, Quality of Service, Light Architecture, Algorithmic Optimisation.

Résumé

Les travaux réalisés dans le cadre de cet axe de recherche ont pour but d'améliorer la couche transport de l'architecture réseau de l'OSI. La couche transport est de nos jours dominée par l'utilisation de TCP et son contrôle de congestion. Récemment de nouveaux mécanismes de contrôle de congestion ont été proposés. Parmi eux TCP Friendly Rate Control (TFRC) semble être le plus abouti. Cependant,tout comme TCP, ce mécanisme ne prend pas en compte ni les évolutions du réseau ni les nouveaux besoins des applications.

La première contribution de cet axe de recherche consiste en une spécialisation de TFRC afin d'obtenir un protocole de transport avisé de la Qualité de Service (QdS) spécialement défini pour des réseaux à QdS offrant une garantie de bande passante. Ce protocole combine un mécanisme de contrôle de congestion orienté QdS qui prend en compte la réservation de bande passante au niveau réseau, avec un service de fiabilité totale afin de proposer un service similaire à TCP. Le résultat de cette composition constitue le premier protocole de transport adapté à des réseaux à garantie de bande passante.

De concert avec l'expansion des services au niveau réseau, de nouvelles technologies ont été proposées et déployées au niveau physique. Ces nouvelles technologies sont caractérisées par leur affranchissement du support filaire induisant la mobilité des systèmes terminaux. De plus, les méthodes d'accès à des réseaux sans fil sont généralement déployées sur des entités où la puissance de calcul et plus généralement les ressources systèmes sont inférieures à celles des ordinateurs personnels traditionnellement connectés aux réseaux filaires. La deuxième contribution de ce travail de recherche consiste en la proposition d'une adaptation de TFRC à ces entités via la définition et la mise en œuvre d'une architecture de TFRC centrée sur l'émetteur et réduisant de façon très sensible les traitements opérés par le récepteur des flus TFRC. Cette version a été implémentée, évaluée quantitativement et ses nombreux avantages et contributions ont été démontrés par rapport à une implémentation traditionelle de TFRC.

Enfin, nous avons proposé une optimisation des implémentations actuelles de TFRC. Cette optimisation repose tout d'abord sur un nouvel algorithme pour l'initialisation du récepteur basé sur l'utilisation de l'algorithme de Newton. Nous proposons aussi l'introduction d'un outil nous permettant d'étudier plus en détails la manière dont est calculé le taux de perte du côté récepteur.

Mots-Clés : Protocole de transport, Contrôle de congestion, Qualité de Service, Architecture légère, Optimisation algorithmique.

Table of Contents

Chapter 1

Résumé de la thèse en français

1.1 INTRODUCTION (CHAPITRE 2)

1.1.1 Contexte

Durant les années 90 la recherche en réseau s'est focalisée sur la qualité de service offerte aux utilisateurs par le biais de modifications successives de la couche réseau [BCS94, BBC+98, RVC01]. Pour des raisons relatives à leur déploiement et aux politiques inter-fournisseurs d'accès ces solutions ne furent jamais mises en place de manière globale.

Ces cinq dernières années ont cependant vu émerger de nouveaux services applicatifs pour les utilisateurs. Ces services s'appuient sur la même base protocolaire, à savoir TCP/IP. Ces nouveaux services ont été rendus disponibles grâce à l'augmentation de la capacité du réseau et ont été développés au niveau applicatif. Ils englobent entre autres les réseaux pair à pair (P2P), le web 2.0, ou encore la vidéo et la voix sur IP.

En parallèle de ces nouveaux services applicatifs, de nouvelles technologies de communication sans fil ont été développées au niveau de la couche physique. Ces nouvelles technologies englobent les standards IEEE 802.11* [Soc07] et 802.15* [Soc05], ou encore la 3G [Uni99]. Ces nouvelles architectures se sont affranchies de la connectivité filaire. Par conséquent, de nouveaux problèmes relatifs à la mobilité des systèmes terminaux sont à prendre en compte lors de la conception de nouveaux protocoles des couches supérieures. Une des principales différences avec la précédente architecture filaire réside dans le fait que les pertes identifiées de bout en bout sont différentes de celles détectées dans la précédente architecture filaire. En effet, dans la précédente architecture la détection d'une perte est presque toujours synonyme de congestion. Dans le contexte des communication sans fil, cette interprétation est potentiellement erronée car la perte correspond souvent à un obstacle dans la transmission radio ou à un manque de couverture de l'antenne relais.

Au contraire de ces avancées au niveau de couches applicatives et physiques, peu de changements existent du côté des protocoles réseaux et transports. En effet, Internet utilise toujours la version 4 du protocole IP et la plupart des communications sont faites avec le protocole TCP et ses diverses versions. Toutefois, de nouveaux protocoles de transport ont récemment été standardisés. Les deux protocoles de transport nouvellement standardisés sont DCCP [KHF06] et SCTP [SA00]. Ces deux propositions diffèrent de TCP par le fait qu'elles sont basées sur une communication par datagrammes plutôt que par flux d'octets. De plus, dans le cas de DCCP, deux mécanismes de contrôle de congestion sont proposés ; un basé sur une fenêtre comme dans TCP, le second est un mécanisme de contrôle de congestion qui n'est pas basé sur l'utilisation d'une fenêtre est offert. Ce mécanisme de contrôle de congestion se nomme TCP Friendly Rate Control (TFRC). TFRC utilise une équation modélisant le comportement de TCP Reno afin de transmettre les paquets d'une manière similaire à TCP.

1.1.2 Contributions de cette thèse

Cette thèse vise à combler le fossé existant entre les nouveaux services de niveau applicatif et les couches réseau et physique. Ceci est en particulier réalisé par la proposition de nouveaux mécanismes de contrôle de congestion qui puissent tenir compte soit de la qualité de service offerte par le fournisseur d'accès à Internet, soit de la capacité de l'entité sur laquelle la communication s'effectue. Cette thèse s'appuie sur le mécanisme de contrôle de congestion TCP Friendly Rate Control (TFRC). Ce mécanisme constitue à ce jour l'une des alternatives au contrôle de congestion de type AIMD, tel celui utilisé dans TCP, la plus aboutie pour les applications multimédia.

Dans un premier temps, cette thèse propose une spécialisation de TFRC afin de pouvoir tenir compte des garanties de bande passante préalablement négociée avec le fournisseur de service réseau. Nous démontrons que cette proposition permet d'obtenir la qualité de service négociée quelles que soient les conditions du réseau. En effet, de nombreuses études ont montré que lors de l'utilisation de TCP sur ce type de réseau, l'utilisateur ne pouvait pas obtenir le service qu'il avait payé lorsque certaines conditions du réseau étaient présentes [SNP99]. En plus de cette amélioration, nous avons intégré un mécanisme de fiabilité, basé sur une adaptation de SACK [FMMP00], permettant de délivrer ainsi un service similaire à TCP. Le résultat de cette composition constitue le premier protocole de transport fiable spécialement conçu pour les réseaux à garantie de bande passante.

La seconde contribution de cette thèse consiste à proposer une adaptation de TFRC aux hôtes mobiles. Cette nouvelle adaptation permet un allègement du receveur en termes d'utilisation de la mémoire et du processeur. En effet, de nos jours de plus en plus de communications sont effectuées grâce aux périphériques sans fil et mobiles. Néanmoins, les entités mobiles (PDA, téléphone portables, etc.) n'ont pas la même capacité de puissance de calcul et de mémoire disponibles que les ordinateurs standards. C'est pourquoi une adaptation des tâches récurrentes de communication et en particulier du mécanisme de contrôle de congestion est nécessaire afin d'obtenir de meilleures performances.

Enfin, la dernière contribution de cette thèse propose une analyse et une optimisation du mécanisme TFRC. Cette analyse a pour objectif de mieux comprendre les différents com-

posants impliqués dans le calcul de la probabilité de perte durant la communication. Nous proposons par ailleurs une optimisation de l'initialisation de TFRC du côté récepteur grâce à une analyse numérique de l'équation régissant le taux de transfert et par l'application à l'inversion de cette équation de l'algorithme de Newton. Cette optimisation nous permet d'améliorer d'un facteur non négligeable la rapidité de convergence vers la solution ainsi qu'un allègement de la charge de calcul et de l'utilisation de la mémoire du côté récepteur.

1.1.3 Plan de cette thèse

Cette thèse est organisée comme suit :
- le chapitre 3 définit le contexte de cette thèse en donnant une vue d'ensemble des précédents travaux sur les architectures réseaux pour la qualité de service, les protocoles de transport et des contrôles de congestion ;
- le chapitre 4 présente la conception et la mise en œuvre d'un protocole de transport avisé de la qualité de service capable de fournir à l'utilisateur la bande passante négociée avec le fournisseur d'accès ;
- le chapitre 5 présente et évalue une version allégée du contrôle de congestion TFRC ;
- le chapitre 6 propose une analyse de ce contrôle de congestion ;
- enfin, le chapitre 7 donne les principales conclusions de ce travail ainsi que ses perspectives.

1.2 LE CONTEXTE (CHAPITRE 3)

L'augmentation de la capacité de transmission des systèmes de bordure et des réseaux de communication a grandement accéléré le développement des systèmes distribués. A l'origine, les applications distribuées étaient caractérisées par des besoins modestes en matière de communication, principalement liés à l'ordonnancement et la fiabilité des paquets. Aujourd'hui, les applications multimédia sont de plus en plus utilisées et demandent de fortes garanties en termes de délais et bande passante disponible.

Afin de pallier à ces contraintes, dans un premier temps, les recherches se sont focalisées sur la définition de nouvelles architectures réseau. Néanmoins, les architectures proposées n'ont toujours pas encore été déployées. La première architecture proposée fut l'architecture *Integrated Service (IntServ)* [BCS94]. Une autre architecture visant à résoudre les problèmes de résistance au passage à l'échelle de l'architecture Intserv est connue sous le nom *Differentiated Service (DiffServ)*. Des contributions intervenant dans les couches 2 et 3 ont été introduites visant à introduire un contrôle de la qualité de service comme dans l'architecture MPLS [RVC01]. Nous ne reviendrons pas sur ces architectures qui ont été largement étudiées [Loc04, Del00]. Récemment une nouvelle architecture a été proposée dans le cadre d'un projet européen, le projet EuQoS. Cette nouvelle architecture propose une virutalisation des ressources et des services afin de proposer aux utilisateurs certaines qualité de services.

Néanmoins, toutes ces propositions ont été faites sans se soucier des besoins des couches supérieures et des architectures de communication. Cette absence de cohésion entre les couches a résulté en l'introduction de problèmes complexes afin d'obtenir la qualité de

service (QdS) négociée. En effet les protocoles de transport et les mécanismes de QdS mis en œuvre dans la couche réseau ne prennent généralement pas en compte les mêmes informations et ne sont pas au courant d'émettre au dessus d'un réseau à QdS.

Un mécanismes clef des protocoles de transport est le contrôle de congestion du réseau. Ce mécanisme est indispensable pour la pérennité de l'Internet. En effet, il permet à différents flux de partager équitablement la bande passante disponible afin de préserver le principe d'équité cher à l'Internet. Cependant, ces mécanismes de contrôle de congestion s'affranchissent de la qualité de service au niveau réseau et applicatif et visent uniquement à préserver l'état du réseau. Actuellement, deux types de contrôle de congestion sont déployés : soit basés sur une fenêtre, soit sur un taux de transfert. Les mécanismes de contrôle de congestion utilisant une fenêtre émettent les paquets selon un nombre calculé de paquets qui est mis à jour par les messages de contrôle et s'arrêtent lorsque ce nombre de paquets est emis. Les mécanismes utilisant un taux de transfert considèrent l'émission des paquets selon un temps inter-paquets, ce taux est mis à jour lui aussi par les messages de contrôle. Par conséquent, dans le cas de l'utilisation d'un mécanisme de contrôle de congestion à taux le transfert continue même si l'émetteur ne reçoit pas de mises à jour. Dans le contexte de cette thèse, nous nous intéressons particulièrement à un certain contrôle de congestion utilisant un taux, connu sous le nom de *TCP Friendly Rate Control*. Et nous proposons dans un premier temps une spécialisation de ce mécanisme afin de le rendre avisé de la qualité de service garantie par le réseau. Par la suite nous adaptons ce mécanisme aux hôtes mobiles qui ont des ressources limitées. Enfin nous proposons une optimisation algorithmique de ce mécanisme.

Dans ce chapitre, nous donnons une vue d'ensemble des différents services et architectures réseaux ainsi qu'un bref résumé des causes de leur non déploiement. Ensuite, nous présentons un état de l'art des protocoles de transport. Enfin, nous présentons les différents contrôles de congestion basés sur une notion de taux de transfert ainsi que leurs limitations.

1.3 CONCEPTION ET IMPLÉMENTATION D'UN PROTOCOLE DE TRANSPORT À QDS (CHAPITRE 4)

De nos jours, les applications multimédia sont de plus en plus utilisées et requièrent des garanties en termes de délais et de bande passante. Plusieurs solutions sont disponibles au niveau réseau afin d'assurer une certaine qualité de service. Dans le cadre du projet EuQoS, nous nous sommes intéressé aux deux classes de service qui garantissent une bande passante à l'application : la classe *High Trhoughput Data* et la classe *Multimedia Streaming*. Ces deux classes de services peuvent etre obtenues dans le cadre d'un déploiement de l'architecture DiffServ [HBWW99] par le service de la classe *Assured Forwarding (AF)* [BCB06]. C'est pourquoi dans un soucis de simplicité nous nous plaçons tout au long de ce chapitre seulement l'architecture DiffServ et plus spécialement la classe *Assured Forwarding (AF)*. Nous montrons également en fin de chapitre que la solution proposée est également compatible avec d'autres architectures de garantie de service. Dans le cadre d'un service AF, une bande passante assurée minimum (aussi appelée taux cible) est procurée à l'application en accord avec le profil de l'utilisateur et le fournisseur de service. Cette

garantie de bande passante minimale est particulièrement adaptée pour les applications de type multimédia ou à forte contrainte de bande passante [BCB06].

La plupart des applications utilisant Internet sont supposées s'adapter à l'état du réseau et utilisent TCP [Pos81] comme moyen pour transférer leurs données. TCP offre un service orienté flux offrant une fiabilité totale et un ordre total. De plus, TCP comprend un contrôle de flux et un contrôle de congestion afin d'éviter le débordement du tampon au niveau du récepteur et la congestion du réseau. Malgré le bon comportement de TCP en termes d'utilisation des ressources réseau et de partage de la bande passante, TCP s'avère inadapté pour beaucoup d'applications ayant des contraintes de délais et de la bande passante et cela sans pour autant nécessiter une fiabilité totale.

Une alternative classique à l'utilisation de TCP est UDP [Pos80]. UDP peut-être considéré comme un protocole de transport minimaliste qui ne fournit aucune fiabilité ni ordonnancement ni contrôle de congestion. Les applications utilisant UDP devraient proposer au niveau utilisateur les contrôles précédents afin de partager équitablement le réseau avec TCP. Le protocole DCCP (*Datagram Congestion Control Protocol*) a été récemment standardisé par l'IETF [KHF06] et offre un service non fiable aux applications tout en mettant en place un contrôle de congestion contrairement à UDP. De fait il constitue un substitut pour les applications utilisant UDP car il combine l'efficacité et la légèreté d'UDP avec un contrôle de congestion compatible avec TCP. Un des mécanisme mis en place dans DCCP pour assurer le contrôle de congestion est *TCP Friendly Rate Control* (TFRC) [HFPW03]. Ce mécanisme est présenté en détails dans le chapitre 3.

L'utilisation de protocole de transport réactif à l'état du réseau, tel TCP, au dessus d'un réseau à garantie de service, ne permet cependant pas une utilisation optimale des ressources. En effet, plusieurs études ont montré que TCP ne pouvait pas toujours obtenir la bande passante négociée [SNP99, PC04a]. Dans le cas particulier d'utilisation de la classe AF de DiffServ, une bande passante minimale est fournie (appelée *in-profile traffic part*), tout en permettant aux flux d'avoir une bande passante supérieure à celle négociée (appelée *out-profile traffic part*). Néanmoins, de manière similaire à TCP au dessus de DiffServ/AF, TFRC n'utilise pas le service offert de manière optimale et produit des résultats en deçà de ce que l'application pourrait espérer. En effet, comme TFRC modélise le contrôle de congestion de TCP, son comportement est identique en moyenne à celui de TCP fonctionnant au dessus de DiffServ/AF.

Dans ce chapitre, nous nous intéressons au comportement de TFRC au dessus de la classe AF de DiffServ. Nous montrons tout d'abord que notre implémentation est conforme aux implémentations standards de TFRC et par conséquent conserve les bonnes propriétés de TFRC, à savoir son caractère plus stable en termes d'oscillation du taux de transfert et le partage équitable de la bande passante lorsqu'il est en concurrence avec TCP ou TFRC. Néanmoins, nous montrons que TFRC seul ne profite pas pleinement de la garantie de bande passante offerte par la classe AF. Cet échec à l'obtention de la garantie est due à la forte dépendance de TFRC envers le RTT et le taux de perte, ainsi que le fait que TFRC ne puisse pas différencier la perte d'un paquet marqué *in-profile* d'un paquet marqué *out-profile*. Afin de résoudre ce problème, nous présentons dans ce chapitre une modification de TFRC nommée *g*TFRC. Cette modification permet à l'application d'obtenir la bande

passante négociée quelle que soit le RTT et la valeur de la bande passante. Nous validons ce nouveau mécanisme à l'aide d'une implémentation et une campagne de mesures.

Suivant cette première validation, nous étendons le service offert à l'application en combinant ce contrôle de congestion avec un mécanisme de fiabilité et un contrôle de flux spécialement conçus pour des protocoles utilisant un taux de transfert comme contrôle de congestion. Le mécanisme de contrôle d'erreur utilisé est une adaptation du mécanisme *Selective ACKnoledgement* (SACK) pour un protocole utilisant des datagrammes. Ce mécanisme nous permet, grâce aux informations qu'il contient, de proposer dans le chapitre 5 une nouvelle architecture pour TFRC afin de s'adapter aux hôtes mobiles et légers. Il résulte de cette composition de mécanismes le premier protocole de transport fiable spécialement conçu pour des réseaux à garantie de bande passante.

Ce chapitre est organisé de la manière suivante. La section 4.2 donne un état de l'art concis des travaux relatifs à la relation de DiffServ/Af avec les divers mécanisme de contrôles de congestion. La section 4.4 présente le problème et notre solution *g*TFRC. La section 4.5 détaille l'implémentation et la validation de ce nouveau mécanisme grâce à l'utilisation d'un framework en Java introduit au chapitre 3. Basée sur cette première implémentation nous présentons dans la section 4.6 l'implémentation d'un protocole de transport complet orienté QdS. Ce protocole est évalué au dessus d'un réseau utilisant DiffServ/AF ainsi que d'autres mécanismes génériques garantissant une bande passante. Enfin, la section 4.7 propose une conclusion pour ce chapitre ainsi que les possibles travaux futurs.

1.4 NOUVELLE APPROCHE POUR UN TFRC ORIENTÉ ÉMETTEUR (CHAPITRE 5)

Dans le chapitre précédent nous avons montré comment la composition d'un mécanisme de contrôle de congestion basé sur une modification de TFRC et d'un mécanisme de contrôle d'erreur similaire à SACK permettait d'obtenir le débit négocié dans le cadre de réseau à garantie de débit. Grâce aux informations fournies par cette composition nous proposons dans ce chapitre une nouvelle architecture permettant d'alléger le mécanisme TFRC du côté récepteur. En effet, nous savons que TFRC produit un débit plus stable que TCP ce qui fait de lui un bon candidat pour le transfert multimédia et le streaming. Néanmoins dans le scénario d'une communication client-serveur utilisant TFRC, si les serveurs multimédia sont de puissantes machines en matière de calcul et de débit de sortie, cela n'est pas le cas pour des clients mobiles. En effet, ces clients sont des entités aux ressources limitées qui posent le problème de l'optimisation de ces ressources en particulier pour des tâches systèmes récurrentes et des tâches de communication.

Dans ces conditions, l'allègement des ces processus est critique pour l'amélioration des performances des systèmes mobiles autonomes. Un des principaux coûts du mécanisme TFRC est le calcul périodique du temps aller-retour (RTT) et du taux de perte de paquet de la communication. En particulier, la RFC 3448 [HFPW03] propose que l'estimation de ce taux de perte soit faite du côté receveur. Ce standard suggère aussi que ce calcul puisse être fait du côté émetteur.

Nous avons donc développé cette idée en spécifiant et évaluant une mise en œuvre de TFRC orienté émetteur. Dans cette proposition, le transfert fiable des paquets de contrôle est assuré par l'utilisation d'un mécanisme similaire à SACK [FMMP00]. Ce mécanisme est reconnu pour sa robustesse lors de communications dans des canaux à pertes car cette robustesse permet d'éviter la mise en place de mécanismes de contrôle d'erreur trop complexes [FMMP00]. De plus, grâce à sa migration sur les serveurs de flux, l'architecture orientée émetteur proposée est robuste face aux receveurs opportunistes et résout un problème de sécurité identifié dans la RFC 3448. Ce problème de sécurité hérite du fait que le récepteur renvoie à l'émmetteur la valeur du taux de perte de la communication. Dans le but de recevoir une meilleure bande passante, un récepteur mal intentionné pourrait sous évaluer ce taux. Grâce à une architecture orienté émetteur, le serveur n'est plus dépendant de la précision et de la véracité des informations renvoyées par le receveur. D'autre solutions ont été proposées afin de sécuriser TFRC contre ces receveurs opportunistes dans [GG05] en utilisant RTSP [SRL98]. Notre solution requiert moins de modifications, et qui plus est des modifications plus simples, pour l'entête des messages et l'algorithme de TFRC.

Une autre solution introduisant un TFRC orienté émetteur a été proposée dans [FKP06]. Cette solution requiert de la part du receveur l'envoi dans les paquets de contrôle des intervalles d'évènement de pertes. A notre connaiscance, cette solution n'a jamais été ni implémentée ni testée. De plus, comparée à notre solution, cette proposition est supposée être plus proche du mécanisme original mais le receveur reste plus complexe car il doit toujours maintenir une structure permettant de différencier une perte de paquet d'un évènement de perte.

Ce chapitre est organisé comme il suit : nous résumons brièvement le contexte de cette étude. Ensuite nous expliquons en détails notre nouveau dessein et le cheminement qui nous a amené à celui-ci. Nous validons par la suite une première implémentation de cette architecture en la comparant avec TFRC et TCP. Cette validation est mise en œuvre grâce à, dans un premier temps, une étude qualitative de notre proposition et, dans un deuxième temps, des métriques nous permettant de quantifier notre proposition. Enfin nous quantifions le gain en matière de cycle de processeur et le gain en mémoire introduit par cette architecture. Finalement nous proposons les perspectives possibles à cette étude.

1.5 COMPRÉHENSION ET OPTIMISATION SUR LE MÊME THÈME (CHAPITRE 6)

Les chapitres précédents ont permis de présenter deux modifications architecturales de TFRC. Ces deux contributions ont pour but de permettre une spécialisation de l'actuel TFRC à des réseaux à garantie de service ainsi qu'une adaptation à la capacité matérielle de l'entité sur laquelle la communication est effectuée. Ces propositions n'effectuent aucune modification notoire des variables statiques des implémentations traditionnelles de TFRC. Ces variables sont utilisées pour l'estimation du délai dans la communication ou encore de l'estimation du taux de perte de paquets dans la transmission.

Les deux principales variables sont la valeur initiale de l'initialisation du taux de perte et l'ensemble des poids régissant la structure *loss history*. Ces deux variables sont responsables du calcul du taux de perte et par conséquent du taux d'émission de paquets.

L'initialisation de la structure *loss history* est essentielle pour l'établissement de la communication. Quand un évènement de perte est détecté cette structure doit donc être initialisée. Cependant, le nombre de paquets reçus depuis le début de la communication ne représente pas l'état du réseau. Afin de réaliser cette initialisation, les implémentations actuelles doivent donc inverser l'équation utilisée par TFRC. Cependant le degré maximum de cette équation rend impossible une inversion analytique. Nous proposons de résoudre ce problème par l'introduction d'un algorithme qui améliore l'algorithme de résolution utilisé habituellement.

La seconde variable responsable du calcul du taux de perte dans TFRC est constituée de l'ensemble des poids appliqués à la structure *loss history*. Ces poids peuvent être modifiés mais sont généralement configurés comme proposés dans [HFPW03]. Nous proposons un outil permettant d'étudier la relation entre le taux de perte calculé et celui donné par un modèle de perte du réseau. Cet outil est basé sur un programme d'analyse numérique utilisant des évènements discrets qui nous permet d'intégrer facilement plusieurs distributions de probabilité afin de comparer les résultats obtenus en sortie de TFRC avec les résultats théoriques liés à ces distributions.

Dans ce chapitre nous proposons dans un premier temps une optimisation de l'initialisation de la structure *loss history* en utilisant une méthode numérique appliquée à l'équation de TCP. Ensuite nous étudions la relation entre le calcul du taux de perte et différents schémas de perte. Enfin nous concluons et donnons les perspectives de ces améliorations.

1.6 CONCLUSION (CHAPITRE 7)

Ce chapitre conclue cette thèse. Nous proposons dans une première partie un résumé des problèmes auxquels nous avons été confrontés. Ces problèmes ont été mis en exergue tout au long des chapitres de cette thèse et peuvent être synthétisés en trois points :
- l'échec à l'obtention du débit négocié avec le réseau par les protocoles de transport actuels ;
- le besoin pour des entités à faibles capacités de calcul et de mémoire d'alléger les tâches récurrentes ;
- le besoin d'optimiser les implémentations actuelles des protocoles de transport.

Bien que ces problèmes puissent à première vue paraître décorrélés, nous montrons qu'ils peuvent être résolus par une approche commune. En effet, afin de résoudre ces problèmes nous nous sommes basés sur le mécanisme de contrôle de congestion TFRC. Ce mécanisme a dans un premier temps été spécialisé afin de résoudre le problème de la non-obtention de la garantie dans le cadre de réseaux à QdS. Par la suite, nous avons proposé une adaptation de ce mécanisme pour les hôtes légers et mobiles. Enfin une optimisation d'algorithmes internes à ce mécanisme a été proposée et quantifiée.

Dans la deuxième partie de ce chapitre nous présentons un résumé des contributions originales dont est constituée cette thèse. Cette partie repose sur les chapitres 4, 5 et 6. Les solutions décrites dans ces chapitres visent à résoudre les problèmes précédemment cités. Cette résolution repose sur la définition de nouveaux mécanismes de contrôle de congestion pour les deux premiers problèmes et d'une étude du socle commun, TFRC, pour le troisième. Dans le cas de la difficulté pour les protocoles de transport d'obtenir la bande

passante garantie avec des réseaux à garantie de service, nous modifions, en effet, le mécanisme TFRC afin qu'il puisse tenir compte de la QdS préalablement négociée avec le réseau. Cette modification est par la suite composée avec un mécanisme de contrôle d'erreur et de flot. De cette composition découle la solution pour le second problème. En effet, le mécanisme de contrôle d'erreur nous permet, par le biais des informations nécessaires à son fonctionnement, de modifier l'architecture de TFRC en déplaçant le calcul du taux de perte vers le serveur. Tout au long de ces études, nous avons aussi tenté d'optimiser les méthodes internes de TFRC.

Enfin, la dernière partie de ce chapitre propose les orientations possibles qui de ces travaux de recherche. Ces perspectives incluent :

- un déploiement sur un réseau *multi-hop* de la contribution concernant l'allègement de TFRC en y rajoutant l'option de contrôle d'erreur et une étude plus approfondie de la relation entre le calcul du taux de perte dans TFRC et le taux de perte réel dans le réseau ;
- une étude de l'impact de l'initialisation de la structure *loss history* sur le reste de la communication ;
- enfin, en nous basant sur des travaux précédents et grâce à l'expérience acquise tout au long de cette thèse, nous pensons qu'une adaptation de la couche transport avec des applications de calcul sur réseau pair à pair est possible et serait bénéfique à la fois pour ces applications particulières et l'utilisation générale du réseau.

Chapter 2

Introduction

2.1 CONTEXT OF THIS THESIS

During the nineties, numerous studies in networking focused on the improvement of the quality of service (QoS) offered to the end user. This research has lead to the development of new network architectures. Problems in the deployment of these architectures have made them unavailable to the end-user. Some of these architectures have been standardised at the IETF[1], for example the Integrated Service (IntServ) [BCS94], the Differentiated Service (DiffServ) [BBC+98] and the Multi-Protocol Label Switching (MPLS) [RVC01]. Lately some projects, such as the EuQoS project [MBYSG+07], have proposed new architectures based on the virtualization of network resources and services for providing QoS to the end-user. The issues of QoS control in the network layer will be discussed in chapter 3.

In the last five years, new application services, based on former IP technology, have been made available to the user. These new services have been made possible due to the increase in network capacity and have mainly been developed at the application layer. These new services embraced peer to peer (P2P) networks, web 2.0, TV and Voice over IP. These services still use the common TCP/IP stacks.

In parallel to these two improvements, new technologies have been made accessible to the user at the physical layer. These technologies, such as wireless 802.11* [Soc07] or 802.15* [Soc05], or 3G [Uni99], introduced mobility and wireless capability in the access networks. These new technologies are no longer constrained by a wired architecture. As a result, new problems linked to the mobility of end-systems have to be considered when designing new network and transport protocols. One of the main differences with the previous network architecture is the interpretation of what the end-systems should do about packet losses. In the previous architecture when a loss occurs in the communication, reactive transport protocols interpret this loss as congestion in the network. Nevertheless, in mobile or

[1] Internet Engineering Task Force

wireless access networks, losses can also be entailed by handovers, an obstacle in the radio transmission or a loss of coverage.

Linked to these new network technologies, an evolution of the end-terminals has also occurred. Indeed, end-systems have become progressively lighter and more mobile. This emerging new generation of end-systems intrinsically offers limited processing, storage and power capacities.

Nevertheless, during this time few changes have been made to the core of the Internet architecture (i.e. transport and network layer). The dramatic evolution of the two extreme protocol layers of the Internet coupled with the ossification of the TCP/IP core protocols introduced a hourglass shape protocol stack that is not anymore able to deliver an efficient adaptation between new application service and physical layer. This can be synthesised by the Figure 2.1 and the Figure 2.2.

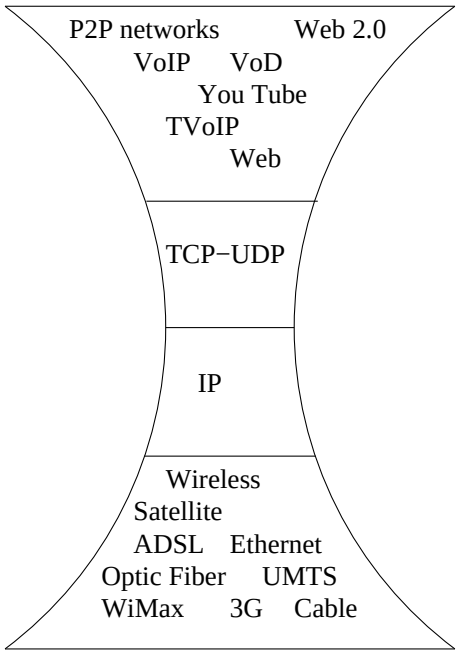


Figure 2.1 Illustration of today's Internet

In order to make this adaptation between the application layer and the physical layer, the transport layer has a great role to play since it is localised at the end systems and therefore it is easier to deploy than the network layer. In the transport layer the congestion control mechanism is one of the main mechanism. Indeed, the congestion control allows to avoid the Internet from collapsing and different flows to share fairly the available bandwidth. Nevertheless, actual congestion control mechanisms only focus on the network congestion and do not take into account neither the application's needs or the new network services.

In order to solve this adaptation issue, new transport protocols, such as DCCP [KHF06] or SCTP [SA00], have been proposed for standardisation. Nevertheless, these new proto-

Figure 2.2 New communication landscape

cols offer limited adaptation capabilities and are not yet implemented in most operating systems. These new protocols are characterised by the use of datagram oriented communication instead of byte-stream oriented communication as in TCP. Furthermore in the case of DCCP, a new kind of congestion control is provided. This congestion control mechanism, named TCP Friendly Rate Control [HFPW03], is no longer based on a window but is instead based on the use of an equation that feeds a rate-based control. TFRC can be considered as a first step for the adaptation to the multimedia application since the throughput obtained is smoother than the one obtained in the same condition with a TCP-like congestion control. Therefore this throughput is better adapted to multimedia streaming [FHPW00].

2.2 CONTRIBUTION OF THIS THESIS

This thesis aims to mitigate the bottleneck as depicted in Figure 2.1 by proposing congestion control mechanisms that allow transport protocols to better take into consideration either the QoS subscribed by the user or the limitations of the device on which the communication is performed. In this thesis, we use the TFRC congestion control mechanism since this currently appears to be the congestion control the most adapted to encompass the multimedia applications but still stays limited. This mechanism will therefore be specialised for QoS network, then an adaptation to lighten and mobile entities will be proposed, and finally an optimisation of internal algorithms will provided.

The first contribution of this thesis consists of a specialisation of TFRC in order to make this mechanism aware of the bandwidth guaranteed by the network service provider. We show that this specialisation of TFRC makes possible the obtention of the negotiated bandwidth regardless of network's conditions. In addition, we have extended TFRC with an error control mechanism based on a Selective ACKnowledgement (SACK) mechanism [FMMP00], and a flow control specially designed for rate-based transport protocol in order to provide a TCP-like reliable service. The result appears to be the first reliable transport mechanism especially designed for bandwidth guaranteed networks.

The second contribution of this thesis is an adaptation of TFRC to handheld devices. This new TFRC architecture lightens the load on the communication receiver in terms of CPU and memory use. Indeed, nowadays, more and more communication is performed using wireless handheld devices. Nevertheless, these mobile entities (PDA, mobile phone) do not have the same capacity as personal computers in terms of CPU power and memory storage. That is why an adaptation of communication recurrent tasks and in particular the congestion control is necessary for better performances and resources usage optimisation.

The final contribution of this thesis consists of an analysis and an optimisation of TFRC. This analysis allows to better understand the different components involved during a session and specially how the loss event rate is computed. Furthermore, we proposed an optimisation of the initialisation of the TFRC receiver. This optimisation is done through a numerical analysis of the equation in use in TFRC for the rate computation and the application of a Newton algorithm for the inversion of this equation. This optimisation allows us to converge faster to the solution and to reduce the CPU and memory usage.

In the rest of this thesis, we therefore present a transport protocol, its architecture and implementation, named the Chameleon Protocol (CP). This protocol can be configured with the previously introduced mechanisms. Chameleon Protocol is based on the networking by component paradigm that makes it possible to compose a rate based congestion control with a reliability mechanism enhanced with a flow control that is adapted to rate based congestion control.

2.3 ORGANISATION OF THIS THESIS

This thesis is organised as follows:

- chapter 3 defines the context of this thesis by giving an overview of the previously introduced QoS architecture, transport protocols, and congestion control mechanisms;
- chapter 4 presents the design and the implementation of a QoS-aware transport protocol able to provide the bandwidth negotiated with the provider;
- chapter 5 presents and quantifies a lightweight version of TFRC congestion control;
- chapter 6 proposes an analysis of this particular congestion control;
- finally, chapter 7 presents conclusions and future directions for the presented work.

Chapter 3

Context

3.1 INTRODUCTION

The increasing capabilities of high performance end systems and communication networks have greatly accelerated the development of distributed computing. Distributed applications were originally characterized by very basic communication requirements mainly related to full packet reliability and order. Today, multimedia applications are in widespread use and require delay and bandwidth guarantees. As we have described in the chapter 2, these new constraints have been partially tackled by the application and the network layers, but an adaptation of the transport layer remains necessary. In the transport layer, we have seen in the previous chapter that the congestion control mechanism appears to be one of the key mechanisms for this adaptation. We will, in this chapter, give an overview of these three layers with a particular focus on the QoS they can offer and we will finish this chapter with a state of the art of the current congestion control mechanisms and their limitations.

In order to fulfill these constraints, researchers initially focused on defining a new architecture for the underlying network. The first proposed architecture has been the Integrated Service (IntServ) architecture [BCS94]. Because of the problem of deployment of IntServ due to the per flow reservation and scalability problems, another architecture has been proposed, the Differentiated Service architecture [BBC+98]. Furthermore, architectures in between levels 2 and 3, such as the MPLS services [RVC01], have been standardised. Recently, another architecture has been proposed in the context of a European project; the EuQoS network[1]. EuQoS is based on the virtualization of network resources and provides QoS through a per domain management mixed with end to end signalling process.

Nevertheless, all of these proposals have been studied and generated without consideration of the upper layer of the networking architecture and in particular the transport layer, which aims at applying an efficient adaptation between the applications needs and the net-

[1] http://www.euqos.eu/

work state and services. This lack of communication between network layer and transport layer has resulted in the incapability to efficiently map the service needs into a complete protocol stack that allow to ensure a rigorous QoS control. In this chapter, we present new propositions for transport protocols. The first two have been standardised at the IETF[2]. The other propositions are still in the experimental stage. Furthermore we focus on FPTP, a transport protocol framework that will be used in this thesis for the implementation of our new propositions.

Congestion control is one of the main transport protocol mechanisms. This mechanism is necessary for the sake of the Internet fairness both currently and in the near future. It allows different flows to share link bandwidth and router buffer space. Currently there are two main kinds of congestion control, the window-based and the rate-based. In the context of this thesis we mainly focus on the improvement of a particular rate based congestion control in order to provide a complete transport protocol.

In this chapter, we will give an overview of the actual mechanisms for the deployment of QoS in the current protocol stack. We first give in Section 3.2 a brief presentation of the mechanisms that have been proposed to enhance the layer application/session. We then give 3.3 an overview of the different network services and architectures and brief summary of the problems of deploying them. We present in Section 3.4 the main transport protocols that have been defined by the IETF and new directions in transport protocol research. Section 3.5 presents different kinds of congestion control implemented inside these protocols and discusses their limitations. Finally Section 3.6 concludes the chapter.

3.2 APPLICATION/SESSION LAYER EVOLUTION

In this section, we present the different application and session protocols for the description and signalling of multimedia applications. In the first time, we present briefly application multimedia profile defined by the International Telecommunication Union (ITU) in the framework F.700 [IT04]. We then present RTP and its dual control protocol RTSP and then we present two signalling protocols that establish multimedia session, SIP and NSIS.

3.2.1 Application Framework and Constraints for Multimedia Services

The ITU-T Recommendations F.700 and G.1010 [IT04, IT96] provide user requirements and communication parameters concerning the QoS level of a multimedia application. In particular, in F.700, different QoS levels are defined according to the nature of medium. As a counter part, the recommendation G.1010 uses time metric and loss rate to quantify the QoS of different applications.

3.2.2 RTP/RTCP

With the evolution of the multimedia applications, new information became necessary. In order to provide this service, the Real-time Transport Protocol (RTP) and its informa-

[2]Internet Engineering Task Force

tional companion protocol the Real-Time Control Protocol (RTCP) have been proposed [SCFJ96].

These two session protocols are usually used on top of the UDP protocol [Pos80]. They aim to provide to the application information about the multimedia stream. In the case of RTP, these information concern the kind of data used in the stream (e.g. codecs), a application data unit sequence number, a time stamp, the source ID corresponding to this stream in order to differentiate for example the audio and the video stream and other optional information. Nevertheless, this protocol does not propose any congestion control, error control or flow control mechanism and therefore let the application implement these functionalities.

These controls can be implemented at the application with the help of RTCP information. Indeed, RTCP provides periodic information, such as the number of sent packets, the number of lost packets or the jitter. This information allows the multimedia application to adapt the stream to the network state. This adaptation can be done through a change of codec, or a degradation of the quality in the codec. Furthermore, this information can be given to a congestion control mechanism, such as TFRC, in order to compute the sending rate as proposed in [GG05].

3.2.3 SIP

SIP [RSC+02] was designed as a signalling protocol for voice over IP application. It allows two end-users to establish a communication in the network with a negotiation of the codecs to use which is usually done with the SDP descriptor [HJ98]. This protocol is responsible for the establishment and ending of the communication. During the establishment session, SIP facilitates the discovery of the IP address of both end-users due to the use of email-like identifiers or telephone numbers. This identification method allows the user to be addressable wherever he is located. SIP can use any transport protocol but it usually choose the UDP protocol, since it avoids the connection establishment and teardown overhead.

3.2.4 Summary

We have described in this section the solutions deployed at the application level for the description and deployment of QoS at the application level. These solutions have demonstrated their benefit but suffered from one main drawback; the use of UDP transport protocol. Indeed, the use of UDP is justified by the fact that this protocol does not need to establish the connection, which can be done via SIP. Nevertheless, this protocol is often filtered at the edge router of the network and requires the application to implement the congestion control at the application level, which overloads the multimedia application.

3.3 NETWORK SERVICES AND ARCHITECTURES

3.3.1 Previous Contributions

The initial and current Internet architecture is based on the paradigm of "Best Effort". In this architecture, the network does not provide any guarantees of reliability, time or bandwidth. In order to provide guarantees to the end user, several network architectures have been proposed during the last fifteen years. One of the first ones standardized by the IETF was the Integrated Services (IntServ) architecture [BCS94]. In this proposal, guarantees are provided per-flow and a reservation path has to be setup. These guarantees include bandwidth and delay in a first set of services (Guaranteed Services) or reliability and delay in another set of services (Controlled Load Service). Since this architecture provides QoS per flow and needs a reservation process, poor scalability is one of the main reasons that prevented its deployment in the core network.

Differentiated Services (DiffServ) [BBC+98] was the second network QoS architecture standardized by the IETF. In order to solve the problem of scalability introduced by IntServ, DiffServ provides QoS to a class based flow aggregate. In this proposal two kinds of services can subscribed to by the user. The first one is included in the Expected Forwarding (EF) class of service. The second class of service is called Assured Forwarding (AF). In EF class, a temporal service is provided and a complementary AF class provides a bandwidth guarantee to the aggregate. These two kinds of services are setup by distinguishing the two parts of the networks, the edge and the core of the network. The edges routers are in charge of the traffic conditioning. This conditioning is done through the use of the DS field in the IP packet, according to the SLA of the user. Then, the core routers forward the packets according to the this DS field following the Per Hop Behaviour associated to the class of service.

More recently the Multi-Protocol Label Switching (MPLS) architecture [RVC01] has been standardised at the IETF. This architecture has been proposed in order to simplify network routing. Indeed, MPLS enables the setting of VPN tunnels which allows a scalable solution with the help of routing protocols that restrict the topology information known to an incoming packet from a VPN site. It also offers Class of Service (CoS). Like the DiffServ architecture, MPLS uses bits in Layer 3 header to specify a Class of Service (CoS) which also can be mapped in the MPLS Layer 2 label. Thanks to this CoS information, MPLS can provide differentiated services.

Based on the previously introduced network architecture, new propositions have emerged in order to provide some quality of service. In this context, the newly introduced EuQoS project, presented in the following section, provides class based QoS. This QoS is set-up through the virtualisation of the resources and the services and a per domain management. In this architecture, the inter-domain management is done via the introduction of new end to end signalling protocols.

3.3.2 The EuQoS system

Much research has been carried out on Quality of Service mechanisms for packet-switched networks over the past twenty years. The results of these efforts have still not lead to the deployment of multi-domain networks providing QoS guarantees [CMX+05]. The EuQoS project [MBYSG+07] is an integrated project under the European Union's Framework Program 6 which aims at deploying a flexible and secure QoS assurance system over a pan-European testbed environment. The EuQoS System aims to deliver QoS to many applications requiring QoS guarantees such as voice over IP, video on demand or medical applications over multi-domain heterogeneous environments such as WiFi, UMTS, xDSL or Ethernet technologies.

For this purpose, the EuQoS System integrates various architectural components such as signalling protocols, traffic engineering mechanisms, QoS routing and admission control to resource reservation scheme and also tackles the issue of QoS aware transport protocols. In this context, network configuration (i.e. resource allocation and reservation) is done according to the user's SLA and applications' requirements. This configuration consists mainly of the establishment of a QoS-path between the two end systems through different service providers. In order to successfully establish this path, the EuQoS uses the following key components:

1. the Resource Managers (RM), in charge of managing the QoS inside each domain;
2. the Resource Allocators (RA), in charge of applying the decision of the RMs;
3. the Enhanced QoS Border Gate Protocol (EQ-BGP), an evolution of BGP-4 that can be used for interdomain routing;
4. the Enhanced QoS Next Steps in Signaling (EQ-NSIS), an extension of NSIS;
5. the Enhanced QoS Common Open Policy Service (EQ-COPS), in charge of the signaling between RAs and RM.

As a result, in this proposal the Quality of Service is achieved through the implementation of five Classes of Services (CoS):

1. IP telephony;
2. Real-time interactive;
3. Multimedia streaming;
4. High-throughput data;
5. Best Effort.

Several prototypes of this system have been deployed over testbed composed of GEANT (the European research network) and the National Research and Education Networks (NRENs) of the partners involved.

In this architecture, the QoS is provided directly to the application. Therefore, in this thesis we take advantage of this guarantee to propose a novel congestion control that is able to provide the negotiated guarantee at the application level.

3.3.3 Summary of Network Layer Contributions

We have presented, in this section, the currently standardised QoS network architectures and their limitations. We have shown that these present contributions suffer from some scalability problems in the case of IntServ and inter-domain deployment for DiffServ and MPLS. The solution proposed by the EuQoS system combines the efficiency of the MPLS and DiffServ classed based QoS with the definition of new signalling protocols and per domain management in order to solve the inter domain problems. In this architecture, the network informs directly the application about the negotiated QoS without giving any information to the transport layer. In this thesis, we propose to enable the transport layer to make use of this information and propose a specialisation of the transport layer in order to complete the protocol stack.

3.4 TRENDS IN TRANSPORT PROTOCOLS

3.4.1 Introduction

In recent years, new transport protocols have been proposed. Two main protocols have emerged from a myriad of proposals to the IETF, namely the Datagram Congestion Control Protocol (DCCP) [KHF06] and the Stream Control Transmission Protocol (SCTP) [SA00]. These two transport protocols differ from the widely used TCP protocol on two main characteristics. The main difference consists in the fact that these protocols are datagram-oriented instead of byte-stream oriented like TCP. This difference is justified by the fact that IP is datagram-oriented and the new direction for application definition as described in the Application Level Framing (ALF) architecture [CT90]. The second difference is in the introduction of new congestion control mechanism. Recent work on transport protocols [HFPW03, KHF06, FHPW00, WBB04] has proposed alternatives to the generally used window-based congestion control. These protocols compute a sending rate which reproduces the TCP behaviour. These proposals have been defined as an alternative to UDP in order to carry multimedia traffic while respecting the fair-share principle cited in [Jac88].

Furthermore, previous studies [SM05, KK05] have demonstrated the poor performances of TCP over wireless and multi-hop networks while others emphasise the good behaviour of rate controlled congestion control over these networks [CNV04, AP03]. Following these studies, the logical step is to consider reliable rate-based protocols in order to provide a fully reliable service for multi-hop networks such as vehicular networks (VANET) as emphasised in [LSF+06].

In this context, the IETF is pushing for the creation of a new congestion control working group [Egg07, FA07]. The outcome of this working group should follow one of the definitions of "TCP-friendliness", even if this principle is subject to criticism nowadays [Bri06].

However, in this thesis we will follow the definition in RFC3448 saying: *"[...] a flow is 'reasonably fair' if its sending rate is generally within a factor of two of the sending rate of a TCP flow under the same conditions"*. It is generally agreed that this definition concerns instantaneous values, on average equivalent rates should be achieved.

In this section, we fristly present the usual standardised transport protocols, UDP, TCP, SCTP and DCCP. We then present the Fully Programmable Transport Protocol (FPTP), a transport protocol framework that allows an adaptation between application QoS needs and the transport layer. We finally present the future direction in the transport layer and give a summary of this section.

3.4.2 UDP

UDP can be seen as a minimalist transport protocol as it provides to the application only a multiplexing over the network protocol. Nevertheless, since it does not provide any reliability and congestion control it does not introduce any jitter at the application layer but at a counterpart the in the case of a lossy channel the multimedia application is degradated. Thanks to its lightness, UDP is more and more used with in particular the RTP protocol as described in the section 3.2.2. Nevertheless, since it does not implement any congestion control this protocol is usually shaped or even filtered out at the edge of the network.

3.4.3 TCP and its Evolutions

TCP is actually the most used transport protocol in the Internet. Nevertheless while the generic name has remained the same, the current TCP version used is different from the first TCP version in the early eighties by Postel [Pos81]. In this section, we describe four main TCP variants and the reasons that have led to the definition of these different versions: TCP Tahoe [JB88], TCP Reno [JBB92], TCP Vegas [BP95], TCP New Reno [FHG04] with SACK [FMMP00]. Finally, we present last version of TCP algorithms able to better perform over high throughput networks and wireless networks.

TCP Tahoe

One of the first TCP version has been described by Van Jacobson in [JB88]. In this version, the congestion control mechanism is based on the estimation of losses by the sender and a congestion window regulates the number of packet that can be sent over the network (i.e. emitted rate). The increase of this congestion window follows two different stages: the slow-start and the congestion avoidance phase. In the slow-start phase, the congestion window grows exponentially until a certain threshold. Once the protocol has reached this value, it follows a congestion avoidance phase where it only increases the congestion window by a value of one more segment.

This version of TCP suffered from numerous drawbacks. The first concerns the error recovery mechanism used which was the Go-back-N mechanism. This mechanism is not efficient mainly because it can only retransmit packets that have already been received. The second drawback of this TCP version concerns the method for the detection of losses

and the recovery of losses. Indeed, the detection of losses in this version is done using a timer that is triggered for every packet and stayed active either until the reception of the corresponding acknowledgement packet or by the use of the fast retransmit algorithm. Furthermore, when the loss is finally detected, the protocol goes back to the slow start phase with a threshold value of the half of the actual congestion window.

In order, to solve these problems new mechanisms have been proposed, the selective repeat mechanism for the problem of the error recovery mechanism and fast recovery added to the fast retransmit mechanism for the problem of the loss detection.

TCP Reno

TCP Reno version took into account the drawbacks previously introduced of TCP Tahoe in order to improve the protocol. Indeed, it modifies the fast retransmit algorithm that could have been implemented in the Tahoe version to integrate the fast recovery algorithm. The fast recovery algorithm consists of halving the congestion window, instead of going back to the slow start algorithm. This congestion window is increased during this period by the number of duplicate ACKs. Furthermore, this version applies the selective repeat mechanism for the recovery of packets lost. Nevertheless, these mechanisms also contain some minor drawbacks: the successive fast retransmit problem or the false fast retransmit followed by a false recovery problem. This version also suffers from performance problems when multiple packets are dropped in the same sending window.

TCP Vegas

TCP Vegas has been introduced in [BP95] and proposes new algorithms for the slow-start phase, the estimation of the available bandwidth in the congestion avoidance phase and the loss detection compare to TCP Reno. In order to detect congestion in the network, TCP Vegas defines a $BaseRTT$ as the minimum measured RTT and the $ExpectedRate$ as the ratio of the ratio of the congestion window to the $BasedRTT$. Furthermore, the sender measures the $ActualRate$ based on the sample RTT, then if the difference between the $ExpectedRate$ and the $ActualRate$ is superior to an upper bound the sender linearly decreases the congestion window during the next RTT. Otherwise if this difference is lower than a lower bound the sender linearly increases the congestion window. According to [BP95], this TCP version achieves a better rate than the Tahoe and Reno TCP version. Nevertheless, this version was never deployed due to scalability and stability concerns identified later in [AHA97, JPBF94]. One of the main drawback of TCP Vegas concerns its poor performances when mixed with other TCP versions.

TCP New Reno

In order to improve the behaviour of TCP Reno when multiple packets are lost in the same window, TCP New Reno has been proposed. In this version, a modified version of the fast recovery algorithm has been integrated, where partial ACKs are used to indicate multiple losses in the same window. This new fast recovery algorithm has been described in [FHG04].

This last version of TCP appears to be adequate to wired networks where the bandwidth-delay product is not too high, but performs poorly over high bandwidth-delay product and wireless networks. These problems over wireless networks are due to the interpretation of a lost packet. Indeed, the TCP loss detection mechanism supposes that packets are lost because of a congestion in the network. However, in a wireless access network context, these losses can be due to the channel errors or bad transmission. This fact has motivated new TCP enhancements that are presented in the following section.

TCP variants for High Throughput and Wireless Networks

Nowadays, networks offer more and more bandwidth capacities to the end user. Nevertheless, TCP alone cannot take advantage of these new services because of the AIMD algorithm which makes TCP too slow to adapt its sending rate to the bottleneck of the network. In order to solve this problem, two kinds of transport protocols based on a variation of the TCP AIMD congestion avoidance phase algorithm have been proposed. The first proposals remain close to the TCP architecture as they do not require routers to modify their internal algorithms and marking such as in BIC, HSTCP or STCP protocols [XHR04, Flo03, Kel03]. The second proposal, such as XCP [KHR02] and VCP [XSSK05], performs better than both TCP New Reno and TCP Westwood but requires the network to provide information about the actual network congestion level thanks to an ECN-like mechanism [RFB01].

Nowadays, another TCP problem concerns its poor performances over wireless networks. Indeed, TCP has been designed to perform over wired networks where a packet lost means network congestion. As a result, the congestion control mechanism, as described above with the fast recovery and fast retransmit mechanisms, decreases the congestion window in order to decrease the congestion in the network. Nevertheless, in the case of wireless communications, losses can also be due to urban obstacles, mobility of devices or channel interferences. In this context, new versions of TCP congestion control have been proposed such as WTCP [SNV+02] or TCP Westwood [GSW+01]. Another solution, TCP Veno (Vegas + Reno), proposed to let TCP Vegas estimate the available bandwidth and to act as TCP Reno when a loss is detected due to a congestion [FL03]. Nowadays, the current TCP used in the Internet are TCP New Reno (*BSD), TCP Westwood (Microsoft Windows Vista), and a TCP BIC variant (GNU/Linux) all with SACK enabled by default.

3.4.4 SCTP

SCTP stands for Stream Control Transmission Protocol [SA00]. This protocol provides a reliable message-oriented transport service to the applications. In order to achieve this service, SCTP differs from TCP on several points. The main difference concerns the definition of several streams within a connection. In this definition, a stream is not a reference to a byte stream such as in TCP but as a sequence of messages. Furthermore, SCTP provides a multihoming service to the application which is initialised at the beginning of the connection. This multihoming can be therefore used to provide a different order service on the different streams. This function is useful in the case of prioritized messages.

3.4.5 DCCP

The Datagram Congestion Control Protocol (DCCP) [KHF06] offers an unreliable transport service for message-oriented applications. Furthermore, DCCP provides a congestion control mechanism in order to avoid the Internet collapsing and to share fairly the network bandwidth with TCP flows. This congestion control can be configured actually according to two profiles called Congestion Control ID (CCID) 2 or 3. In CCID 2 [FK06], a congestion control similar to the window based congestion control is provided. In CCID 3 [FKP06], a rate-based congestion control is provided. This congestion control is based on the TCP-Friendly Rate Control [HFPW03], which will be presented in more detail in section 3.5.5.

3.4.6 FPTP framework

In the remaining chapter of this thesis, the implementation of the proposed contributions have been done in a user-level framework [Exp03]. This framework is based on the principle of composition of micro-mechanisms to build a complete transport protocol.

In [Exp03], the author developed some micro-mechanisms in order to provide certain quality of service according to the media profile of the application. This profile is given to the transport protocol through the use of an XML file. In this configuration file different levels of application level QoS can be defined. Based on this file, the transport protocol configures the micro-mechanisms needed to fulfil the QoS requirements. This configuration of micro-mechanisms has been made possible through the definition of an language able to described both multimedia data and the action to apply to every kind of data [EMR+03].

3.4.7 Future Directions and Summary

In addition of the standardised transport protocols, researchers have proposed new architectures and services for transport protocols. One of these new architectures consists of building transport protocol as a combination of micro-mechanisms. This idea has been first introduced in the xkernel proposal [HP91]. Based on this idea numerous frameworks for the definition of new service have been proposed [GG07, EPM04, For07].

In [GG07], the authors proposed a UDP-based transport protocol for high-speed, wide area networks named UDP-based Data Transfer (UDT). UDT is a connection oriented duplex protocol which supports reliable and partially reliable communications. UDT uses a rate congestion control similar to the one used in RAP [RHE99] since it applies an AIMD algorithm to the inter packet time. UDT allows the use of a different congestion controls. In a modular version of UDT the application can choose numerous numbers of TCP algorithms.

In [For07], the author proposed a new abstraction stream-based transport protocol named Structured Stream Transport (SST). In SST, applications can create independent substreams from an original stream. Ordered and reliable services are provided for intra-stream but not inter-stream. Furthermore, SST allows the creation of short lived datagram oriented communication due to the introduction of ephemeral streams. These streams do not

provide any reliability and therefore are more suitable with time-constrained applications. The connection in this transport protocol is handled by three abstraction layers responsible for different jobs inside the protocol, the *stream protocol*, the *channel protocol* and the *negotiation protocol*. The *channel protocol* is a connection-oriented, best effort delivery service that provides packet sequencing, acknowledgment, privacy and congestion control. The *negotiation protocol* is in charge of configuring the channel. Finally, the *stream protocol* is built on top of the two previously introduced protocols and is in charge of the delivery of a reliable stream-oriented service to the application.

New services have been also introduced in recent years. One of these services is the introduction of partial reliability and partial order in order to fill the gap between UDP and TCP services. These services have been successfully implemented in SCTP and FPTP. Other services are based on the cross layering between the transport layer and the application or the network layers.

3.5 CONGESTION CONTROL: STATE OF THE ART

In the section 3.4, we have presented the main transport protocol with a particular focus on the evolution of TCP versions and its congestion control. In this section we will present briefly the New Reno congestion control and use this version as a control congestion control group for rate-based congestion control. In addition to the congestion control mechanisms that have been standardised, numerous rate-based transport protocols have been proposed the last ten years. In this section we give an overview of the common TCP-friendly rate-based congestion control.

3.5.1 Methodology

In the following of the section we will present and evaluate congestion control with a particular focus on the rate based congestion control. In order to perform this evaluation we will try to apply the metrics introduced in [MHT07]:

- *efficiency*: which represents the occupancy of the bottleneck by a transport protocol;
- *fairness*: when sharing a bottleneck, the fairness represents how fair this share is accomplished. It is usually quantified by the $min - max$ method [BG92]. If only one bottleneck is present in the network, we can use Jain's fairness criteria [Jai91] in order to measure this characteristic;
- *convergence speed*: this metric represents the time passed before reaching the equilibrium;
- *smoothness*: this metric represents the oscillation in the the throughput during the equilibrium state;
- *responsiveness*: this metric represents the convergence time for one flow while the convergence speed is applied to the whole system;

- *TCP-friendliness* (or inter protocol fairness): the TCP-friendliness represents the fairness between different protocol while the fairness is applied only to the same kind of protocol.

In the following of this section, we present three of the main rate-based congestion control mechanisms and quantify them in regards to the previously introduced metrics.

3.5.2 TCP New Reno congestion control mechanism

As described in section 3.4.3, the TCP congestion control has evolved along the different versions standardised at the IETF. We present in this section the most commonly used TCP version; TCP new Reno. In this version the congestion control is composed by two main phase; the *slow start* and the *congestion avoidance* phase.

The slow start phase aims to avoid congestion at the beginning of the connection while efficiently increasing the size of the congestion window. Indeed, at the beginning of the transmission, the TCP sender's congestion window has a size of 1 or 2 Minimum Segment Size (MSS). Then, the size of the congestion window is increase by one MSS for every correct acknowledged packet, which results in an exponential increase of the congestion window every RTT. This phase can stop according to two criteria, either there is a detection of losses or the congestion window reaches a threshold value (*SSThresh*).

In the case of reaching the threshold value for the congestion window, TCP enters into the congestion avoidance phase. During this phase it increases the congestion window by one MSS every RTT until the detection of a loss. This algorithm is called AIMD. This loss can be identified according to two mechanisms; the trigger of a timer or the reception of three duplicate ACKs for the same packet. This last method is called *fast retransmit*. In the case of the timer, TCP goes back to the slow start algorithm. Otherwise it enters into the fast recovery algorithm where it continues the AIMD algorithm starting with a congestion window divided by two, compared to the value before the detection of the loss.

The fast recovery and fast retransmit algorithms of TCP New Reno differ from the algorithms in TCP Reno by few points [FHG04] and have been introduced in the section 3.4.3

3.5.3 Model of TCP congestion avoidance phase

One of the first rate-based congestion controls proposed a model of TCP congestion avoidance phase [MSMO97]. This model predicts the steady state of TCP in the scenario of a light to moderate loss ratio. This model is based on several assumptions. The first assumption is that TCP avoids retransmission timeout. Secondly, this model also assumes that both the receiver and sender have sufficient receiver windows space. Furthermore, in this proposal the authors supposed that loss events are periodic. Based on this last assumption they apply a derivation to the stationary distribution of congestion window of an ideal TCP connection. This model results in modelling the TCP windows as described by the equation (3.1).

$$Bw = \frac{MSS}{RTT}\frac{C}{\sqrt{p}} \tag{3.1}$$

Where C is a constant function of the acknowledgement policy and is usually equal to $\sqrt{\frac{3}{2}}$. RTT is the round trip time of the connection, MSS is the maximum segment size of TCP and p is the random packet loss at constant probability.

Evaluation

This model suffers from its numerous assumptions on both the TCP behaviour and the periodic probability loss and therefore it is not able to correctly reproduce TCP throughput. Other models have been later proposed in order to enhance this TCP throughput model. An enhanced model has been proposed in [] in order to take into account the window size and the RTO value. The resulting equation can be summurased as follows:

$$F = (p, RTT, RTO, Wmax, b) \tag{3.2}$$

Nevertheless, this model shows to be accurate in the case of randomised loss in the bottleneck which cannot be applied in every network scenario.

3.5.4 RAP

In [RHE99], the authors have designed a congestion control mechanism able to provide a TCP-friendly congestion control suitable for media streaming.

This protocol is based on three main functions, the *decision function*, the *increase/decrease function* and the *decision frequency*. Unlike TCP, RAP is not Ack-clocked, meaning if it does not receive ACKs for different packet it does not stop emitting. Thus RAP still uses a timeout to detect losses in the case of non reception of ACKs.

The decision function is based on the detection of losses. If no losses have been detected, the protocol increases its rate periodically, otherwise it decreases its rate immediately. The frequency between every rate adjustment is not linked to the reception of feedback from the receiver as in TFRC. Indeed in RAP, the receiver sends an acknowledgement for every received packet.

The increase/decrease algorithm is based on an AIMD algorithm. Nevertheless, since this protocol is not window-based the AIMD algorithm is applied to inter-packet-gap(IPG). This increase mechanism follows formula below:

$$\begin{cases} S_i & = & \frac{PacketSize}{IPG_i} \\ IPG_{i+1} & = & \frac{IPG_i * C}{IPG_i + C} \\ \alpha & = & S_{i+1} - S_i = \frac{PacketSize}{C} \end{cases}$$

where S_i and α denote the transmission rate and step height and C is a constant. The decrease process follows the next process expressed as:

$$S_{i+1} = \beta S_i,\ IPG_{i+1} = \beta IPG_i,\ \beta = 0.5$$

The AIMD process as described above is performed at a frequency of a least once per RTT.

Evaluation

In [HK00], the authors proposed a simulation-based performance comparison of RAP. In their experiments, they mixed different versions of TCP with RAP and other rate-based congestion control mechanisms. They showed that RAP is not TCP-friendly when mixed with TCP Reno and that TCP obtains less than 20% of the bottleneck (instead of 50%) when the researchers configured the router to apply a simple FIFO policy.

In [RHE99], the authors evaluated their proposal according to different scenarios. They show that RAP is not TCP-friendly with every TCP version. They also show that RAP fairly share the bandwidth when it is mixed only with other RAP instantiations. They finally showed that RAP provides a smoother and more predictable throughput than TCP.

3.5.5 A model for unicast data transfer

In [PFTK98], the authors proposed another model for the estimation of TCP throughput. This model is equation-based and the equation in use is the following (3.3):

$$X = \frac{s}{(RTT \cdot \sqrt{\frac{p \cdot 2}{3}} + RTO \cdot \sqrt{\frac{p \cdot 27}{8}} \cdot p \cdot (1 + 32 \cdot p^2))} \tag{3.3}$$

where s is the packet mean size of the communication, RTT is the round trip time of the connection, p is the packet loss rate of the network path and the RTO is the TCP retransmit timeout.

Compared to the model in [MSMO97], this model does not suppose that the loss events occurring in the network are periodic. This model has allowed the introduction of a complete congestion control mechanism named TFRC [FHPW00, HFPW03]. This mechanism uses equation (3.3) and defines a more complete protocol for the beginning of the connection and the role of the both sender and receiver side. Indeed at the beginning of the connection the mechanism uses a slow start phase. This phase, as the TCP slow start, increases exponentially the emission rate at the reception of every feedback packet according to equation (3.4):

$$X = 2 * X_{prev} \tag{3.4}$$

Where X_{prev} is the previously computed rate. This slow start phase stops when the sender received a non-nil estimation of the packet loss. This estimation is done at the receiver either by an inversion of the TCP throughput equation (3.3) when the first loss occurs or by a weighting moving average of the loss event interval (i.e. interval between two loss events[3]). This phase can start again in the transmission if the RTO timer is triggered.

The sender is responsible for the computation of the RTT and the estimation of the RTO. This component receives the information of the packet loss rate from the receiver through feedback packets that are supposed to be sent at least once per RTT. Then, as described above, the sender either applies the slow start equation (3.4) or the TCP throughput

[3] a loss event is defined as one or more packet lost during a period of one RTT

equation (3.3) and minimises the output by comparing it to twice the receiving rate which has been given in the feedback message.

Evaluation

The introduction of so-called TCP-friendly congestion control in the Internet has raised many questions about the performance of equation based congestion control and the definition of TCP-friendliness.

The performance of equation-based congestion control has been studied following both empirical and analytical approaches. In [Wid00], the author shows, by measurement, that long RTT causes TFRC mechanism to have a very long transitional state before reaching the actual bottleneck rate of the network even if it is the only one competing. Therefore, compared to TCP, the TFRC mechanism will have a slower convergence speed and responsiveness if the RTT is high, but a similar one in the case of low RTT. Furthermore, in [PFTK98], it is shown that TFRC mechanism obtains a smoother throughput than TCP in the same network conditions.

Recently, researchers explained the behaviour of equation based congestion control by analysing the mathematical properties of the equation and its different parameters [VB05, RX05, XH06]. In [VB05], the authors studied how the supposed TCP-friendliness of equation-based congestion control is influenced by mathematical factors. In order to fulfil this study the authors accepted the definition of TCP-friendliness as an axiom that requires the non-TCP source to obtain a long-run term average sending rate not larger than the one that TCP would have obtained under the same circumstances. As a result, they identified four sub-conditions whose conjunction implies TCP-friendliness. The first is called *conservativeness*, which means the source reaches a rate not larger than the TCP throughput formula used by this protocol. The second is that the loss event rate experienced by the source is not smaller than the one TCP would have experienced. The third is that the RTT observed by the source is not smaller than the one TCP would have experienced. The fourth is that TCP obtains a throughput at least as large as the TCP throughput formula. In this study, the authors analysed three equation-based congestion controls that follow a basic control for the computation of the packet loss rate with a constant RTT. Based on these assumptions, enumerated numerous propositions and theorems. They then verified these propositions and theorems through simulation and experiments.

In [RX05], the authors follow the notations and assumptions introduced in [VB05] to underline the limitation of equation-based congestion controls and of TFRC in particular. The authors examined how three of the main factors of TFRC, namely the rate equation, the loss event rate and the RTO estimation, could influence the long-run throughput difference between TFRC and TCP. They outlined that two flows competing for the same bottleneck will see a different loss event rate if their sending rate is significantly different. They also outlined that this difference in loss event rate can amplify the difference in the sending rate. They attributed these differences to the convexity of the throughput equation (as in [VB05]) and the difference in the estimation of the RTO between TFRC and TCP. This study may be limited by the same assumptions as [VB05].

3.5.6 Conclusion of the congestion control mechanism

In this section we have described four congestion control mechanisms, TCP congestion control, and three of the main rate-based congestion control mechanisms. We have seen that even if it is not fully TCP friendly, TFRC remains the closer congestion control mechanism when mixed with TCP. Furthermore, TFRC provides the smoothest rate compared to both TCP and RAP.

3.6 CONCLUSION OF THE CHAPTER

In this chapter we have presented the context of this thesis with a particular focus on the issues faced by congestion control mechanisms. We have, first briefly presented the possible network architectures and services with a focus on the EuQoS system which is the framework of the first contribution of this thesis. We then presented the trends and possibilities for the design of the transport protocols. Finally we gave a review of the current options for the integration of rate based congestion control inside these new transport protocols. This last review also introduced some explanations for the differences between these congestion controls and standard window-based congestion control.

Chapter 4

Design and implementation of a QoS-aware transport protocol

4.1 INTRODUCTION

Today, multimedia applications are in widespread use and require strong delay and bandwidth guarantees. The Assured Forwarding (AF) class of the IETF/DiffServ [HBWW99] provides a guaranteed minimal throughput of which these applications can take advantage. The offered service is called Assured Service (AS) and is built on top of the AF Per Hop Behavior (PHB). The minimum assured throughput (also called target rate[1]) is given according to a negotiated profile between the user and the network service provider. This service is particularly designed for elastic flows. These flows are generated by applications able to adapt their network usage to the available network resources (also called adaptive applications). This means that the application is able to increase the traffic to use the available network resources and can decrease it when a congestion occurs.

Most of today's Internet applications are designed to be adaptive and use TCP [Pos81] as a mean to transport their data. TCP offers a reliable and in-sequence, end-to-end stream-oriented data transfer service. Moreover, TCP implements flow and congestion control mechanisms in order to avoid network congestion and the receivers' buffers overflow. Despite a fair TCP behaviour in terms of network resource usage and bandwidth sharing, TCP is not appropriate for many applications that integrate time and bandwidth constraints and do not require full reliability [WKST04].

A classical alternative to the use of TCP is the User Datagram Protocol (UDP). UDP is a minimalist transport protocol which does not provide any packet reliability, order and flow congestion control. As a result, UDP needs the application to implement user-level congestion control in order to compete fairly with other TCP flows. The Datagram Congestion Control Protocol (DCCP) is a recently standardized protocol offering a congestion

[1] We call in this chapter TCP and TFRC target rate, the TCP or TFRC flow target rate

controlled, non-reliable transport service [KHF06]. DCCP is suitable to applications currently using UDP, indeed congestion control is a fundamental mechanism that should not be delegated to application layer because of risk of unfair selfish behaviour. DCCP aims to provide a transport service that combines the efficiency of UDP with TCP congestion control and network friendliness. To realize that, one of the congestion control mechanism implemented in DCCP is the TCP-Friendly Rate Control (TFRC) [HFPW03]. TFRC is a congestion control mechanism for unicast flows operating in a best-effort Internet environment. Based on the TCP throughput equation [PFTK98], it is designed to be fair when competing for bandwidth with TCP flow. It generates a flow with much lower throughput variations over time than TCP. As a result, it is particularly suitable for multimedia applications such as video streaming or telephony over the Internet.

In the particular case of the DiffServ/AF class, a minimal bandwidth is provided (called in-profile traffic part), with the possibility to reach higher bandwidths (called out-profile traffic part) depending on the network congestion level. As stated previously, multimedia applications are natural candidates to use this service class. Nevertheless, as for classical TCP flows over DiffServ/AF class, TFRC does not use the full potential of the offered service and produces unexpected results in terms of user requirements as it will be demonstrated in the following of this chapter. As the TFRC mechanism models the TCP AIMD congestion control algorithm, its behaviour remains similar in average to TCP over this class. The flow RTT drives the obtained long term throughput, the guaranteed bandwidth not being efficiently used by the application in case of long RTT.

In this chapter, we focus on the behaviour of TFRC mechanism in the context of a DiffServ/AF class. Through our implementation we show the good properties of classical TFRC in terms of bandwidth smoothing and sharing when mixed with other TFRC or TCP flows. Nevertheless, even if a throughput guarantee is provided to the application by the underlying network, as for TCP, the throughput obtained by TFRC mainly depends on RTT and loss probability. Thus, the application does not always receive the negotiated guaranteed throughput. To cope with this problem, we propose a simple TFRC specialisation, namely gTFRC, allowing the application to reach its target rate whatever the RTT and the target rate value of the application's flow. We validate, for the first time, the newly introduced QoS-aware congestion by implementing it inside a Java framework. After this validation, we combine this congestion control with a SACK-like reliable mechanism and a flow control in order to provide the first complete reliable transport protocol compliant with a bandwidth guaranteed network service. This protocol is the first version of the Chameleon Protocol (CP) denoted in the rest of this chapter CP_QoS.

This chapter is structured as follows. Section 4.2 provides related work about the DiffServ/AF and congestion control mechanisms. We then present and validate in Section 4.3 our implementation of TFRC mechanism. Based on this implementation we present one case where TFRC is not able to obtain the negotiated bandwidth. In order to solve this problem, Section 4.4 details the problem statement and presents the gTFRC mechanism. Section 4.5 details a real implementation of gTFRC in the Java framework introduced in chapter 3. Based on the implementation of the QoS-aware congestion control, the section 4.6 presents the implementation and evaluation of CP_QoS. Finally section 4.7 gives a perspective of this work and provides a conclusion.

4.2 RELATED WORK

In order to better understand the problem; how TFRC can be specialized to be QoS-aware, in this chapter, as TFRC models the TCP congestion control, we first recall previous studies on TCP over DiffServ/AF, then, we present a section concerning related work for TFRC.

4.2.1 TCP over DiffServ/AF class

Many studies related to the performance of TCP-flow over assured service have already been conducted. In [SNP99], five factors have been studied (RTT, number of flows, target rate, packet size, non responsive flows) and their impact has been evaluated in order to provide a predictable service to TCP flows. In an over-provisioned network, the target rate associated with the in-profile traffic is achieved regardless of these five factors. However, these factors have a deep impact on the distribution of the out-profile excess bandwidth. In their paper [PC04a], Park and Choi demonstrate the unfair allocation of out-profile TCP traffic and conclude that the smaller target rate aggregate (resp. larger target rate) occupies more (resp. less) bandwidth than its fair-share regardless of the subscription level. As the TCP protocol uses the AIMD congestion control algorithm which fairly shares the bandwidth available, the only mean to obtain a service differentiation with the TCP protocol is to use DiffServ traffic conditioners such as token bucket color marker (TCM) [HG99] or time sliding window color marker (TSWCM) [FSa00]. The behaviour of these traffic conditioners has a great impact on the service level, in terms of bandwidth obtained by TCP flows. Several others conditioners have been proposed to improve throughput insurance [EGS02], [FRK00], [HBF02], [KAJ01], [LAF05a], [LAF05b], [NPE00]. These contributions clearly showed that the key parameters to the TCP throughput guarantee problem are given by $loss\_probability$, the RTT and the $target\_rate$) associated to each flow.

4.2.2 TFRC over DiffServ AF class

As previously introduced, TFRC is an equation-based, rate control mechanism aiming at reproducing the behaviour of TCP congestion control with the use of the equation (3.3).

There are few known studies about TFRC behaviour over a DiffServ networks. In particular, the authors of [KK03] investigate AF-TFRC performances and give a service provisioning mechanism allowing an Internet Services Provider (ISP) to build a feasible DiffServ system. In that study, the problem of large RTT difference between long and short transfers were not addressed. Moreover, for experimental purposes (based on loss rate estimation) all simulations were carried out during 1000 seconds. This duration associated to invariant network conditions allows a TFRC flow to converge easily to its target rate. As a result, the flows achieve an average throughput near the target rate.

In the following section, we will present and validate our implementation of TFRC in user-space and we present a limitation of TFRC over DiffServ/AF.

4.3 VALIDATION OF THE TFRC IMPLEMENTATION

In this section we present the implementation and the performance analysis of our implementation of TFRC congestion control mechanism. This implementation has been done in user-space by using the Fully Programmable Transport Protocol (FPTP) framework [Exp03]. FPTP has been introduced to offer a generic transport service and a dynamically configurable transport protocol. FPTP is a connection-oriented and message-oriented transport protocol. FPTP offers, among other things, a partially ordered, partially reliable, congestion-controlled and time-controlled end-to-end communication service. FPTP has been designed to be statically or dynamically configured according to the application layer QoS requirements. FPTP services are implemented by the composition of configurable micro-mechanisms suited to control and manage the QoS required by sessions' flows.

We have implemented the TFRC mechanism as a processing module in this compositional architecture (see chapter 3 section 3.4.6). This TFRC mechanism has been specialised and QoS adapted, as described in the previous section, in order to take into account the QoS delivered by the underlying network. FPTP uses an object-oriented approach to dynamically instantiate micro-mechanisms. The Java language has been used for implementing FPTP, because of its object-oriented properties. The rest of this section focuses on TFRC evaluation and its behaviour above the DiffServ/AF service. We show that the basic TFRC mechanism is not able to efficiently use the underlying level of service.

In this section we present a part of the validation measurements that have been obtained on both the ns-2.28 simulator (named in the following the reference TFRC implementation) and the Chameleon Protocol framework (named in the following CP/TFRC) with an underlying network of which the behavior is emulated and controlled by the Dummynet tool [L. 97]. We performed several tests and in this section give an overview of this validation.

4.3.1 General Assumption and model

In order to validate the TFRC implementation, we used the simple topology given in Figure 4.1 for the ns-2 simulator and the real testbed. Over this topology we demonstrate that our implementation react in the same order of magnitude than the ns-2 TFRC implementation when, first, there is no other flow in competition for the space in the router's buffer. Then we introduce random losses during a given period in order to analyse the reactiveness of our implementation. Afterward, we introduce a UDP flow in the network to better quantify this reactiveness. Finally we show in a simple experiment a limitation of TFRC congestion control mechanism over a DiffServ/AF network.

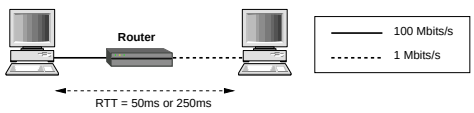


Figure 4.1 The simulation topology for TFRC validation

The real testbed is composed of two end-stations on GNU/Linux, and one router with FreeBSD. We use a Dummynet pipe in order to emulate various RTTs and packet-loss rates (PLR). For ns-2 simulation and real emulation: the packet size was fixed to 1000 bytes; the router queue size was 50 packets; measurements were carried on during $180sec$. For each experiment, we computed the average throughput at the server and at the receiver side.

4.3.2 Network with constant bandwidth

In the scenario presented in Figure 4.2, we show that in a network without any loss and with a constant bandwidth, CP/TFRC implementation using the framework described in chapter 3, section 3.4.6, acts like the reference implementation. Figure 4.2 (a) shows the reference implementation results and Figure 4.2 (b) the CP/TFRC results. In this scenario, the bandwidth is fixed to $1000Kbits/s$ and $RTT = 50ms$. No loss is introduced in the network.

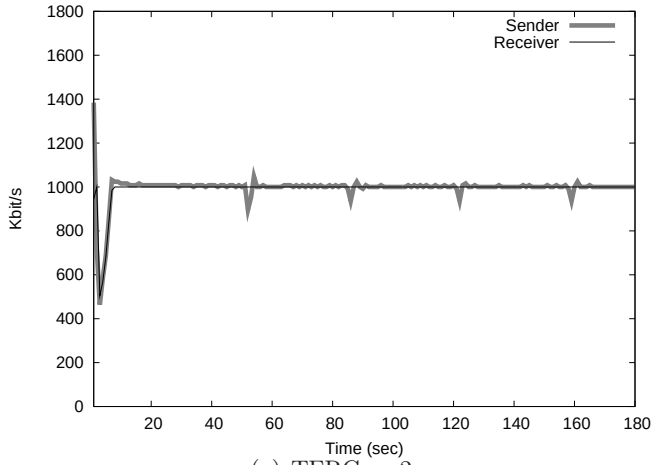
These figures show that at the receiver side, the measured throughputs are identical on both figures. We can note that the throughput oscillations on the sender side are more important on 4.2 (b) than on 4.2 (a). This slight difference can be explained by the different environments (i.e., simulation and real systems) and particularly in the real implementation host processing and the CPU load influences the packet treatment and as a result end-to-end delay variance is higher for the real implementation. Nevertheless, the CP/TFRC behavior remains strongly similar to ns-2 and above all, on the receiver side, the same throughput is obtained.

4.3.3 Impact of losses and end-to-end delay

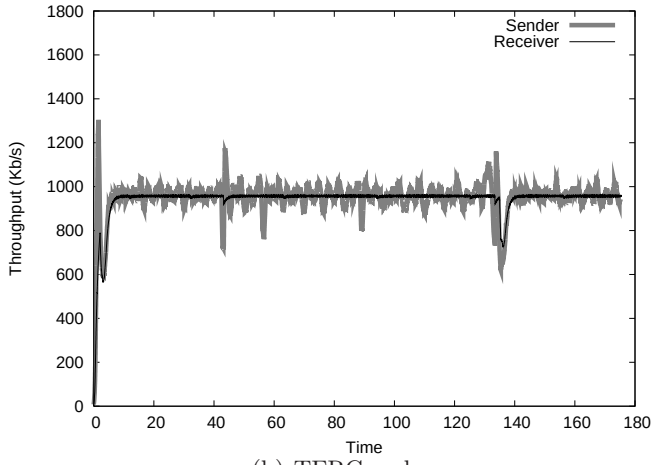
The aim of this experiment was to show that in the case of high RTT ($250ms$) and a loss rate of 1% between $t = 60s$ and $t = 120s$, the CP/TFRC implementation reacts in a similar way to the reference implementation and that the convergence toward the available rate is identical after a loss period. In Figure 4.3, we show that CP/TFRC implementation responds properly to loss detection, during the specific packet loss rate period. The readjustment to a normal sending rate is done in roughly the same amount of time (nearly $25sec$ in this particular RTT case).

4.3.4 Impact of an UDP flow

In this experiment, the bottleneck remained unchanged. There were no losses and the RTT is set to $100ms$. A UDP flow with a rate equal to $500Kbits/s$ was sent between $t = [30sec, 90sec]$. In Figure 4.4, due to the packet multiplexing with non-responsive UDP flow, both implementations significantly decreased during the UDP transmission. Furthermore, CP/TFRC implementation responded to the detection of losses due to the UDP flow in the same way as the reference implementation. When the UDP flow stops, the response of both implementations remains similar.



(a) TFRC ns-2



(b) TFRC real

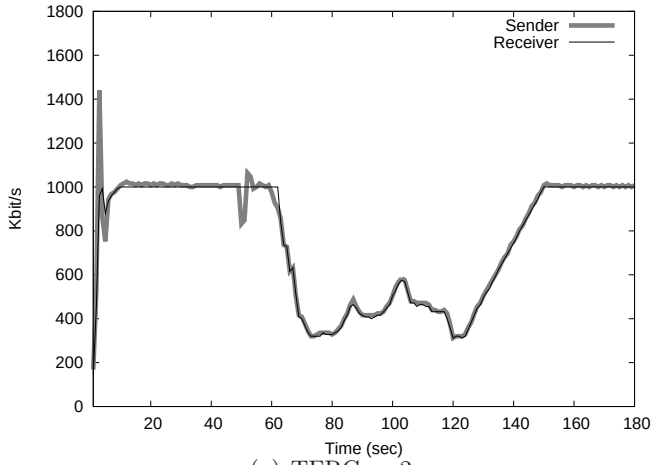
Figure 4.2 BW=1000Kbits/s RTT=50ms PLR=0

4.3.5 TFRC limitations over a DiffServ/AF network

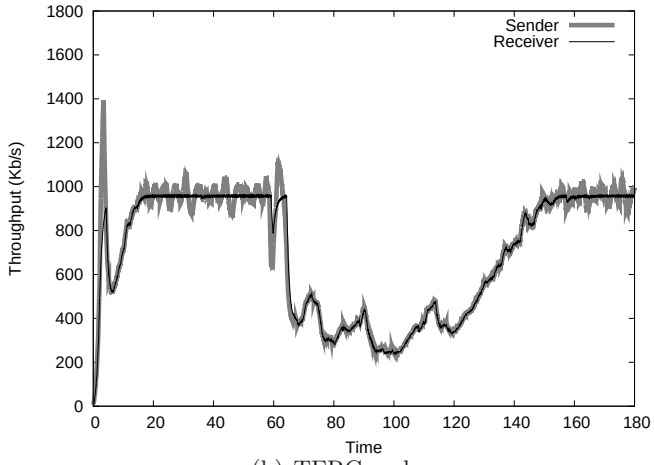
In this section we give an example of the TFRC limitations over a DiffServ/AF network. We will show in more details these limitation in the section 4.5.

In order to illustrate these limitation we will use the DiffServ network show in Figure 4.5. The hosts were PCs on GNU/Linux and routers run FreeBSD with ALTQ [Cho99] in order to implement the DiffServ network. The experiments were carried out using the following configuration: the packet size was fixed to 1500 bytes; a two colors token bucket marker with a bucket size of 10^4 bytes was used on the edge router; the routers were configured with queue size of 50 packets and RIO parameters in the core router correspond to (min_{out}, max_{out} ,p_{out} ,min_{in}, max_{in}, p_{in})=$(10, 20, 0.1, 20, 40, 0.02)$; the bottleneck between the core and the egress router was $1000Kbits/s$; measurements were carried out for $180sec$.

In this illustration, we configured the network to be over-provisioned by 20%.



(a) TFRC ns-2



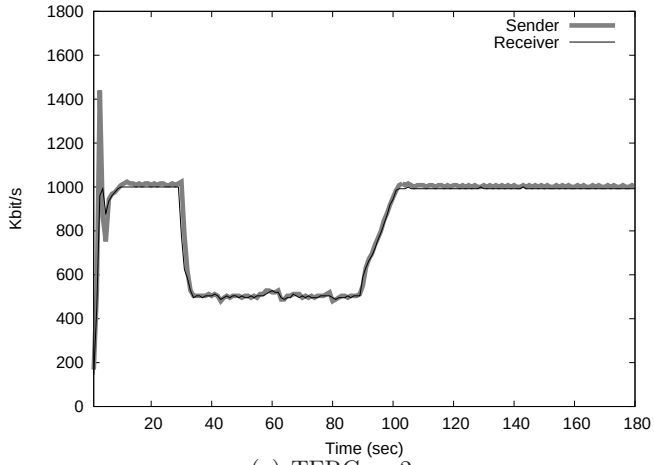
(b) TFRC real

Figure 4.3 BW=1000Kbits/s RTT=250ms PLR=1%

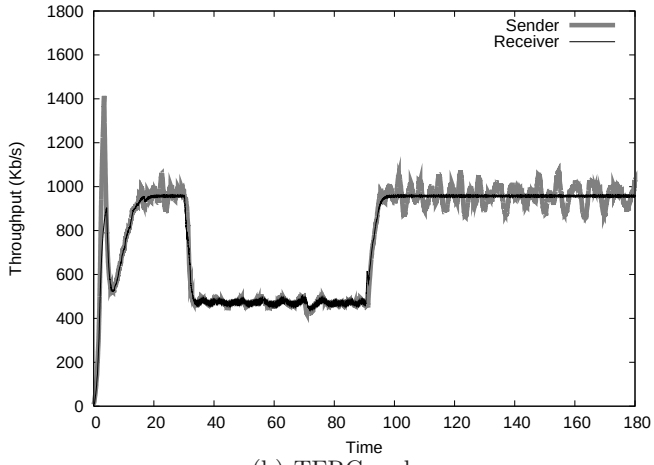
We present in this scenario a network configuration where one TFRC flow cannot reach its target. The network is configured with an excess of 20% of bandwidth. The two flows in this network have respectively a target rate of $600kbit/s$ and $200kbit/s$ and an RTT of $300ms$ and $10ms$. As already identified for TCP in [SNP99], in such a scenario, the flow with the higher RTT and target rate is not able to obtain its negotiated bandwidth. In a similar manner, as show in Figure 4.6, TFRC alone is not able to reach its target rate. We will give in section 4.5 more evidences about the non-obtention for TFRC's flow of negotiated QoS. In order to solve this problem we present in the next section a solution that allows TFRC to obtain its target rate regardless of the network conditions.

4.3.6 Conclusions

We performed experiments with several other scenarios similar to those defined in [Wid00] for the TFRC ns-2 validation. Even if this is not the purpose of this chapter, all these



(a) TFRC ns-2



(b) TFRC real

Figure 4.4 PLR=0 BW=1000Kbits/s UDP flow 500Kbits/s $t = 30sec, t = 90sec$

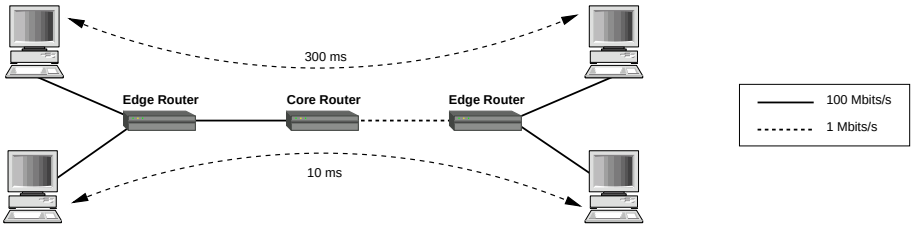


Figure 4.5 The testbed topology for DiffServ experiments

experiments delivered very similar results and allowed us to conclude that our CP/TFRC implementation compares well to the ns-2 implementation.

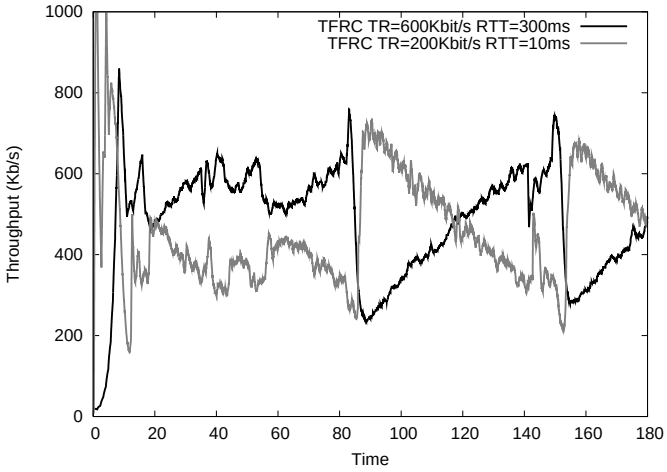


Figure 4.6 TFRC over an 20% over-provisioned network

Furthermore we showed that TFRC mechanism cannot take the full advantage of the underlying bandwidth guarantee in every network conditions. In the section 4.5, we extends the analysis of CP/TFRC over DiffServ/AF networks and we show that the solution introduced in the next section allow the transport protocol to obtain the negotiated bandwidth.

4.4 *G*TFRC: A QOS-AWARE RATE CONTROL

We showed in the previous section, that in some scenarios TFRC mechanism cannot obtain the negotiated bandwidth. In the present section we explain why neither TCP or TFRC could not obtain its target rate in all network condition. Then we present gTFRC , a QoS-aware congestion control which specialises TFRC for bandwidth guaranteed network such as DiffServ/AF. We conclude this section with a discussion about some potential security concerns the introduction of such a mechanism could introduce in the network.

In the context of DiffServ/AF class providing a known guaranteed rate, a flow throughput breaks up into two parts:

1. a fixed part that corresponds to a minimum assured throughput. In the event of congestion in the network, the packets of this part are marked as inadequate to loss (colored green or marked in-profile);

2. an elastic part which corresponds to an opportunist flow of packets (colored red or marked out-profile). No delivery guarantee is delivered to these packets. They are conveyed by the network on "best-effort" (BE) principle and are dropped first when congestion occurs.

This study assumes that the network is well-provisioned and that on the whole, in-profile traffic does not exceed the resource allocated to the AF class. In case of excess bandwidth in the network, the application can send more than its target rate, so the network should

mark its excess traffic out-profile. If the network becomes congested, this out-profile traffic is predisposed to losses.

As noted in section 4.2, the only way to make use of this service differentiation with TCP protocol is to set a DiffServ traffic conditioner. Even if the knowledge of the guaranteed bandwidth could be provided to the transport level, the AIMD congestion control integrated into TCP is not able to use the instantaneous throughput as an input value for its congestion control. Only acknowledgements and timeout analyses indirectly allow TCP to act on the rate control. On the contrary, the TFRC mechanism effectively processes the instantaneous throughput according to the flow's RTT and loss event rate.

Nevertheless, the optimal rate estimated by TFRC can still be under the target rate needed by the application and provided by the underlying DiffServ network. TCP would react in a similar manner by halving its congestion window. As for TCP in the AF class, the TFRC flow is not aware that a loss is corresponding to an out-profile packet and that it should not decrease its emitted throughput below the target rate. For TCP, the solution was to design a new conditioner, able to better mark the TCP flows than a simple token bucket or propose to add a new QoS congestion window as in [FKSS98] or [SPF04].

In contrast to TCP, the use of the TFRC equation makes the mechanism able to directly process the equivalent TCP throughput in function of the flow RTT and loss event. As a result, TFRC does not send packets according to a window but according to a rate that is translate into an inter-packet delaying. The gTFRC mechanism therefore conditioned this rate to be compliant with the negotiated bandwidth. In the present study, the gTFRC congestion control mechanism is made aware of the target rate negotiated by the application with the DiffServ network. Thanks to this knowledge, the application's flow is sent in conformance with the negotiated QoS while staying TCP-friendly in its out-profile traffic part. This conformance is achieved by computing the sending rate as the maximum between the TFRC rate estimation and the target rate as given in (4.1).

$$X' = min(\mathbf{max}(\mathbf{X_{calc}}, \mathbf{g}), 2 * X_{recv}) \tag{4.1}$$

Where: X' is the transmit rate in bytes/s; g is the target rate in bytes/s, and X is the transmit rate in bytes/s computed by the TCP throughput algorithm. The rest of the gTFRC mechanism follows entirely the TFRC specification specified in [HFPW03].

gTFRC requires knowledge of the underlying bandwidth guarantee that the DiffServ/AF network service provides to the session. We assume this information is made available to the mechanism at socket-creation time, directly by the application. This guaranteed bandwidth resulted from a contract between the network service provider and the user. So, the target rate parameter can be set for example by the `setsockopt()` function. Without loss of generality, this parameter is supposed to be known by an application after it has been previously negotiated on an end-to-end basis. This can be accomplished through a proper signalling protocol that a DiffServ architecture should provide [HKLdB05]. The main concern with this approach relates to security. Indeed, if we give the possibility to the application to instantiate through a `setsockopt()` function the target rate negotiated, we can imagine that a misbehaving person could abuse this functionality by giving an higher value to g. We discuss this problem in the rest of this section.

4.4.1 Discussions about the security in gTFRC mechanisms

In our proposal, the application has to provide the target rate negotiated with the QoS network to the transport protocol. This is done at the socket level through a `setsockopt()` specification that allows the application to abuse the network by giving an higher value than guaranteed. Another problem linked to this proposal is the case where the QoS service provider gives an incorrect configuration to the application. In the following sections, we address both problems.

Preserving the provider interest against a denial of service

As we give the possibility to instantiate through a `setsockopt()` function the target rate negotiated between the network service provider and the user, we can easily envisage that a misbehaving user could take part of feature by giving to g a higher value than the negotiated one.

In the context of a DiffServ/AF class, the edge router marks in-profile the packets according to the negotiated profile and out-profile the excess part. A misbehaving client will increase his out-profile traffic part and when a network congestion occurs and the dropping precedence set in the core router will entail the dropping of this excess traffic. Therefore, the misbehaving application will increase its own packet loss rate and will not get any bandwidth advantage. In summary, increasing the value of g at the user level does not impact on the in-profile traffic that is bounded by the SLA between the service user and the provider. Therefore, this kind of denial of service is avoided by the DiffServ conditioning mechanisms.

Preserving the network service user and provider against wrong network configurations

This second case can potentially induce issues both for the network service user and provider. Indeed, in this case a discrepancy between the user and provider configurations either would induce a risk for the service user to get a poorer service than the negotiated one or for the service provider a risk to dedicate to the service user more resources than needed. For instance, such an inconsistency could occur if the service provider miscalculates the resource needed for the related service layer agreement. In a DiffServ context, the in-profile traffic is not guaranteed anymore when a QSTP flows gets losses while emitting below its target rate. In such a case, two actions are possible for the sender. The first one is to continue to emit at the guaranteed rate, g. This is a legitimate behavior because of the service provider must provide to his client the service he has paid for. The second type of action would be to react to the observed congestion and to warn the application or the user that the SLA has been broken off. This can be done thanks to an additional mechanism that would be able to detect that a bunch of losses occurred in the in-profile part. Anyway, in the case of an under-provisioned network, TCP (and TFRC) would react as if the target rate is lower than the expected one. Conversely, gTFRC, does not lower its sending rate and ,in such a pathological situation, suffers from losses. However, by re-

stricting the sender rate to be not higher than twice the receiver rate, the TFRC algorithm bounds this volume of losses.

The TFRC algorithm prevents in a certain manner these problems. Indeed, the algorithm will not return a sending rate higher than twice the receiving rate (given by $2 * X_{recv}$ in equation (4.1)). However, we believe that these security concerns are out of the transport layer scope. We claim that it is definitely not the responsibility of the protocol to detect a selfish user behaviour or to react to a wrong setting. We therefore do not present results concerning an under-provisioning network.

4.5 IMPLEMENTATION AND PERFORMANCE ANALYSIS OF A QOS-AWARE TFRC MECHANISM

In this section we present the implementation and the performance analysis of the previously introduced QoS-aware congestion control. The FPTP framework has been initially modelled and evaluated over a best-effort network [Exp03, EPM04]. However a context such as the previously mentioned EuQoS system allows the transport protocol to be informed of the underlying network's QoS characteristics. In such a context, the network service description can be provided to FPTP through an Extended Application Programming Interface (E-API) for deciding which micro-mechanisms to compose in relation to the associated FPTP session.

The rest of this section focuses on TFRC evaluation and its behaviour above the DiffServ/AF service. We show that the basic TFRC mechanism is not able to efficiently use the underlying level of service and we then propose an extension which improves the QoS delivered to continuous flows.

4.5.1 Testbed measurements in a DiffServ network

This part deals with the use of CP/TFRC implementation and the implementation of the QoS-aware congestion control, gTFRC , as presented in section 4.4, above a DiffServ/AF class.

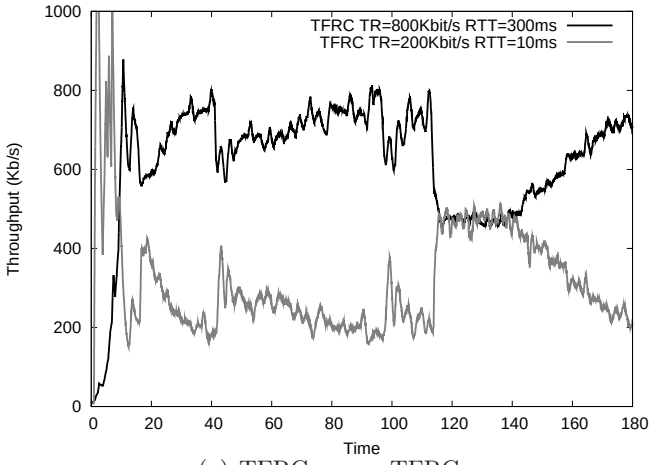
Model and general assumption

CP/gTFRC performances have been evaluated over the DiffServ testbed presented in Figure 4.5. In this topology we use the same parameters as explained in section 4.3.5.

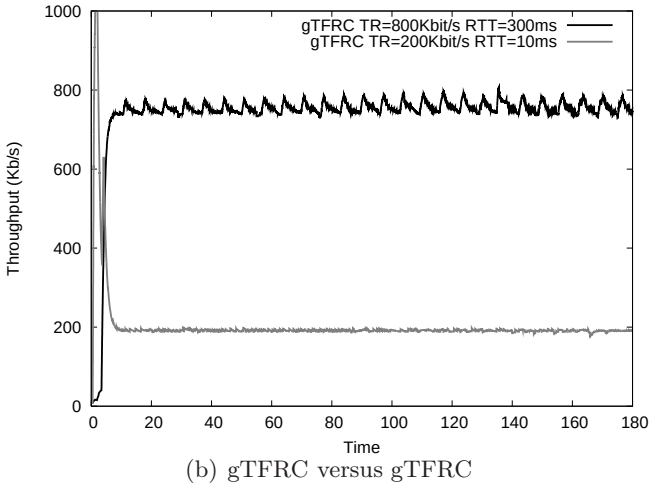
We experimented with many different RTTs and target rate configurations and present in this section a representative measurement of the efficiency of CP/gTFRC . We measured the performance obtained by CP/gTFRC in three scenarios; first we configured the network to be exactly provisioned, then we study the behaviour of gTFRC in the case of over-provisioned network when it is mixed with in a first time TFRC and TCP in a second time.

Exactly-provisioned network

In Figures 4.7, two flows are transmitted over the testbed. The first one has unfavourable conditions since it has the highest target rate to reach and a high RTT ($RTT = 300ms, TR = 800Kbits/s$). The second flow has the lowest target rate ($200Kbits/s$) and a low RTT ($10ms$). The results for CP/TFRC are presented in Figure 4.7 (a) and for CP/gTFRC in Figure 4.7 (b). We can see that CP/gTFRC make it possible to reach the target rate more quickly than TFRC and that CP/gTFRC keeps its target rate. The reason is obvious since at the first rate decrease entailed by the TFRC algorithm, CP/gTFRC evaluates a rate equal to the target rate.



(a) TFRC versus TFRC



(b) gTFRC versus gTFRC

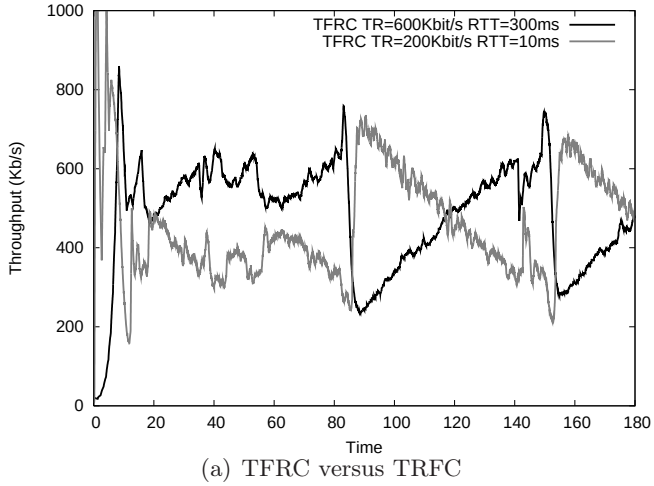
Figure 4.7 Exactly-provisioned network

In Figure 4.7 (a), we can see that the decreasing phase occurs for TFRC around $t = 10sec$ and that CP/gTFRC does not, at this time, deliver a rate lower than the negotiated target rate (Figure 4.7 (b)). Furthermore, we can see that the gTFRC mechanism entails smaller

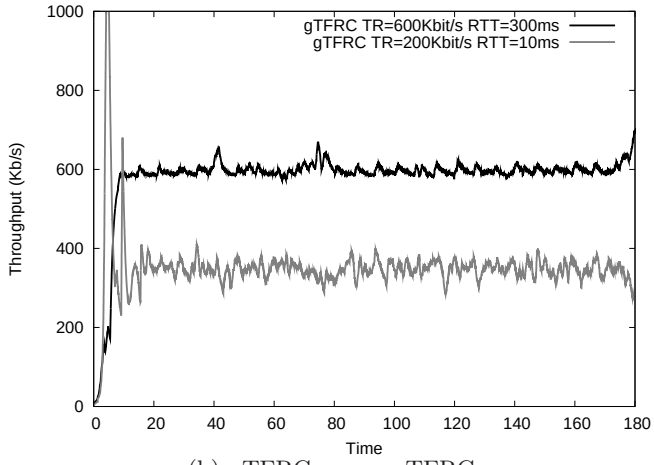
throughput variation than the general TFRC mechanism over DiffServ/AF network. Figure 4.7 (b) shows that the flow with the lower target rate and the lower RTT is constrained to reach its own target rate of $200Kbits/s$.

Over-provisioned network

These experiments dealt with an over-provisioned network in two different situation, where respectively, the sum of the target rates is equal to $800Kbits/s$ (Figure 4.8) and the sum of the target rates is $600Kbits/s$ (Figure 4.9).



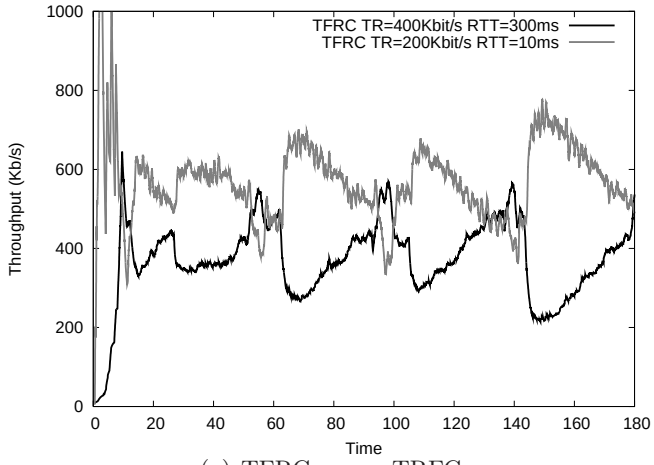
(a) TFRC versus TRFC



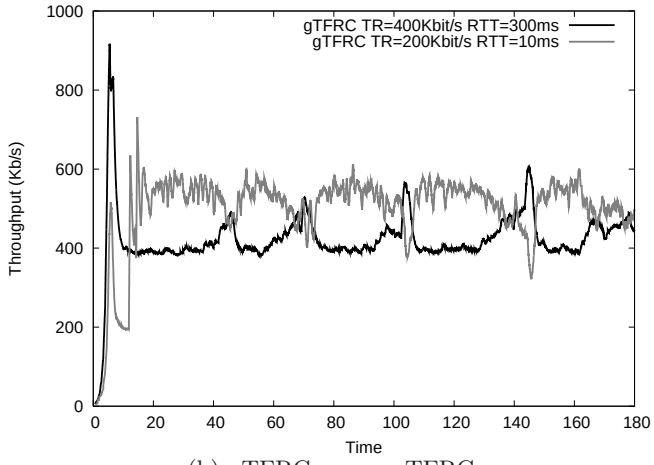
(b) gTFRC versus gTFRC

Figure 4.8 Over-provisioned network 20%

The networks have respectively 20% and 40% of excess bandwidth. Moreover, the more excess bandwidth in the network, the more difficulty the flow with the highest target rate will have to reach its target rate. This is due to the increase of the out-profile traffic which involves more losses in the network. These losses are more critical for the flow with



(a) TFRC versus TRFC



(b) gTFRC versus gTFRC

Figure 4.9 Over-provisioned network 40%

the highest target rate and the highest RTT than for the lowest one. Indeed, the TFRC algorithm can process a rate lower than the negotiated target rate and due to a long RTT, the flow can have difficulty to retrieve its initial throughput as experimented during the period $[80sec, 140sec]$ on Figure 4.8 (a).

This is not the case with the use of CP/gTFRC . We can see in figures 4.8 (b) and 4.9 (b) that the flow with a lower RTT and lower target rate obtains a higher part of excess bandwidth. It is important to take into consideration that the proportional sharing of the excess bandwidth was not the aim of the proposed specialisation of TFRC over an AF network service. This problem should remain under the responsibility of the edge router conditioning.

Interaction with a TCP aggregate in an over-provisioned network

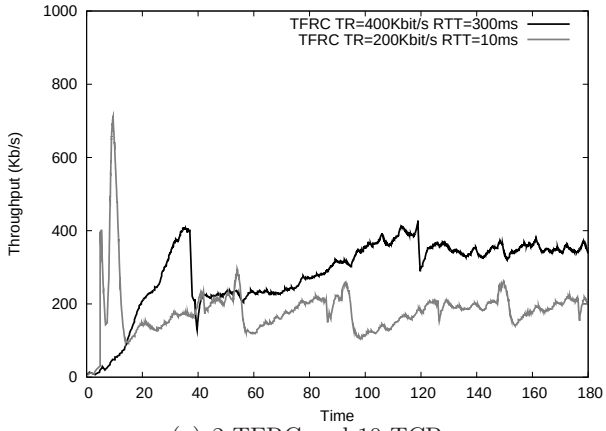
The last experiment shows the interaction of TFRC or gTFRC and a TCP aggregate. In this experiment, two Chameleon Protocol flows with either TFRC or gTFRC mechanisms were sent versus an aggregate of ten TCP flows.The TCP aggregate crosses a token bucket marker with a target rate of $200Kbits/s$ and has an RTT equal to $1ms$. Both CP flows have a target rate of $400Kbits/s$ and $200Kbits/s$ for RTT equal to $300ms$ and $10ms$ respectively. Figures 4.10 give the results obtained for both CP flows.

Concerning the TCP aggregate, the obtained throughput was of $447Kbits/s$, $403Kbits/s$, $362Kbits/s$ respectively for figures 4.10 (a, b, c). Therefore, the TCP aggregate always reached its target rate. In Figure 4.10 (a), conversely we see that with CP/TFRC , both flows had difficulties in reaching their respective target rates and that the flow with the higher target rate and RTT, did not reach a correct throughput value before $t = 120sec$. In Figure 4.10 (b), CP/gTFRC flow easily reached its target rate. Nevertheless, due to the increase of the in-profile traffic and the aggressive nature of the TCP aggregate, the other flow with CP/TFRC strongly decreased its rate. Finally, on Figure 4.10 (c), both flows used CP/gTFRC and reached their target rate while the TCP aggregate remained aggressive and reached its target rate as well.

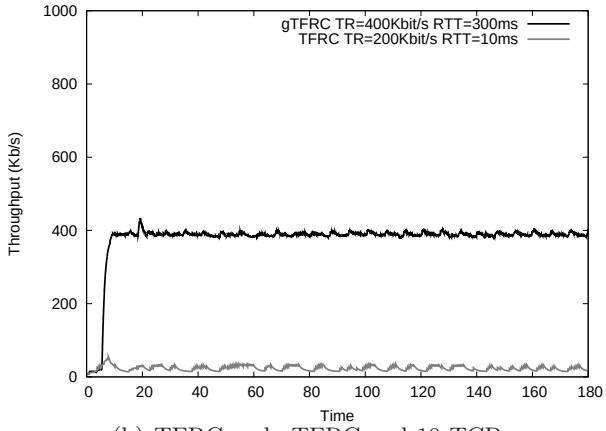
4.5.2 Conclusions

In this section, we have experimentally demonstrated the efficiency of the gTFRC mechanism through a large range of measurements. gTFRC allows to reach a negotiated minimum guaranteed throughput regardless of the RTT or the target rate of a flow. Furthermore, in this section e have only presented the worst cases for a unique flow to reach its target rate. Indeed, as shown in [SNP99], these worst scenario are identified by a large difference in the value of the RTT in addition to a large difference in the value of the target (the smallest RTT corresponding to the smallest target rate value). The second scenario represents the case when a flow is mixed with a TCP aggregate. Thanks to its multiplexing property, the TCP aggregate can outperform the single flow which cannot reach its target rate. gTFRC requires only to be aware of the target rate negotiated by the application. The transport protocol based on such a QoS-aware congestion control mechanism is well suited to multimedia applications which do not require any reliability.

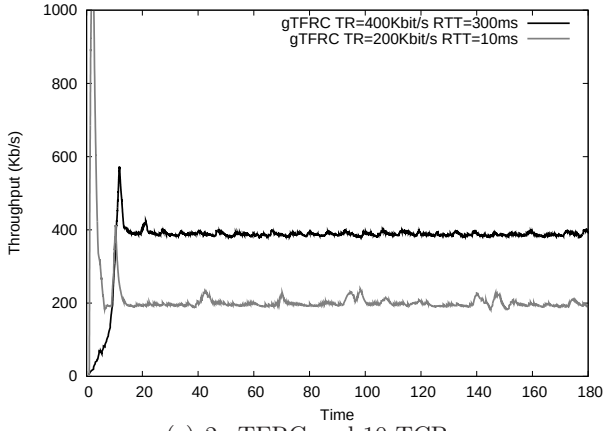
In the following section, we present the composition of this mechanism with a reliability mechanism in order to deliver a reliable and congestion controlled transport service. This service is provided by the combination of a SACK-like mechanism for loss recovery and a flow control mechanism especially designed for rate-based transport protocols. This version of the protocol is well suited for applications which required a reliable service and guaranteed bandwidth. An example of such applications could be a tele-medicine application where doctors want to share high quality images or videos.



(a) 2 TFRC and 10 TCP



(b) TFRC and gTFRC and 10 TCP



(c) 2 gTFRC and 10 TCP

Figure 4.10 Over-provisioned network 20% with ten TCP flows

4.6 DESIGN AND IMPLEMENTATION OF A COMPLETE QOS-AWARE TRANSPORT PROTOCOL

Error and congestion control are two fundamental mechanisms of a QoS oriented transport layer. However, the legacy error and congestion-control mechanisms proposed by TCP focus exclusively on network behaviour and impact badly on application layer bandwidth, delay and jitter constraints. Indeed, when packets are dropped by the network, the standard window-based congestion control mechanism causes TCP to change indirectly its ACK-clocked sending rate. Advances in error control have lead to the SACK mechanism that potentially opens the door to more efficient retransmission strategies during loss periods [FF96]. However, the TCP congestion control mechanism, even when used with SACK, imposes retransmission constraints that entail a severe sender rate decrease during recovery phase [FHG04]. This behaviour may alter the quality of service delivered to continuous flows such as video or audio streams. As underlined by RFC 2018 *"future research into congestion control algorithms may take advantage of the additional information provided by SACK"*.

The present contribution aims at demonstrating how the combined use of TFRC and SACK can improve a TCP-compliant transport service, especially during loss bursts. Indeed, TFRC and SACK share the common goal of improving the QoS delivered to flows by offering both a mechanism for enhancing flows' rate smoothness and a mechanism for loss recovery. Their combined use offers potential performance improvements that this chapter aims to explore. Therefore, we show how two QoS parameters, i.e. bandwidth and reliability, can be managed jointly in a non-conflicting way (i.e. conversely to TCP) to deliver a better transport service than TCP, regardless of the underlying network service. In addition, the composition of SACK and TFRC has two other main advantages: first, SACK allows fully or partially reliable error control disciplines to be achieved; then, the SACK data structure can be easily integrated with TFRC feedback packets.

In the following section, we present the context of this work and we introduce the SACK mechanism and the flow control. Then, section 4.6.1 shows how gTFRC can be composed with the SACK mechanism for delivering an AF compliant and reliable transport service. The resulting protocol appears to be the first reliable transport protocol especially designed for the DiffServ/AF class.

4.6.1 CP/QoS Design and Implementation

In this section we present how reliability can be performed an adaptation of SACK to gTFRC, and due to the design of an efficient flow control adapted to a rate based congestion control.

Reliable TFRC

Section 4.4 focuses on the first component of our QoS-aware reliable transport protocol, that is QoS-aware congestion control mechanism. Indeed, gTFRC allows the target rate negotiated by the application to be ensured whist being TCP-friendly. The next step is

the integration of *g*TFRC with a SACK-based mechanism to provide a reliable transport service. SACK offers a powerful base to provide a sophisticated error-control mechanism compared to the basic *Go back N* error-recovery mechanism. As specified in [MMFR96], the mechanism aims to give information about the set of missing TPDUs[2]. Since TFRC is a datagram-oriented mechanism and SACK is byte-stream oriented, we made change to the SACK mechanism to make it aware of packet losses instead of byte losses.

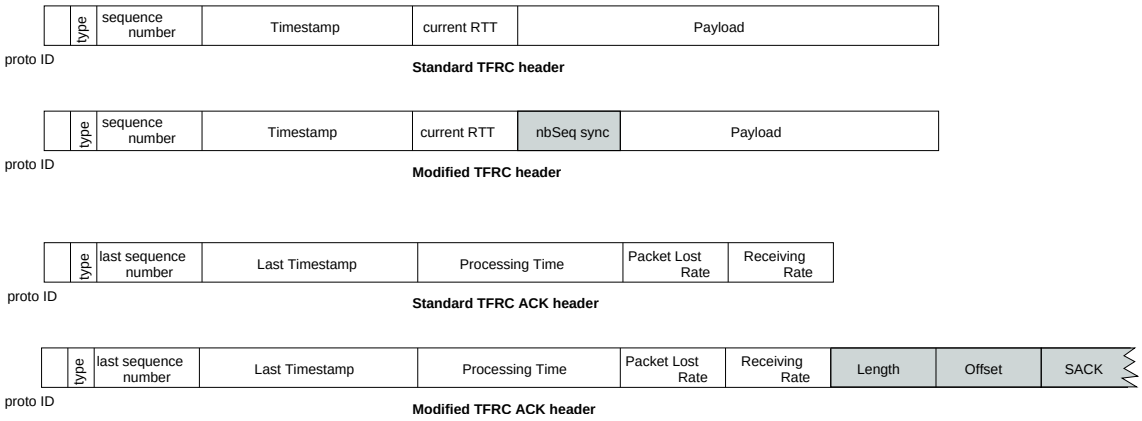


Figure 4.11 Modification in TFRC header

In Figure 4.11, the two first protocol data units represent respectively the TFRC header and the new header that results from the composition of *g*TFRC and SACK. The two last PDU represent respectively the feedback given by the receiver for the classical TFRC protocol and TFRC/SACK composition. In these headers, each field is either 4 or 8 bytes encoded field except for the `proto ID` (one byte), the `type` (one byte) and the `SACK payload` (variable length). The datagram oriented SACK mechanism is defined in the same way as the stream oriented one. The `SACK payload` is constituted by a sequence of pairs of sequence numbers[3]. These pairs represent the edge of intervals of correctly received contiguous packet. The `length` represents the number of pairs to analyse for the sender. Finally the `Offset` represents the sequence number of the first packet of the first pair. We can note that in an implementation these pairs of sequence number could be implemented as a bit field to be more efficient, but we choose to present here a solution close to the one introduced in the SACK RFC [MMFR96]. We can note that the SACK mechanism can help to implement a partial order transport service that would retransmit mandatory packets only.

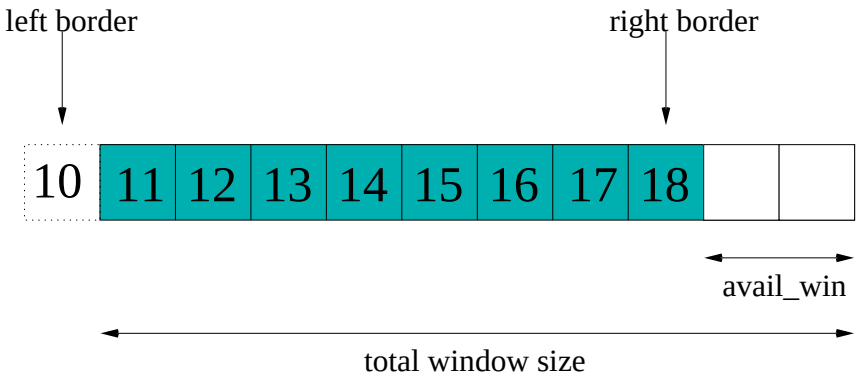
Design of a flow-control adapted to a rate-based reliable Transport Protocol

In this section, we describe the design of our flow-control mechanism. Since the SACK mechanism requires receivers to maintain a buffer for the in-order delivery of packets to

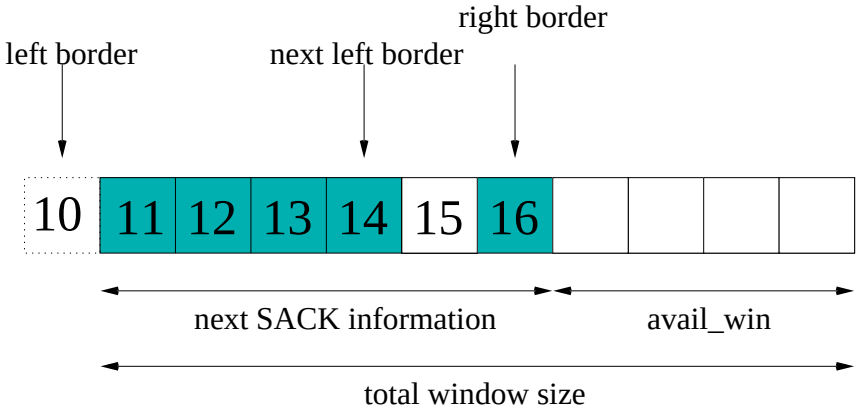
[2]Transport Protocol Data Units
[3]this SACK structure could also be implement as a bit field

the application, for rate control purpose, we introduced a new window variable, *avail_win*, representing the space available in this buffer in the case of constant packet size. This window is similar to the TCP receiver window. The purpose of this variable is to maintain, at the sender, the amount of buffer space available at the receiver and prevent the sender from transmitting more packets than the receiver can buffer. Other candidates solutions for the design, including modification of the TFRC equation, are discussed later. We note in the rest of the section "reading rate" the maximum rate we allowed the application to read from the receiver buffer when there is available data.

Figure 4.12 gives an illustration of sender and receiver windows. In this figure, the dark boxes represent data packets already sent or received.



(a) The sender's window. *Left border:* highest acknowledged packet ID; *Right border:* highest packet ID sent so far. *avail_win:* available window size to send further data packets.



(b) The receiver's window. *Left border:* highest packet ID of the previously sent SACK vector; *Next left border:* highest packet ID of all correctly and in-order received packets. *Right border:* highest packet ID received so far. *avail_win:* available window size to receive further data packets.

Figure 4.12 The sender's and receiver's window

At the sender, the flow-control mechanism should stop transmitting data packets if the receiver's buffer is full. To achieve this, we use the *avail_win* variable, which, at the

receiver, represents the available space in the receiver buffer, in number of packets. This variable is integrated in the TFRC-SACK feedback messages as a one byte field after the `Receiving Rate` field of the last header in Figure 4.11. The *avail_win* variable therefore indicates, at the sender, the supposed number of packets which can be sent. *Avail_win* is never negative and is upper-bounded by the total window size. When this variable is non nil, the sender sends data packets at the rate computed by TFRC algorithm. Each time a packet is sent, *avail_win* is decreased by one at the sender. When *avail_win* is nil, the sender has already sent the maximum number of data packets which could have been accepted by the receiver. Note that the TFRC rate still conditions the speed at which packets are sent, the *avail_win* variable conditions are the maximum number of packets which can be sent between receiving two feedback messages.

Indeed, as mentioned previously, each feedback message sent by the receiver contains the available buffer space. At the sender, upon reception of a feedback message, the local *avail_win* variable is computed by withdrawing the number of packets sent since the header's `Offset` (which can be seen as the *last byte sent* in TCP) from the header's `avail_win` field. A feedback message can therefore unfreeze the sender if the newly computed local *avail_win* variable is non-nil or the SACK vector indicates that some packets need to be retransmitted.

At the receiver side, when a data packet is received, if its sequence number (S_{new}) is higher than the highest previously received sequence number (S_{old}), *avail_win* is reduced by $S_{new} - S_{old}$. Otherwise, this packet is out-of-order and is therefore placed in the reception buffer. When the application reads packets from the buffer, the *avail_win* is increased by the corresponding number of read packets.

Discussion In this section, we discuss the design of our flow control for TFRC and explore alternative solutions.

The main feature of a rate-based congestion control mechanism is the use of an equation to determine the sending rate. This equation typically uses network measurements (or estimations) to calculate the theoretical rate at which TCP would send in similar conditions. Following this observation, we first investigated two other possible solutions to the flow-control problem.

The first solution is to obtain the reading rate of the receiving application and send it back to the sender. This can be done either by estimating the reading behaviour of the application or by assuming that the application can communicate this reading rate to the transport protocol. The sender would then adjust its sending rate to the minimum between its computed congestion control sending rate, twice the receiver's receiving rate, and the application reading rate. However, this solution has two major drawbacks. Firstly, the reading rate depends on different parameters such as application type and CPU usage. and may therefore follow complex patterns, which can be difficult to estimate. This may result in erroneous values leading in buffer overflow. Secondly, in order to provide packet-ordering, the receiver temporarily buffers out-of-order packets. This can lead to a situation where the application reading rate is nil, therefore the sender would stop even if there is space in the buffer.

From this, it follows that there is no improvement or benefit in including the flow control in the TFRC's sending rate algorithm.

Validation of the flow-control mechanism In this section, we validate our flow-control mechanism using simulation in ns-2.30. We first implemented the SACK-like mechanism within ns-2.30's TFRC. We also extended the ns-2 simulator to include the application layer to simulate an application reading from the socket buffer at different rate. Using this implementation, we conduct a set of simulations to demonstrate the effectiveness of our flow control mechanism, and quantify the potential impact of the SACK and flow control modifications over the TFRC flow dynamics.

Validation of TFRC-SACK: TCP-friendliness conservation and Reliability The first experiment aims to verify the TCP-friendliness of TFRC-FC-SACK when sharing a bottleneck with other TCP flows. Currently, the definition of the TCP-friendliness is still being debated [Bri06]. In this study, we will first follow the definition in RFC3448: *"[...] a flow is "reasonably fair" if its sending rate is generally within a factor of two of the sending rate of a TCP flow under the same conditions."*. This definition concerns instantaneous values. Another common view is that, on average, a flow is TCP friendly if *the non-TCP source obtains a long-run term average sending rate not larger than the one TCP would have obtained under the same circumstances* [FF99].

To quantify the TCP-friendliness we therefore use an expression of the means ratio as shown on equation (4.2).

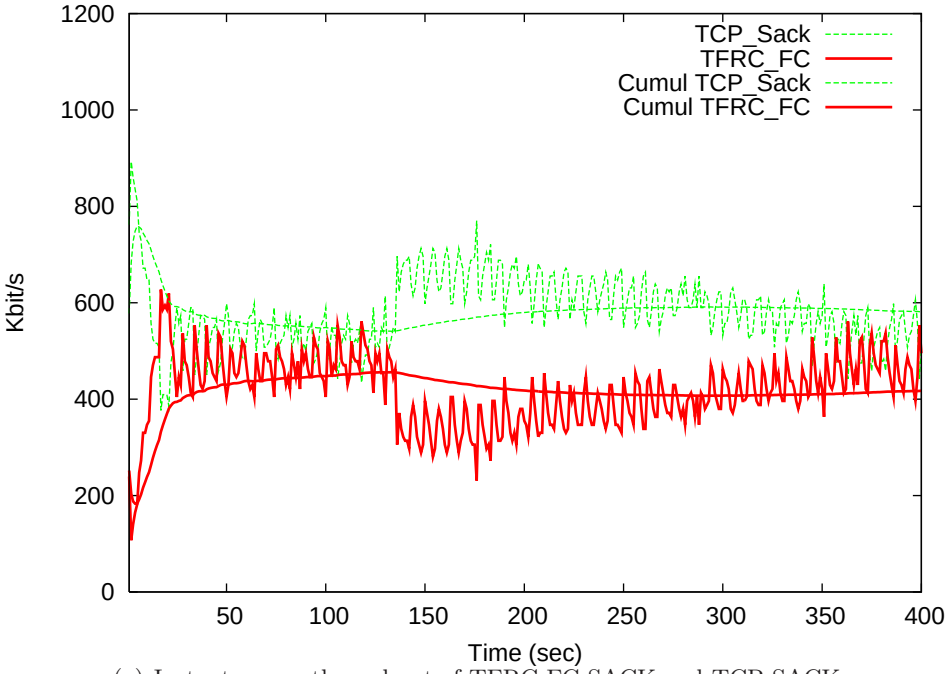
$$T(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}\overline{x_i}}{\frac{1}{m}\sum_{i=1}^{m}\overline{y_i}} \tag{4.2}$$

Where X is the protocol being studied, $\overline{x_i}$ the average throughput of the i^{th} X flow, n the number of X flows, $\overline{y_i}$ the average throughput of the i^{th} TCP flow and m the number of TCP flows. In this formula, if T has a value of less than 1 then the non-TCP flow is TCP-friendly, if T is equal to 1 then we have an ideal friendliness and finally if T if greater than 1 then the non-TCP flow overruns TCP.

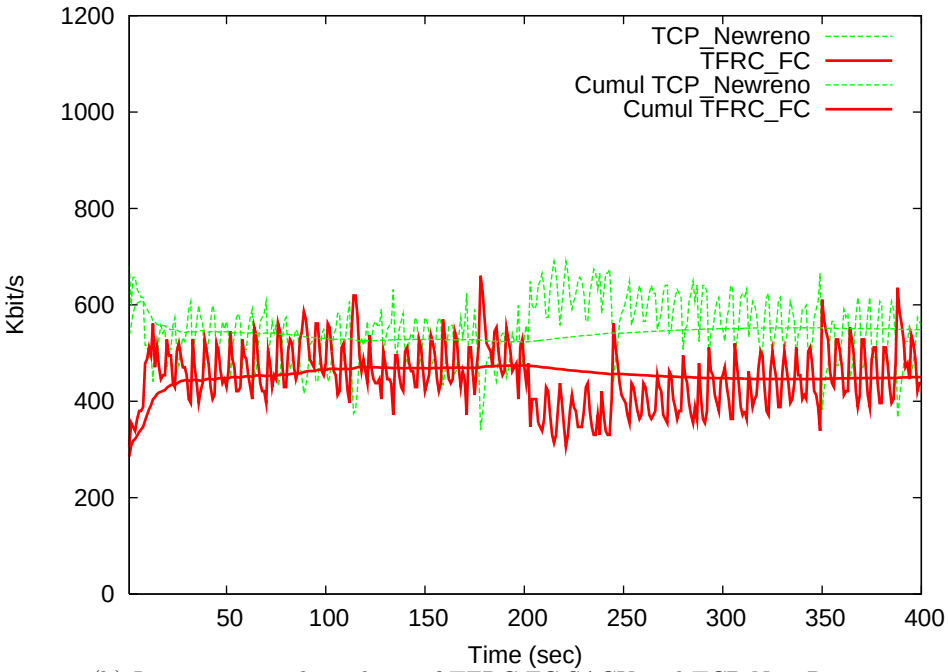
In this simulation scenario, we used a butterfly topology as illustrated in Figure 4.5. We performed two experiments where TFRC-SACK first competed with TCP-SACK, then with TCP New Reno. All three protocols were set to the same packet size of $1KByte$ and a maximum window size $64KBytes$.

Results are presented in Figure 4.13. Each graph shows the flow's instantaneous throughput at the receiver computed with an average sliding window throughput estimation with a $1ms$ window. In both experiments, the application reading rate can be considered as infinite.

From Figure 4.13, we can see that the TFRC-SACK-flow instantaneous throughput is slightly under that of both TCP SACK and TCP New Reno flows. The TFRC-SACK instantaneous throughput is well within the $2x$ factor imposed by our TCP friendly definition. We can therefore conclude that this TFRC-SACK implementation is friendly with



(a) Instantaneous throughput of TFRC-FC-SACK and TCP SACK



(b) Instantaneous throughput of TFRC-FC-SACK and TCP New Reno

Figure 4.13 Validation of TFRC-SACK composition in ns-2.30

both TCP SACK and TCP New Reno[4]. In Figure 4.13 (a), TFRC-SACK equally shares the

[4]we conducted a series of other experiments with different range of RTT and bottleneck bandwidth which confirm this result

bottleneck with TCP for almost $130s$. At $t = 130s$, TFRC suffers from consecutive losses and therefore sharply decreases its throughput. TFRC-SACK then attempts to re-adjust its throughput to the equilibrium with TCP, but this process converges slowly [Wid00]. When competing against TCP New Reno, as shown in Figure 4.13 (b), TFRC-SACK behaves similarly except that TFRC-SACK stays longer at the first equilibrium ($200s$ instead of $130s$). Furthermore, after consecutive losses, TFRC-SACK reaches the equilibrium with TCP New Reno faster than with TCP SACK. These differences can be explained as the TFRC equation models TCP Reno.

Table 4.1 presents the TCP-friendliness index of TFRC-FC-SACK calculated using equation (4.2). As all figures are below one. This confirms that TFRC-SACK is friendly with both versions of TCP.

Table 4.1 TCP-friendliness index results

TCP version	T(TFRC-SACK)
TCP/Newreno	0.82
TCP/SACK	0.72

These experiments made it possible to validate the SACK mechanism and to verify that all lost packets are retransmitted until received. In Table 4.2, we summarize the number of sent and lost packets for each of the flows in the previous experiments. We can see from this table that the TFRC-SACK flow sends fewer packets than both TCP versions. This is explained as the TCP flows' overall throughput is higher than the TFRC-SACK and the packets-statistics are collected during a fixed time period of $400s$. Furthermore, we can see that the TFRC-SACK flows experience less packet losses than both TCP flows (in terms of both absolute value and percentage). This is explained by the fact that the rate-based congestion-control mechanism produces a smoother sending rate compared to a window-based mechanism which is more aggressive. Finally, by using packet marking (not shown in the table), we verified that TFRC-SACK did retransmit until all dropped packets were correctly received.

Table 4.2 Packets statistics

	number of sent packets	number of lost packet (percentage)
TCP/Newreno	26702	166 (0.62%)
TFRC-SACK	21962	45 (0.2%)
TCP/SACK	28740	162 (0.55%)
TFRC-SACK	20368	42 (0.2%)

Impact of the Application's Read Rate The objective of this experiment was to validate the flow-control mechanism, by measuring the sender throughput when varying

the receiver application reading rate, i.e. simulating a slow application. We also wanted to confirm that no packets were lost due to a slow receiver unable to accept incoming packets. In addition, in this section, we quantify the impact of our SACK and Flow Control mechanisms over TFRC smoothness, by measuring the throughput stability during the data transfer.

In order to quantify this stability, we consider the average throughput for each time-unit interval. For each time interval we compute the standard deviation of the throughput for each flow [JWL04] and use the average of variation coefficients of considered flow so considering the following metric equation (4.3).

$$S = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{\overline{x_i}}\sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(x_i(k)-\overline{x_i})^2}\right) \tag{4.3}$$

Where in the following, $\overline{x_i}$ is the average throughput of the i^{th} TFRC-FC-SACK (resp. TFRC) flow, n is the number of flows, $x_i(k)$ is the throughput of the i^{th} TFRC-FC-SACK (resp. TFRC) flow for the k^{th} time interval and m is the number of time intervals.

For these experiments, we used a simple topology where two nodes communicate through a third. Packets were crossing two consecutive links of respectively 10 Mbps and 1Mbps bandwidth, for an overall 20 ms RTT (5ms delay on each link).

Figure 4.14 shows the throughput of a TFRC-FC-SACK flow as the application reading rate is set to $600kbit/s$ at the receiver.

Each packet-loss event is illustrated on Figure 4.14 by a cross on the x-axis. At the beginning of the transmission, the sender sends packets according to the slow start algorithm. This phase stops when the first packet loss event occurs. TFRC then enters the congestion avoidance phase. As soon as the receiver's buffer is full due to the application slow read-rate, the sender can no longer send further packets. As soon as the application reads packets from the buffer non nil *avail_win* values are sent to the sender.

Hence, the sender is only allowed to send new packets when the receiver has delivered some packets to the application. Consequently, Figure 4.14 confirms that the flow control mechanism operates correctly as the throughput is adapted to the receiver application reading-rate. Furthermore, Figure 4.14 shows that the receiver does not drop any packets.

In Figure 4.15, we mix one TFRC-FC-SACK and one TFRC flow in the same network as the one used previously. However, contrary to the previous experiment, the application reading rate varied in time and followed a specific pattern as shown in Figure 4.15. We chose this specific pattern as it represents a mix of reading rate that are respectively above, under and equal to the fair share throughput.

From Figure 4.15, we can first see that a reading rate above to the theoretical fair share value ($500kbit/s$) does not impact on the behaviour of TFRC-FC-SACK: TFRC and TFRC-FC-SACK share equally the link bandwidth. Furthermore, the transition from this reading rate to another one less than $500kbit/s$ does not induce any packet loss at the receiver buffer for TFRC-FC-SACK (but for TFRC). Between $t = 100s$ and $t = 150s$, the application reading rate is set to $100kbit/s$, under the theoretical fair share value. During this phase, we can see from Figure 4.15 that the TFRC-FC-SACK sending rate follows

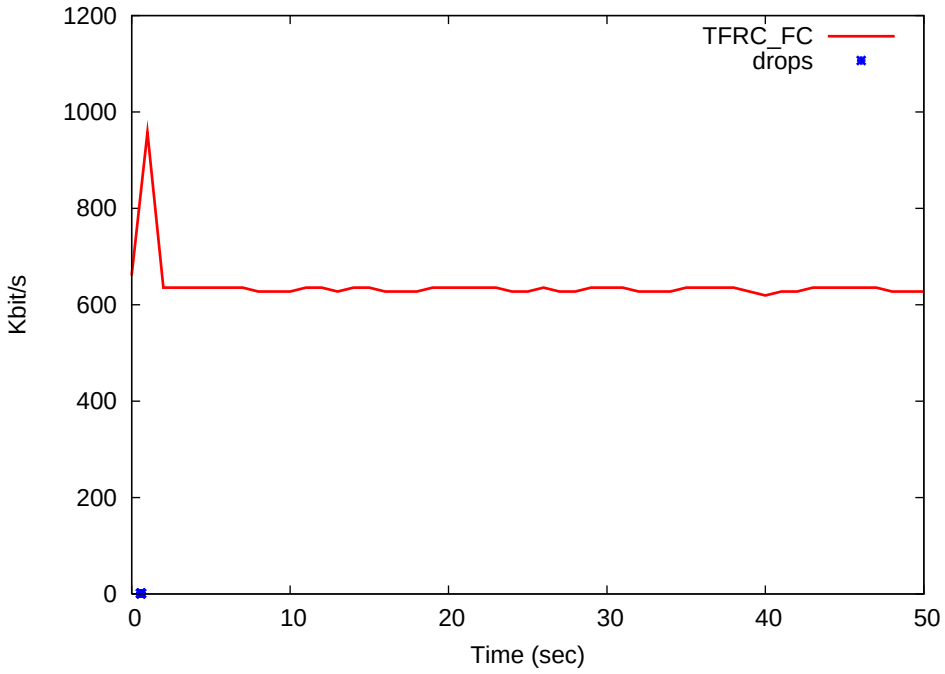


Figure 4.14 TFRC-FC-SACK with read rate 600 Kbps, 20 ms RTT, queue limit of 10 packets (the cross represents six losses)

the application reading rate while TFRC flow can fulfill the rest of the bottleneck. At $t = 150s$, the application reading rate is set again to values above to the fair share for $100s$. We can see from the graph that during this period TFRC-FC-SACK and TFRC equally share the bottleneck bandwidth as expected. Finally, for the remaining variations of application reading rate, TFRC-FC-SACK continues to behave consistently with reading rate variation.

To quantify the impact of the flow control mechanism on the throughput smoothness, we used the stability metric defined in equation 4.3. We applied this metric on a series of experiments that aimed to check that the flow control did not introduce any degradation in the smoothness characteristic of TFRC.

In Table 4.3, we present the results of experiments where two identical flows shared a bottleneck of $1Mbit/s$ during $400s$. We show in this table that TFRC-FC-SACK remains as smooth as TFRC when it is not limited by the application read rate. Furthermore, when we introduce for both flows a receiver reading rate of $300Kbit/s$, the resulting stability of the system is increased. This result can be explained by the fact that the oscillations in the throughput are usually due to the congestion control mechanism that tries to increase until the detection of a loss. In the case of a system limited by the application read rate the two flows do not try to increase nor decrease and therefore are more stable.

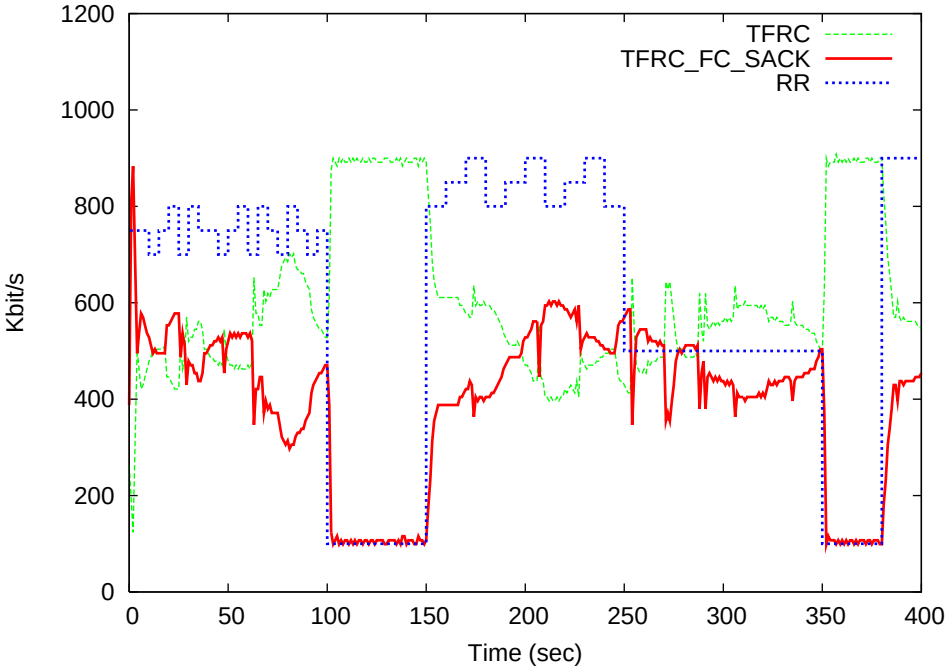


Figure 4.15 TFRC versus TFRC-FC-SACK with experiencing variation of read rate

Table 4.3 Stability index for different protocols

	TFRC	TFRC-FC-SACK	TFRC-FC-SACK reading rate
S	0.094	0.097	0.051

Implementation

In this section we present the implementation of a CP/QoS protocol based on a compositional transport protocol framework [Exp03]. Basically, this framework, developed in Java language, allows easy instantiation of transport layer mechanisms and to compose them to build a transport protocol which applies an efficient adaptation between application needs and underlying network characteristics [Exp03]. Figure 4.16 gives an overview of the micro-protocols (i.e. processing modules) that have been composed for the instantiation of the CP/QoS protocol. CP/QoS is composed on both sides by seven Processing Modules (PM) respectively dedicated to (see Figure 4.16 for details):

- the processing of the outgoing flow (`Add Header`, `Set Header`, `Rate Control`);
- the processing of the incoming flow (`Remove Header`, `Process IN`, `Receive Sock`);
- the `Process Feedback` and the `Create Feedback` deal with the management of the feedback messages (i.e. creation and analysis).

The main components of CP/QoS are:

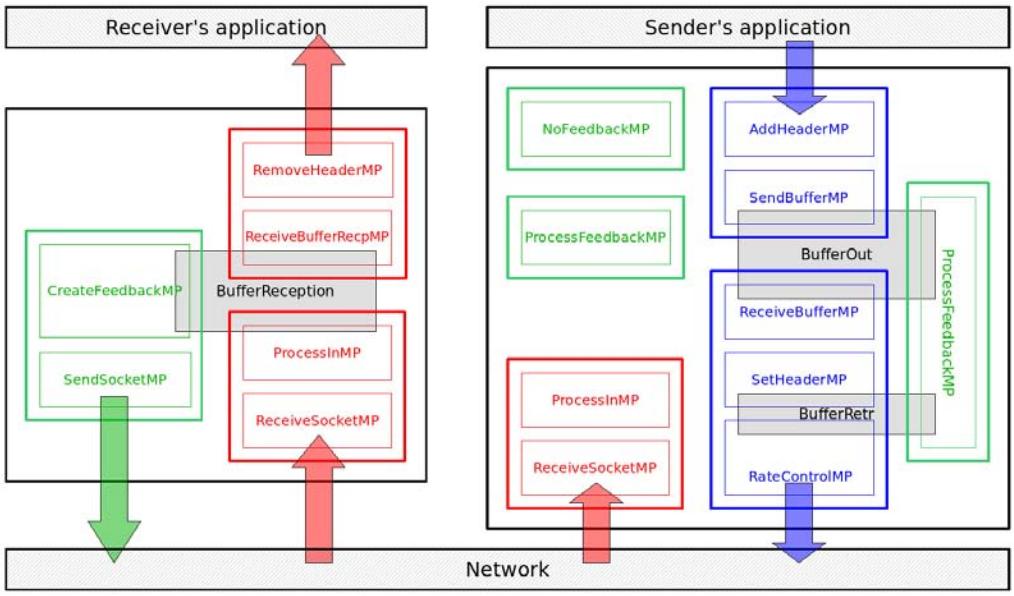


Figure 4.16 Internal mechanisms of the protocol at the sender and receiver side

- the `Process IN` component: this component implements the gTFRC mechanism at the sender side;
- the buffer `BufferOut`: this buffer is the transmission queue upstream from the rate-control component, packets to retransmit are placed on top of this queue;
- the buffer `BufferRetr`: this buffer stores data sent but not yet acknowledged;
- the `Process Feedback` component: this component is in charge of the processing of feedback messages. This component applies error control on packets stored in the `Retransmission Buffer`;
- the `Create Feedback` component: this component computes the loss event rate and creates the Feedback message with the SACK structure and the `avail_win` variable.

Detailed descriptions of this framework can be found in [Exp03].

4.6.2 Performance evaluation of CP/QoS

This section evaluates the CP/QoS service over a bandwidth guaranteed network. We firstly present the experimental model used and the general assumptions. Then, the results and their analyses are provided with respect of various network conditions. For the sake of comparison, the chosen parameters are those used in other well-known papers about TCP over AF, such as [SNP99, CM05, NPE00, EGS02].

Model and general assumption

CP/QoS is implemented in the Java language and evaluated over the DiffServ testbed presented in Figure 4.5. All the nodes are PC, the end-hosts run GNU/Linux and the routers run FreeBSD with ALTQ [Cho99] in order to implement the DiffServ service. The experiments were carried out using the following configuration:

- the packet size is fixed to 1500 bytes;
- a two-color token bucket marker with a bucket size of 10^4 bytes is used on the edge router [HG99];
- routers are configured with a queue size of 50 packets and RIO[5] parameters in the core router correspond to (min_{out}, max_{out} ,p_{out} ,min_{in}, max_{in}, p_{in})=$(10, 20, 0.1, 20, 40, 0.02)$;
- the bottleneck between the core and the egress router has a fixed capacity of $1000Kbits/s$;
- measurements are carried out 10 times during $180sec$ for an FTP-like transfer.

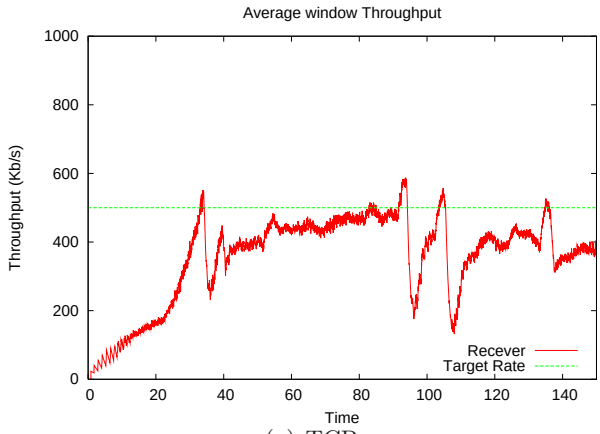
We performed experiments with a large set of different RTTs and target rates. Only a representative part of these results is given in the next section. The choice of these results has been made since the various scenarios presented represent some of the worst cases for a unique flow (TCP and TFRC) to reach its target rate. In the following section we measure first the throughput obtained at the network level at the receiver side. Then we present the "goodput", which is a measure of the throughput at the application level. Finally, we present the jitter obtained for TCP and TFRC flows.

Analysis of CP/QoS behaviour over a standard DiffServ/AF network scenario

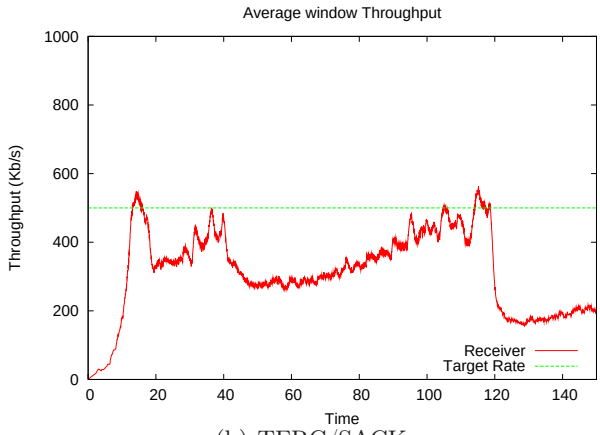
This section aims to illustrate the CP/QoS behaviour above a DiffServ service. The measurements presented in Figure 4.17 gives the corresponding instantaneous throughput at the network level on the receiver side. This throughput is computed using a time-sliding window algorithm of one second as explained in [FSa00].

In this first experiment, we analyzed the behaviour of one flow (i.e. a TCP, TFRC, or CP/QoS flow) versus a TCP aggregate of 15 micro-flows. All the flows have an RTT of $30ms$. This single flow has a target rate of $500Kbits/s$ and crosses the (A, B) path of the DiffServ testbed while the TCP flows aggregate has a target rate of $300Kbit/s$ and crosses the (C, D) path. In all experiments, the TCP aggregate outperformed its target rate. In Figure 4.17, we only report the results for the single flow against the TCP aggregate. First, we give the result obtained by a TCP flow in Figure 4.17(a). As explained in [SNP99], the TCP flow is not in the best conditions to reach its target rate since it has the highest target rate. Moreover, because of the TCP-multiplexing behaviour, when two aggregates with a different numbers of micro-flows are in a network, the larger outperforms the smaller [SNP99]. Figure 4.17(a) shows that the TCP flow does not reach its target rate.

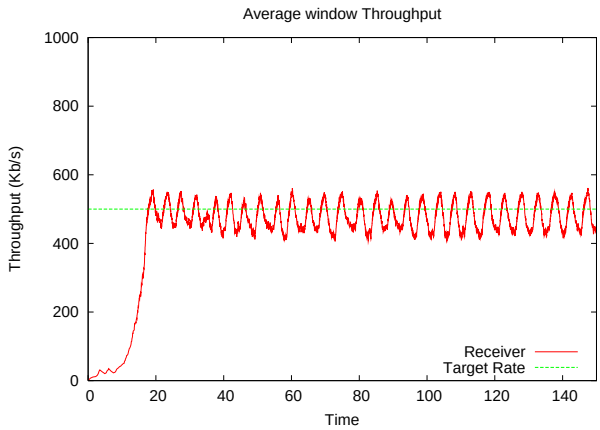
[5] RED In Out queue



(a) TCP



(b) TFRC/SACK



(c) CP/QoS: gTFRC/SACK

Figure 4.17 Throughput of one TCP, TFRC/SACK, CP/QoS flow versus a 15 TCP flows aggregate

In the next Figure 4.17(b), we give the result obtained for a TFRC/SACK flow multiplexed with the same 15 micro-flows aggregate. In this experiment, TFRC/SACK did not reach

its target rate either. Since TFRC reproduces the TCP window-based congestion-control behaviour and since we have added a reliability mechanism, we could expect to obtain a behaviour almost similar to TCP on average. Nevertheless, the smoothing TFRC property makes the TFRC/SACK flow less aggressive than the TCP ones. As the bottleneck of the network becomes loaded, the RTT and the losses in the network increase. As a result, we can see between $t = [40sec, 100sec]$ that TFRC mechanism recovers slowly after a transient congestion [Wid00].

To cope with the QoS-unawareness issue, the CP/QoS protocol composes gTFRC and SACK mechanisms. The results depicted in Figure 4.17(c) illustrates that, conversely to the TCP and TFRC flows, the CP/QoS flow is able to achieve the requested target rate. In conclusion, thanks to the composition of these two mechanisms, CP/QoS can be considered as a DiffServ/AF compliant reliable protocol. Indeed, for these experiments we used only standardized and implemented DiffServ mechanisms such as a token bucket two-color marker on the edge and a RIO queue on the core.

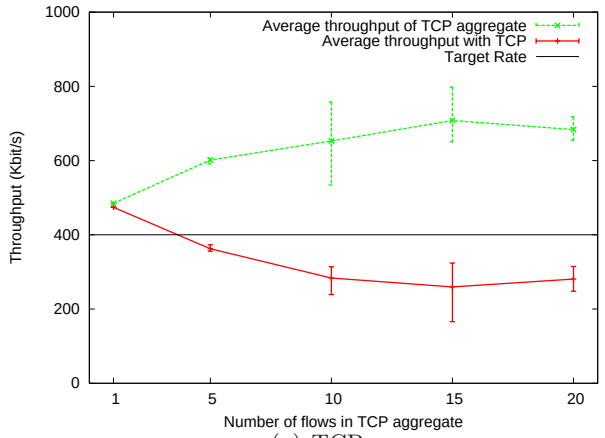
The next section will focus on the study of the impact of these three transport protocols on the QoS offered to the application layer (i.e. the transport service user). In this context, measurements focus on the application throughput (or goodput) at receiver side. In the case of a FTP transfer, this corresponds to the data transfer throughput.

Impact on the QoS perceived at the user level

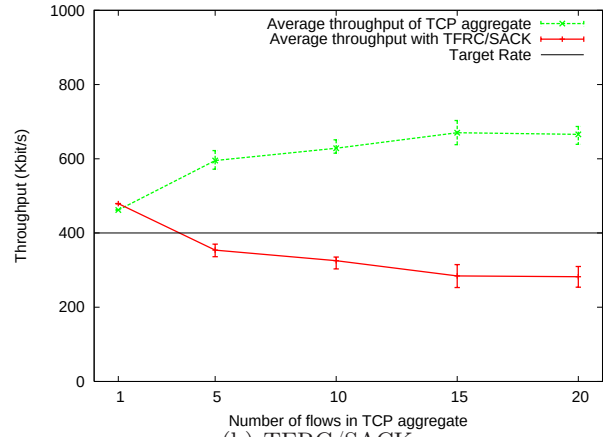
In this study, one flow ($from\ host\ A\ to\ host\ B$) is in competition with a variable size aggregate. The aggregate ($from\ host\ C\ to\ host\ D$) has a variable number of micro-flows ranging from 1 to 20. The RTT of all flows is set to $30ms$ and target rates of (A, B) and (C, D) are equal to $400Kbits/s$. Figure 4.18 gives the results obtained for TCP, TFRC/SACK and CP/QoS flow in function of the aggregate size.

The average application throughput (computed after 150 *seconds*) and the min/max value of ten consecutive measurements are provided in the Figure 4.18. As already underlined for DiffServ networks [SNP99], Figure 4.18(a) illustrates that TCP flow did not reach its target rate. Concerning the TFRC/SACK composition, Figure 4.18(b) shows that the throughput variations are lower than TCP's one. This is due to the "smoothing" property of TFRC congestion control. Nevertheless, on average, the obtained throughput was in the same order of magnitude than TCP. Finally, Figure 4.18(c) confirms the previous results, showing that the CP/QoS flow (A, B) reached the requested target rate regardless of the number of competing micro-flows in the (C, D) aggregate.

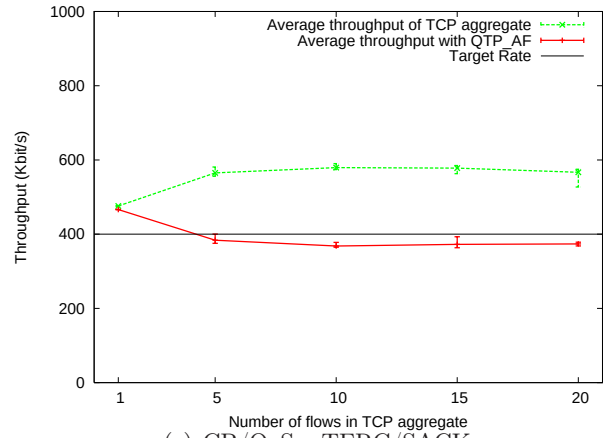
Note that the difference between the target rate at the network level and the average application throughput in Figure 4.18(c) is simply due to the CP/QoS/UDP/IP protocol overhead. Moreover, the min/max interval is the smallest one. For the sake of accuracy and in order to quantify the throughput variations, we give separately in Figure 4.19 the standard deviation of these results. This figure confirms the stability of TFRC and gTFRC over a differentiated network service. Indeed, we can see that the standard deviation for these two congestion control mechanisms is small. The TCP inadaptation to a DiffServ network is unlighted by its large standard deviation.



(a) TCP



(b) TFRC/SACK



(c) CP/QoS: gTFRC/SACK

Figure 4.18 Average throughput according to the number of micro-flows in the aggregate

Illustration over a QoS network with bandwidth guarantee

In this section, we focus on the behaviour of CP/QoS on top of another network level QoS mechanism. This allows us to verify that the proposed protocol can be used over any kind of

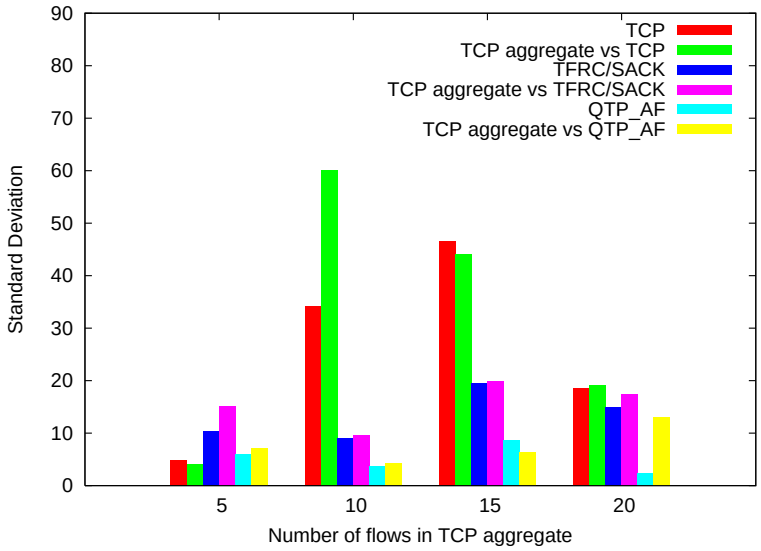
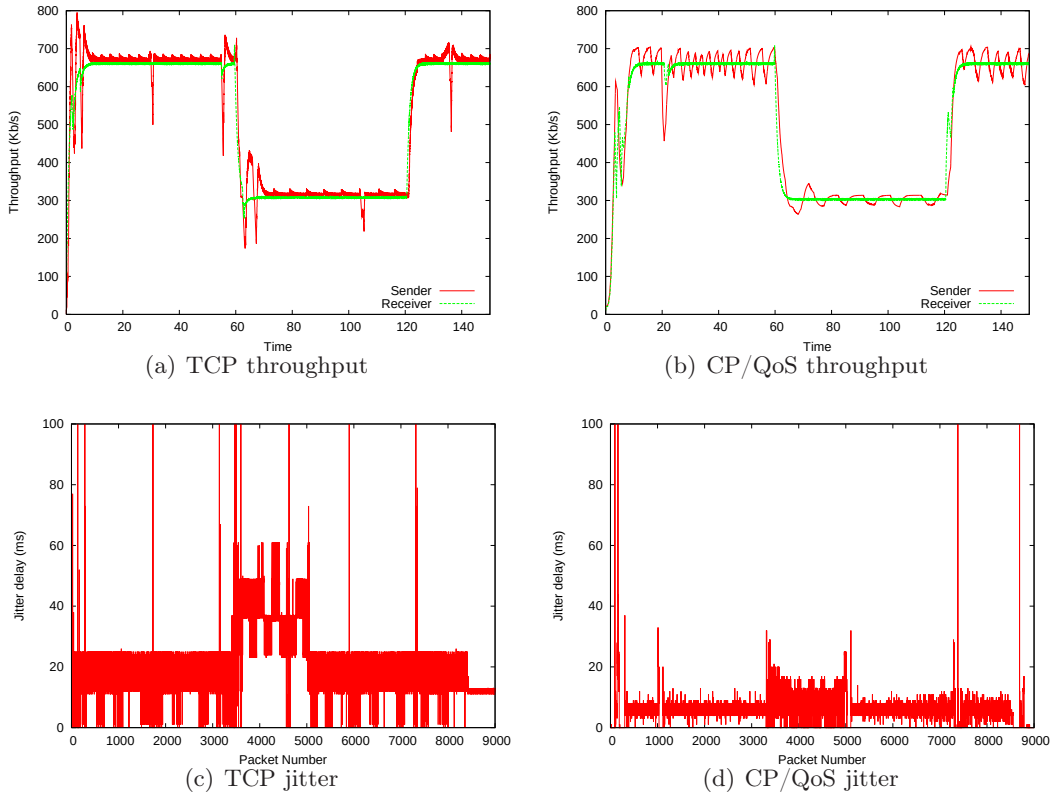


Figure 4.19 Standard Deviation

network providing a bandwidth guarantee. To perform this evaluation, we configure a QoS network with a Class Based Queueing (CBQ) scheduling mechanism [FJ95] that provides a guaranteed pipe of $300kbit/s$ for the studied flow (i.e. TCP or CP/QoS). The network topology used in these experiments remains identical to the one presented in Figure 4.5. The emulated QoS network does not use any admission control. The CBQ is configured in "borrow mode". This means that in the case of no-congestion, the BE traffic can borrow bandwidth from the reserved pipe. This case of configuration is more general as this kind of scheduling algorithm is currently available in commercial routers such as CISCO 4000 and above series. Figure 4.20(a) and Figure 4.20(b) show respectively the throughput of TCP and CP/QoS at the sender and receiver side. Figure 4.20(c) and Figure 4.20(d) show the jitter of these two flows. In these experiments, both flows compete with a UDP flow.

During the experiment, the UDP flow transmitted at $300kbit/s$ except between $[60, 120]$ seconds where it transmitted at $1000kbit/s$. As a result, the bottleneck link was saturated during this interval. Figures 4.20(a) and 4.20(b) give the throughput measured at the sender and receiver side. Once the congestion occurs, the CBQ algorithm starts (i.e. when the UDP flow sends above $700Kbit/s$). Thanks to the CBQ scheduling, both flows obtained their guarantee as shown by these figures.

Figures 4.20(c) and 4.20(d) shows the jitter experienced by both flows in milliseconds. We can see that TCP suffer from a higher jitter than CP/QoS 4.20(d). This is an expected result as TFRC congestion control algorithm emits a non-bursty traffic. Accordingly, the resulting jitter must be lower. Moreover, these measurements show that the composition of TFRC with SACK did not impact on this standard behaviour and that the resulting jitter for CP/QoS was lower than for TCP.



(a) TCP throughput (b) CP/QoS throughput

(c) TCP jitter (d) CP/QoS jitter

Figure 4.20 Jitter of one TCP, CP/QoS flow versus a UDP flow with various throughput

4.7 CONCLUSION OF THE CHAPTER

In this chapter, we have presented the design and the implementation of a fully reliable QoS-aware transport protocol. In order to specify this protocol, we first introduced gTFRC , a specialisation of the TFRC congestion control mechanism, that allows the transport protocol to be aware and to exploit the QoS negotiated with the network service provider. In gTFRC , we identified security issue that can be raised by such an approach. These problems are mainly related to selfish users and misconfiguration of network resources. In the case of a selfish user, the DiffServ conditioners should avoid this kind of deny of service. In the case of a misconfiguration, due to the use of the estimation of the received rate if such a misconfiguration occurs, our proposal will react to it. However, we believe that these security concerns are out of the transport layer scope. We claim that it is definitely not the responsibility of the transport protocol to detect a selfish user or to react to a wrong network configuration.

We implemented this new QoS-aware congestion-control mechanism inside a Java framework. Prior to this implementation, a large range of measurements have been done in the ns-2 simulator [ns2]. The results of this simulation study have been published [LDJ06b, LDJ06a]. In parallel with the implementation of gTFRC , we have integrated gTFRC inside the DCCP protocol in ns-2 and the results of the simulation of gDCCP

have been published in [ELD07]. In this chapter we have just presented the results from the implementation of gTFRC . The results of the emulation campaign show, identically of the implementation campaign, that gTFRC allows a transport protocol to obtain at the network level the negotiated bandwidth.

In order to propose a complete transport protocol, we then composed the gTFRC mechanism with a reliability mechanism. In order to provide reliability we decided to use a SACK-like mechanism that we customised for a datagram-oriented transport protocol. In addition to a loss recovery mechanism we proposed and validated the design of a flow-control especially dedicated for rate-based transport protocol. The result of this composition has been evaluated first at the network layer then at the application layer. We show that, at the application level, our proposal allows the application to obtain a bandwidth close to the one negotiated at the network level whatever the conditions of the network. In particular, we show that contrary to TCP and a reliable TFRC, the obtained bandwidth does not depend of the number of flows against gTFRC . Finally, we show that our proposal is not dedicated only to the DiffServ/AF class. Indeed, we made an experimental evaluation of our proposal in the case of a generic Class Based network service based on the CBQ scheduling algorithm and show that our protocol can still reaches the negotiated bandwidth.

For future work, this proposal should be implemented at the kernel level in order to evaluate this proposition in high throughput networks. Indeed, because the implementation was performed at the application level, we observed a maximum bandwidth of $8Mbit/s$.

Chapter 5

Re-thinking TFRC sender-based architecture

5.1 INTRODUCTION

In the previous chapter, we have shown how a QoS-aware specialisation of TFRC and its composition with a SACK-like mechanism could improve the throughput obtained at both network and application level above a bandwidth-guaranteed network. In this chapter, we will show how the TFRC architecture can be rethinking to light mobile multimedia end-system. This paradigm shift is motivated by the fact that first mobility in the Internet is becoming the rule and second the TFRC smooth rate variation make it a good candidate for the delivery of an efficient transport service to multimedia end-systems. However, in such media-streaming scenarios, if multimedia servers are powerful processing and communication engines, this is not the case fir mobile clients. Indeed, these clients are resource-limited end-systems and are far more sensitive to communication and system processing that should impact as little as possible on the application layer processing.

Therefore, the lightening of recurrent communication processing on light end-systems, that populate increasingly the Internet, is a critical issue for increasing the performance and autonomy of mobile end systems. One of the main costs of the TFRC mechanism comes from the periodic computation of both the RTT and the loss rate of data carried by a connection. In particular, RFC 3448 [HFPW03] proposes the loss rate estimation to be done on the receiver side. A classical receiver-based solution achieves a periodic estimation of the loss event rate before sending it to the sender. This computation requires maintenance of a loss event history data structure. Such a receiver-based solution does not comply with the capacities and resource constraints (i.e. in terms of energy consumption and overall computational performance) of light mobile receivers (e.g. PDAs, mobile phones) which are increasingly pervasive.

RFC 3448 also suggests that this computation could be done on the sender-side: *"It would be possible to implement a sender-based variant of TFRC where the receiver uses reliable delivery to send information about packet losses and the sender computes the packet loss rate and the acceptable transmit rate"*. We developed this idea by specifying and evaluating the design of a sender-based implementation of the TFRC congestion control mechanism. In our proposal, the reliable transfer of feedback packets is ensured by using packet-oriented SACK mechanism [FMMP00]. This scheme is known to be robust to lossy channels while not entailing heavy and complex error control mechanisms [FMMP00]. Moreover, we will see that, because it is located on the flows' servers only, the proposed sender-based approach is more robust to selfish receivers. Indeed, the sender no longer depends on the accuracy and the integrity of the returned information [HFPW03]. Some solutions to secure TFRC from selfish receivers have been proposed in [GG05] using RTSP [SRL98]. Our solution requires fewer and simpler modifications to the TFRC header and algorithm than the proposal in [GG05].

Another sender-based solution has been proposed in [FKP06] where the receiver sends back loss event intervals to the sender. This solution has not to date been either tested or implemented. In comparison to our solution, this solution is supposed to be closer to the original algorithm, but the receiver remains more complex as it has to maintain a structure able to differentiate a loss from a loss event.

This chapter is structured as follows: section 5.2 introduces the context of this study and provides some background information. Section 5.3 gives insights into the design of the new sender-based congestion-control-protocol architecture. Section 5.4 compares the performance of the proposed congestion control protocol with respect to the standard TFRC implementation. We quantify the benefits of our proposal in terms of algorithmic processing and communication load in section 5.5. Finally, section 5.6 provides conclusions and future directions.

5.2 CONTEXT AND RELATED WORK

TFRC estimates the equivalent TCP sending rate X from equation (3.3). This equation depends on the mean packet size s and two periodically processed parameters: the packet loss event rate p and the round trip time RTT. In this equation, RTO refers to the TCP retransmission timeout value which is usually a linear function of the RTT.

During the initialization phase, TFRC acts as TCP does during the slow start algorithm. This slow start phase also occurs during the transfer after the RTO timeout expires. This phase is followed by a congestion avoidance phase as soon as the receiver detects a loss. At this step, TFRC needs to estimate the loss rate in order to compute the sending rate X. The receiver evaluates the packet loss rate by a sliding window-based loss-history structure. This structure stores the eight most recent loss-event intervals and makes it possible to process the loss event rate from low path filter that smoothes the loss event variation. A loss event and its related interval of packets is defined as one or more lost packets during a duration of a least one RTT[HFPW03]. In other words, several packets lost during an RTT define a single loss event and the duration of a loss interval is greater than or equal to the RTT. The algorithm used at the receiver side is given in Figure 5.1.

```
ReceivePacket() {
  Add packet to packet history;
  p_new = new value of packet loss rate;
  if (p_new > p_old){
    stop feedback timer;
    do CreateFeedback();
  }
}
CreateFeedback() {
  compute average packet loss rate;
  calculate measured receive rate;
  prepare and send feedback packet;
  restart feedback timer;
}
```

Figure 5.1 Original algorithm of the receiver

Two main issues can be identified in the receiver-based implementation algorithm. Firstly, the receiver must continuously maintain and update the loss-event history data structure. The management of this data structure is an undesirable processing and memory management overhead for resource-limited mobile receivers. Secondly, the receiver has to continuously process the loss-event rate and send it to the sender at least once per RTT, and as soon as it observes a loss event rate increase. Once again, this processing load squeezes the remaining processing capacity of the receiver. Moreover, such a receiver-based implementation cannot guarantee that selfish receivers do not try to trick the sender by inaccurately reporting the loss rate in an attempt to obtain higher bandwidth [GG05].

5.3 DESIGN

This section presents the design of our sender-based TFRC protocol named TFRC$_{light}$. The design of this protocol is based on the shifting of the loss-rate estimation to the sender side. We identify and propose several changes entailed by this shifting mainly in the feedback packet structure and in the data structures managed by the receiver. The aim of our new TFRC protocol architecture and design is to reduce the receiver load. We discuss in this section the design of TFRC$_{light}$ by first presenting the problems that resulted from shifting packet-loss-rate estimation. Then, we define and experimentally validate efficient solutions to these problems.

5.3.1 Notification of packet loss

In the original TFRC, the receiver has to periodically send feedback information to the sender. These feedback messages contain two parameters that allow the sender to estimate the current RTT value. These parameters are respectively (1) the timestamp of the last packet received (`Last Timestamp`), and (2) the amount of time elapsed between the receipt of the last packet and the generation of the feedback (`Processing Time`). We present these fields of the TFRC header in Figure 5.2.

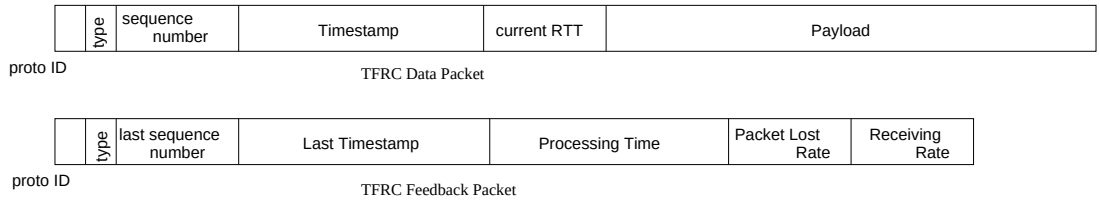


Figure 5.2 Example of TFRC header

Moreover, feedback packets also contain information about the packet-loss rate (`Packet Loss Rate`) and the received throughput (`Receiving Rate`) as processed by the receiver. In TFRC$_{light}$, the packet-loss rate is no longer processed and returned by the receiver. Nevertheless, the receiver still remains the only entity able to detect the loss of a packet and to notify the sender of this loss.

In order to perform this notification, we propose the maintenance of a compact and light data structure at the receiver. This data structure is a simple bits vector (i.e. a SACK vector) that describes, from a given packet number, the distribution of packets received and lost. In other words, if a given packet is received, the bit is set to 0 otherwise 1. This vector is periodically sent to flow source. Such a data structure leverage on the SACK mechanism used when some degree of reliability is needed. Therefore, in this case our approach does entail any additional data structure at the receiver. Thus, two services are delivered for the price of one.

When its sending period is lower than the duration covered by the SACK vector the SACK vector offers redundancy that contributes to the reliable delivery of loss information. The value of the feedback packet sending period will be discussed in the next section. The right vector length can be chosen by considering that the sender-based and receiver-based implementations should react similarly to packet losses. Indeed, as defined in [HFPW03], the sender no-feedback timer expires after $4 * RTT$, where RTT is the exponentially weighted moving average of the round trip time sent by the sender in each packet. A SACK-based mechanism is intrinsically robust to a maximum period of data losses equivalent to the vector range. Then, the loss vector length should cover at least:

$$4 * RTT * PacketSendingRate$$

where, $PacketSendingRate$ is the sending rate included in each data packet header or computed by the receiver as the received-packet rate. In order to reproduce the no-feedback timer behaviour of the standard receiver based version of TFRC, the loss information vector length must be dynamically recomputed with a period of RTT.

The data structure used to compute SACK is a circular buffer, with a pointer keeping track of the most recently received packet. In the next section we first consider a simple initial scheme for managing this structure. Then, from the issues raised by this scheme, we will propose a solution that conforms to the standard TFRC behaviour.

The message headers for the simple initial scheme are given in Figure 5.3.

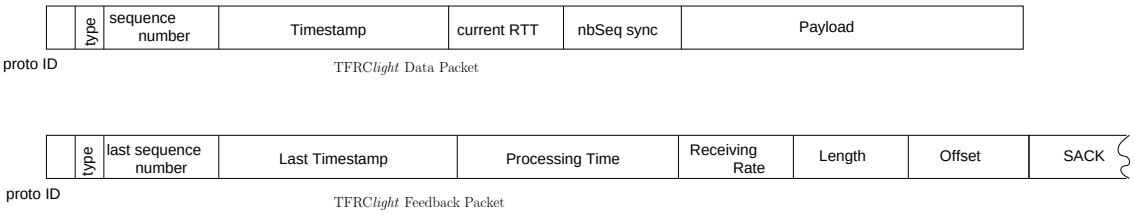


Figure 5.3 Modification in TFRC header

5.3.2 Loss event definition in TFRC$_{light}$

Although the previously introduced data and protocol data unit structures are necessary for implementing an efficient sender-based TFRC protocol, they are not sufficient. Indeed, the loss-history structure is based on the loss-event definition given in [HFPW03]. A loss event is defined as the detection of one or more lost packets during one RTT. For keeping track of loss events, the receiver needs the receiving time of each packet to detect if lost packets correspond to the current loss-event interval.

Since the sender and the receiver cannot maintain a synchronous behaviour, the simple SACK structure previously introduced does not allow the sender to construct an accurate loss-event-history structure even if feedback packets are sent every RTT. Indeed, without a careful design, in certain cases, a loss event may be falsely detected. In Figure 5.4, we give an illustration of such false detection. The time axis is used to represent the arrival time of the data packets. We also show on this axis the times, t_n, when the receiver sends feedback. As an example, we show the tail (i.e. the SACK vector) of three feedback messages below this axis. At t_1, the feedback message reports two losses represented by the two bits set in the `SACK` field. The `Offset` is equal to 100.

In the original TFRC, a timer of RTT time units should have been triggered at the estimated receiving time of the lost packet with the sequence number of 106. This timer range is represented in Figure 5.4 by two-way arrows. At t_2, when the receiver sends its second feedback packet, the SACK vector `Offset` is now equal to 112 and as the RTT period is expired, a loss event should have been detected. At this time, the traditional TFRC algorithm closes the previous loss interval and restarts a new one from packet number 119. Finally at t_3, the losses reported for packets 125 and 127 belong to the previous loss event as the RTT timer expired at packet number 130. Since no other packet is lost after this expiration there is no new loss event. The problem of false detection can potentially result from an interpretation as a loss event of this third feedback with `Offset` field which is equal to 124 and its two marked bits in the vector.

As shown in Figure 5.4, the TFRC mechanism is supposed to see two loss events (symbolized by the two RTTs). In TFRC$_{light}$, if we just shift the packet-loss-rate estimation, since there is no information about the estimated time of the packet loss, and the sender and receiver are not synchronous, the TFRC mechanism will see three loss events. Indeed, it will receive three disjointed feedback messages (one per RTT) with a non-null `SACK` field. Therefore, a simple logical interpretation of these feedbacks leads to the identification of three loss events instead of only two.

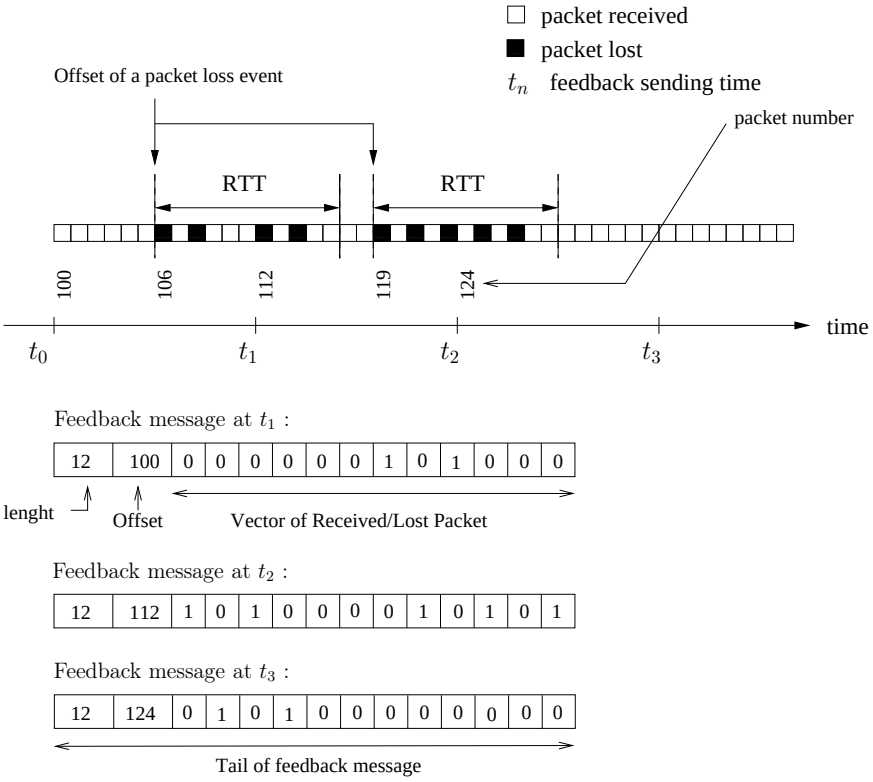


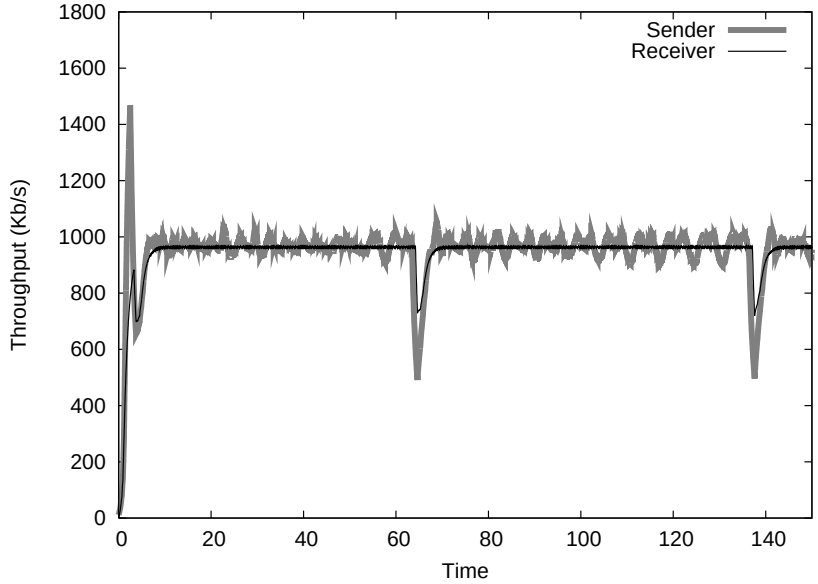
Figure 5.4 Illustration of a the definition of the loss event

Figure 5.5 illustrates the impact of this false detection problem. We give in this figure the instantaneous throughput measured at the sender and instantaneous throughput measured at the receiver. Figure 5.5(a) shows the resulting throughputs of TFRC$_{light}$ with a bad interpretation of loss events. The experiments involve an architecture with two nodes that generate traffic and are connected by a link with fixed capacity of $1Mbit/s$ and $RTT = 100ms$. In Figure 5.5(a), TFRC$_{light}$ detects five loss events just after the slow start phase (between $t = [0, 10]$)[1]. However a correct implementation of TFRC would have seen only four loss events as illustrated in 5.5(b).

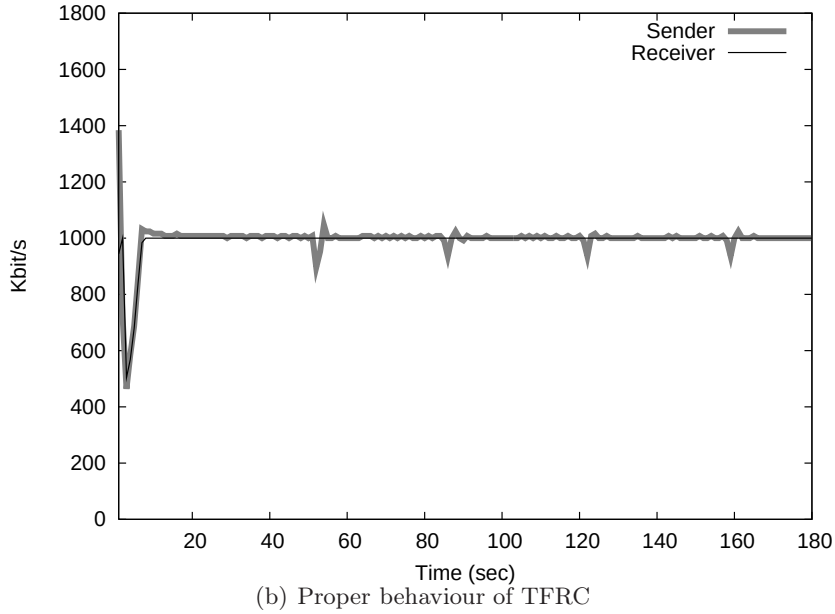
As a result, when a new loss event occurs (i.e. $t = 63s$ and $t = 137s$), the sender will decrease its emission rate more than is needed. In Figure 5.5(a), this behaviour can be seen with the two rate dips. This throughput decrease is explained by the way the loss-history structure is built. Indeed, as the mechanism observes successive loss events, the corresponding entries in the loss-history structure will be filled with loss intervals shorter than they should be. When a new loss event occurs, these erroneously sized loss intervals raise the resulting value of the loss event rate. This loss rate causes an excessive reduction of the sending rate as given by equation (3.3).

In order to solve this issue we propose the following modifications.

[1]Observed by the addition of a memory variable inside the core protocol



(a) Behavior of TFRC$_{light}$ with a falsely detected loss event



(b) Proper behaviour of TFRC

Figure 5.5 Comparison of TFRC$_{light}$with a false detection and a usual TFRC in a network with a bandwidth of $1Mbit/s$, and an RTT=$100ms$

New receiver algorithm

At the receiver side the structure remains similar to the one presented in the previous section. The algorithm used by the receiver is shown in Figure 5.6.

```
ReceivePacket(){
  Manage SACK vector;
}
CreateFeedback(){
  calculate measured receive rate;
  prepare and send feedback packet;
  restart feedback timer;
}
```

Figure 5.6 Receiver algorithm

In this proposal, the receiver is no longer responsible for computing the packet loss rate. Nevertheless, the receiver has to keep updated the SACK vector. This is done by the function "Manage SACK Vector". In this function, the receiver set to 1 the value corresponding to the received packet number in the SACK structure. This algorithm supposes the existence of a new structure that records the arrival or loss of packets.

Modification at the sender side

In order to detect a loss event at the sender side, the server has to set up a structure that stores information about when packets were sent. This structure is identical to the one that traditional receiver-based TFRC receivers use to compute the packet-loss rate, except that instead of keeping trace of the packet-arrival time, this new structure stores the packet-sending time.

Based on this new structure the sender is now able to detect loss events from a sender perspective by considering the sending time of the packets reported as lost in the received SACK vectors. Furthermore, because the sender keeps the packets sending time, the `TimeStamp` field is no longer needed in both data and feedback headers. Figure 5.7 gives the resulting new structure of the TFRC$_{light}$ headers associated with the data and feedback packets.

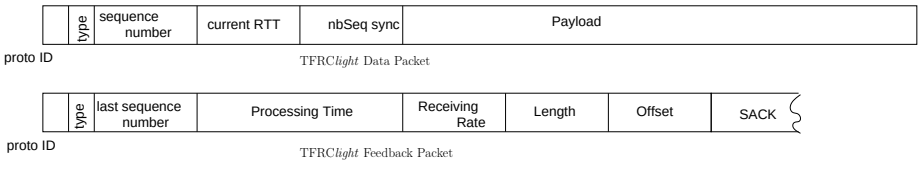


Figure 5.7 Modification in TFRC header for the loss event detection second solution

Translation from Loss History to Loss Events: A Sender Perspective

In our proposal, the sender is now aware of the sending time of each packet. This information, combined with the received SACK vectors, allows the sender to process the packet loss rate as detailed in Figure 5.8.

```
for(int i=0; i<lenghtACK; i++)
{
  if(vector[i]==0)
    add Packet(offset+i) loss History;
    p_new=new value of packet loss rate;
  else
    translation from loss history to loss event;
}
compute average packet loss rate;
```

Figure 5.8 Analysis of the vector of Ack

In section 5.2 of RFC 3448, the authors explain how to build loss events from the loss history. This operation needs:

- S_{loss} the sequence number of the lost packet;
- S_{before} the sequence number of the last packet to arrive, such that $S_{before} < S_{loss}$;
- S_{after} the sequence number of the first packet to arrive, such that $S_{loss} < S_{after}$;
- T_{before} the reception time of S_{before};
- T_{after} the reception time of S_{after}.

In the presented solution, the sender is not aware of T_{before} and T_{after}. Nevertheless, the sender must estimate the arrival time of S_{loss}. In our proposal, we use sending times, not arrival times, to build loss events. These sending times are corrected by the following factor, which the sender evaluates whenever it receives a feedback (where X_{sent} and X_{recv} are respectively the sending and receiving rates):

$$\alpha = \frac{X_{sent}}{X_{recv}}$$

The determination of the new event is accomplished in the same way as in the original TFRC except that the time reference is no longer the arrival time but is now the sending corrected by the factor α. Base on this new loss event, a loss interval is built and stored in the loss history structure at the receiver. Then, the receiver use the same sliding window algorithm as described in [HFPW03] in order to compute the packet loss rate.

Discussion

As feedback messages are not systematically sent when a loss is detected, we recommend that the feedback message sending interval should equal at least one per RTT.

5.4 VALIDATION OF TFRC$_{LIGHT}$

In this section, we present an evaluation of our proposal. This evaluation is done using several experiments on an emulate network. We have implemented a user level prototype of TFRC$_{light}$ in Java. We have evaluated the TFRC$_{light}$ protocol over a simple testbed composed of two end-systems and a network emulated by a FreeBSD/Dummynet pipe [L. 97].

We used both TCP and TFRC as the basis for the comparison against TFRC$_{light}$. An exhaustive comparison between these protocols can be difficult to obtain. In the rest of this section we will study through representative example the behaviour of TFRC$_{light}$. Then, for evaluation purpose, we will use metrics as proposed in [GG07, BG92, Jai91].

5.4.1 Evaluation Strategy

The performance evaluation of TFRC$_{light}$has been achieved regarding four criteria:

- Efficiency (throughput)
- Intra-protocol fairness
- TCP-friendliness
- Stability (oscillations)

Here, we provide the definitions of these metrics. In the next section, we will quantify our scheme in terms of CPU and memory use.

Efficiency (throughput)

In [GG07] a protocol efficiency is defined as the aggregate throughput of all the concurrent flows. Here we will apply a normalised definition to our study is described in equation (5.1).

$$E = \frac{\sum_{i=1}^{n} \overline{x_i}}{C} \tag{5.1}$$

Suppose there are n TFRC$_{light}$ flows in the network with a bottleneck of C $Mbit/s$ and let's be $\overline{x_i}$ the throughput of the i^{th} flow then the equation (5.1) represent the percentage of used bandwidth.

Intra-protocol fairness

The fairness metric represents how flows share fairly the bandwidth. In order to quantify this, the commonly used method is the $max - min$ fairness[BG92]. In this method the lowest throughput is maximised. In the following part of this section, since there is only one bottleneck in all experiments, we will use the Jain's fairness [Jai91] in order to measure this characteristic of TFRC$_{light}$. Therefore, this fairness is given by the equation (5.2).

$$F = \frac{(\sum_{i=1}^{n} \overline{x_i})^2}{n\sum_{i=1}^{n} \overline{x_i}^2} \tag{5.2}$$

Where in this case $\overline{x_i}$ is the average throughput of the i^{th} TFRC$_{light}$ flow and n is the number of flows competing for the bandwidth. F is always inferior or equal to 1. If $F = 1$, then all flows have the same throughput.

TCP-friendliness

TCP-friendliness is nowadays subject to discussion among the networking community. In particular, some researchers claim that, from different point of views, this qualification for a flow is not a real criterion. In this study, we used a metric that follows the axiom that defines that a flow is TCP-friendly if *the non-TCP source obtains a long-run term average sending rate not larger that the one TCP would have obtained under the same circumstances*. This results in evaluating the TCP-friendliness with the equation (4.2).

Stability (oscillations)

The last metric considered in this section is a stability criterion. TFRC is renowned for being well-adapted for multimedia traffic due to its capacity to deliver a smooth throughput [Wid00].

In order to quantify this stability, we consider the average throughput for each time unit interval. For each time interval we compute the standard deviation of the throughput for each flow [JWL04] and obtain the metric equation given by the formula (4.3). In the present case, x_i is the throughput of the i^{th} TFRC$_{light}$ flow, n is the number of flows, $x_i(k)$ is the throughput of the i^{th} TFRC$_{light}$ flow for the k^{th} time interval and m is the number of time intervals.

5.4.2 General behaviour of the TFRC$_{light}$

We have implemented a user-level prototype of TFRC$_{light}$ in Java. We have evaluated the TFRC$_{light}$ protocol over a simple testbed composed of two end-systems and a network emulated by a FreeBSD/Dummynet pipe [L. 97]. For all experiments, the bandwidth and the RTT are respectively set to 1Mbit/s and 100ms. In both figures 5.9, we report the sending/receiving instantaneous throughputs measured respectively at the sender/receiver sides. The results of our experiments show that our sender-based proposal have the same behaviour as traditional receiver based TFRC implementations.

We obtained many measurements to validate this new architectural design and report in this section a representative sample of the results. It is always difficult to compare the performance of a real implementation and a simulated one as the simulation reproduces an ideal case without the overhead introduced by real measurements. Nevertheless, we show that the TFRC$_{light}$ receiver throughput is as stable as the ns-2 version receiver throughput. Concerning the sender throughput, more oscillations occur in TFRC$_{light}$ than in ns-2 TFRC. This can be explained by the overhead introduced by our user level TFRC$_{light}$implementation.

In the experiment illustrated in Figure 5.9, we introduced an UDP flow with a rate of $500Kbits/s$ between $t = [30sec, 90sec]$. This test aimed to verify the responsiveness of TFRC$_{light}$ compared to ns-2 TFRC. In Figure 5.9, due to the packets being multiplexed with a non-responsive UDP flow, both implementations decreases during the UDP flood. Furthermore, both implementations reacted the same way to the losses induced by the UDP flow. When the UDP flow stopped, both implementations responded similarly. Eventually, we concluded from this scenario that the modifications proposed and implemented in TFRC$_{light}$ result in a behaviour similar to ns-2 TFRC.

5.4.3 Efficiency, fairness and stability of TFRC$_{light}$

In the set of experiments discussed in this section, we have measured different criteria when TFRC$_{light}$shares a network with other TFRC$_{light}$flows only. The topology of the network is displayed on the Figure 5.10.

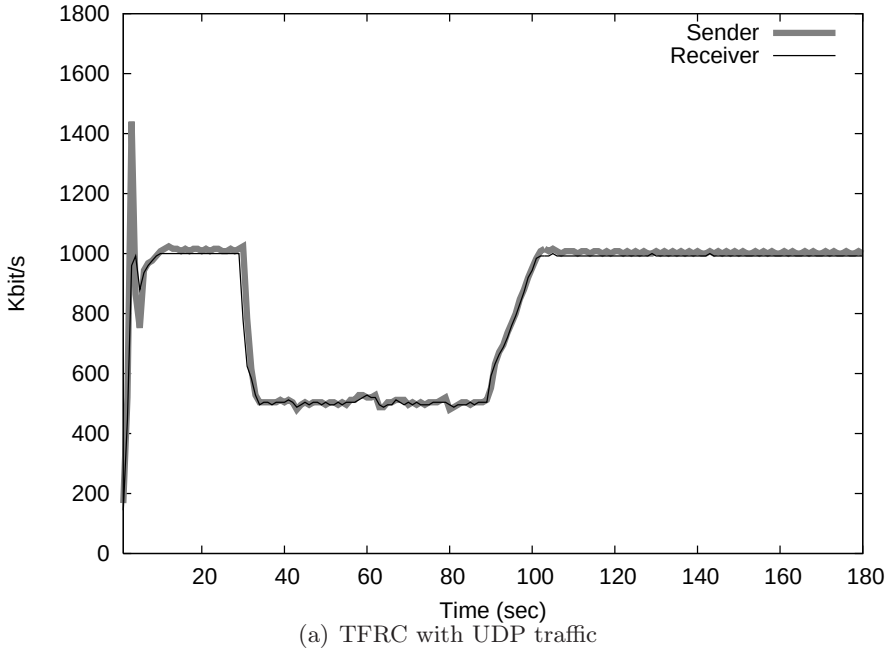
In this topology, we made the number of TFRC$_{light}$ flows vary from 1 to 4 following two patterns. These two patterns differ from the communication stopping time of their streams. Indeed, in the first patterns, every flow starts at the same time but does not have the same duration as depicted in Figure 5.11. In the second pattern, the starting and stopping time of every flow is the same. Thanks to these two different patterns we are able to study the long run behaviour of our proposal and its reactiveness when flows leave the network.

Different Stopping Times

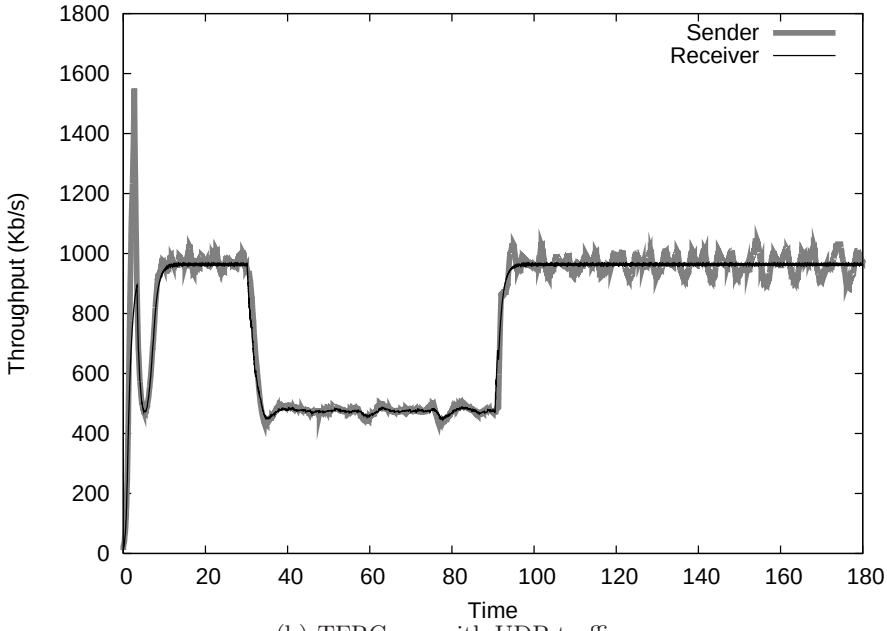
Figure 5.12 represents the perceived throughput at the receiver side. This throughput was computed using a time sliding window of one second as explained in [FSa00].

In Figure 5.12, we show that our proposal equally shared the bandwidth between flows. The difference observed during the first period of the experiment can be explained by two main characteristics. First, our implementation was in Java, therefore the four flows shared the same virtual machine and the last flow get some difficulties to start. Second, all the flows experience their first loss event at different times. The differences in the observation of the loss event explain why some flows have more difficulty reaching the equilibrium throughput again, because the RTT is higher following the increase and variance in buffering delay.

The behaviours represented in Figure 5.12 are confirmed by the previously introduced metrics displayed in Table 5.1. These results have to be compared to what TCP would experience in the same conditions, as given in Table 5.2.



(a) TFRC with UDP traffic



(b) $TFRC_{light}$ with UDP traffic

Figure 5.9 TFRC and $TFRC_{light}$ with a network bandwidth of $1Mbit/s$, an RTT=$100ms$ and introduction of an UDP flow at $t = [30s, 90s]$

Long-term Behaviour

We present in this section the characteristics of our proposal in the case of a long-run communication. In order to process this analysis, we performed the same experiment as previously but without different stopping times.

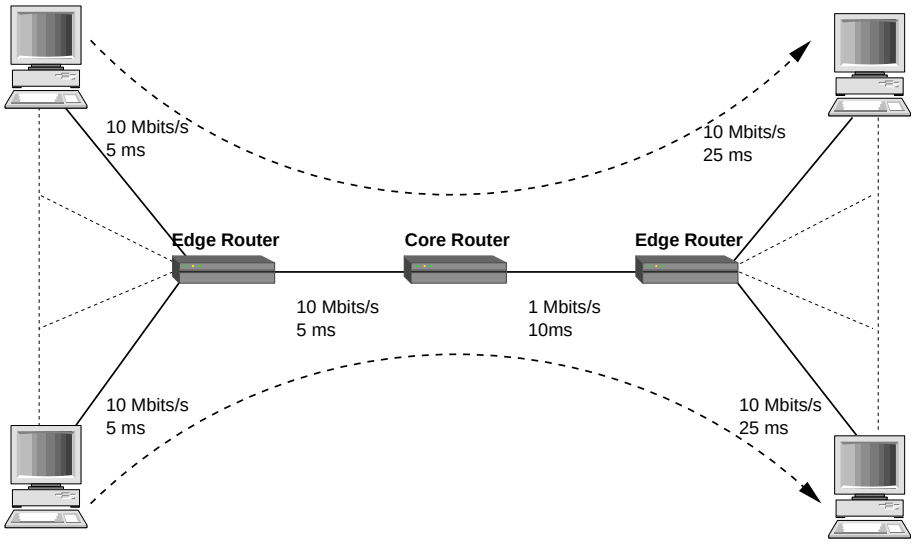


Figure 5.10 Topology

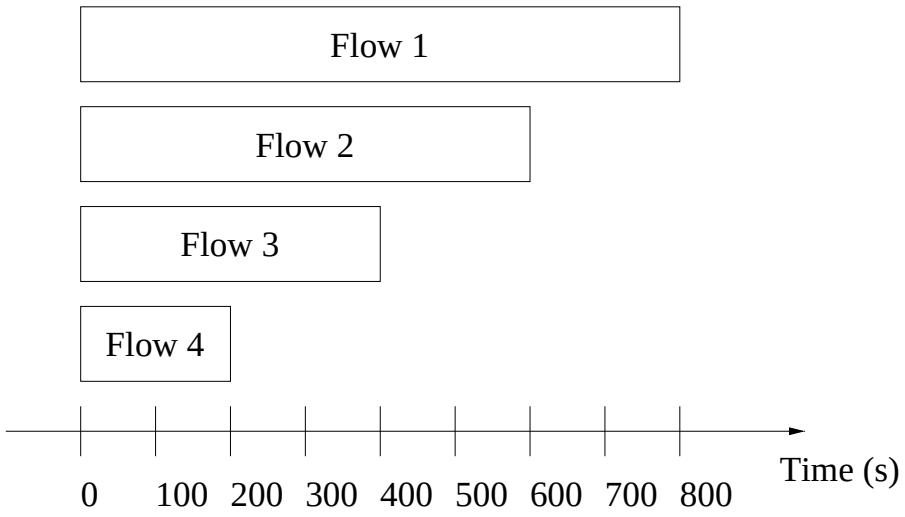


Figure 5.11 Stopping times of the different flows

As expected, the results for the long run behaviour of TFRC$_{light}$ were a stabilised adjustment of the first test-period of the previous set of experiments. As a result, TFRC$_{light}$ is more stable in the long-run experiments than in short-run experiment. In the same way, TFRC$_{light}$ reached the equilibrium and therefore the intra-fairness property was enforced. Concerning the efficiency metric, TFRC$_{light}$ is more efficient in the long term behaviour study than in the previous one due to the fact that the equilibrium is reached compare to the period $0 - 200s$ in the previous experiment. This is explained by the nearly equal to one intra-fairness metric.

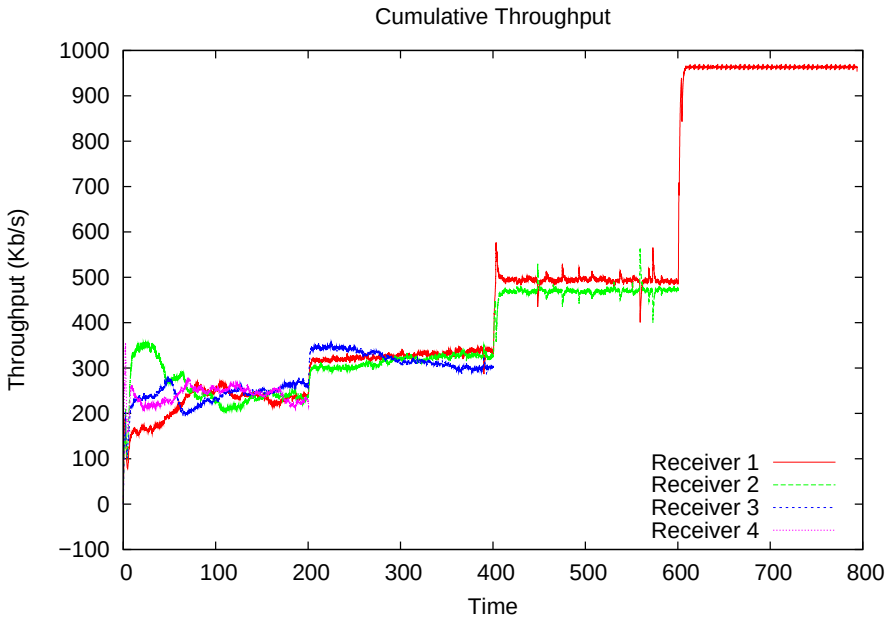


Figure 5.12 Receiving throughput of the different flows

	0-200s	200-400s	400-600s	600-800s
Efficiency	0.949	0.974	0.961	0.958
Stability (oscillations)	0.137	0.043	0.031	0.035
Intra-protocol fairness	0.996	0.999	0.999	1

Table 5.1 Result of the delayed start experiment

	0-200s	200-400s	400-600s	600-800s
Efficiency	0.964	0.965	0.965	0.975
Stability (oscillations)	0.023	0.079	0.167	0.218
Intra-protocol fairness	0.993	0.999	0.999	1

Table 5.2 Result of the delayed start experiment with TCP

	$TFRC_{light}$
Efficiency	0.965
Stability (oscillations)	0.058
Intra-protocol fairness	0.999

Table 5.3 Result of the long-run experiment

5.4.4 TCP-Friendliness

In the following experiments, we have shown that the proposed sender-based TFRC remains TCP-friendly. The results of the TFRC-friendliness property are shown in Table 5.4. These measurements give the average throughput observed at the receiver after $200s$ of transfer. We have driven the first experiment with 5 $TFRC_{light}$ flows only. We also studied the multiplexing behaviour of $TFRC_{light}$ flows with TCP and TFRC flows. The results summarized in Table 5.4 show that $TFRC_{light}$ flows occupied a fair share of the bandwidth when multiplexed with TCP and TFRC flows. This table shows that our proposal is friendly with TCP, as it did not obtain a bandwidth superior that of TCP. On the contrary, $TFRC_{light}$is less friendly with our implementation in user space of TFRC. This can be explained by the fact that $TFRC_{light}$, as explained in the section 5.3, does not react as quickly as the original TFRC algorithm when a loss occurs in the network.

	T($TFRC_{light}$)	T(TCP)	T(TFRC)
5$TFRC_{light}$ and 5TFRC	1.05	N/A	0.95
5 $TFRC_{light}$ and 10TCP	0.92	1.08	N/A

Table 5.4 Inter-Protocols TCP-friendliness

5.5 QUANTIFICATION OF THE GAINED RESOURCES

In Table 5.5, we summarize the benefits and drawbacks of the proposed design compared to the original algorithm.

The main advantages of our solution are the removal of the packet-history structure and the removal of the computation of the packet loss rate at the receiver. Conversely, we have introduced a new light structure that allows the receiver to build the Sack vector sent to the sender in feedback messages. This structure has a size of the order of $4RTT * Bandwidth/(packetsize)$. For instance, in the case of a transmission with a bandwidth of $1Mbit/s$, an RTT of $100ms$ and a packet size of $1000Bytes$, the structure should have a maximum size of $50bits$. This structure is actualized for each data packet received. In the original receiver-based design of TFRC, the receiver had to manage a more com-

benefits	suppression of the loss history structure no processing of the packet loss rate protection from misbehaving receivers simpler timer management simpler sender's algorithm
drawbacks	new structure for Sack vectors management loss events built from sender point of view feedback only sent periodically

Table 5.5 Summary of the benefits and drawbacks of TFRC$_{light}$

plex structure that stores information concerning the arrived or lost packets. The stored information includes:

- the packet timestamp (16bits);
- the packet size (8bits);
- the arrival time (16bits).

Therefore, the elementary size of an entry is $40bits$. Furthermore, this structure potentially entails an unbounded size. Indeed, this structure is emptied after detecting a loss event only. As an example in Figure 5.5, there are no losses between $t = 63$ and $t = 137$. During this entire period, the structure has to be updated at a rate of $1Mbit/s$ which corresponds to $125packet/s$. This structure for the given example would contain:

$$40 * 125 * (137 - 63) = 370Kbits$$

when it can be released. In this particular case, with TFRC$_{light}$, the memory use would decreases from $370Kbits$ to $50bits$. This comparison remains true in the case of the proposal of sender-based as proposed in [FKP06]. Indeed, in this proposal, the receiver is still responsible for the differentiation between a loss event and a packet lost. Therefore, it still needs to maintain a structure storing information of the arrival time of the packet as described above. Nevertheless, the following CPU's cycle comparison only applies to the original TFRC if the sender-based option is configured only to check the value compute the packet-loss rate. Indeed, this option can also be activated only to double check the packet loss rate field in the feedback header. Therefore, the receiver still computes this estimation.

To estimate the computation benefit of our proposal, let's consider how in normal TFRC [HFPW03] the loss rate estimate is processed for every received packet as shown in Figure 5.1. The basic algorithmic sequence for computing the loss rate estimate entails the following set of elementary arithmetic operations: eight additions, eight multiplications, one division and one maximum operation. For instance, at rate of $1Mbit/s$ with a packet size of $1Kbyte$, this estimation should be computed 125 times per second. These elementary operations can be translated into CPU cycles as follows[2]:

[2] According to Intel PIV documentation

- division = 70 cycles
- multiplication = 15 cycles
- addition, maximum = 0.5 cycles

As a result, for the given example, in the original TFRC, the receiver has to use 24312.5 $cycles/s$.

Furthermore, after a slow start phase, the receiver has to initiate its loss history. This initialization is done from the inversion of equation (3.3) in order to find the packet loss rate corresponding to the measured received rate. This initialization is usually done with a binary search and uses the list of elementary operations sum up in Table 5.6.

	$+$	$*$	$/$	$sqrt$
binary search	$4n+4$	$8n+8$	$2n+2$	n
CPU cycles	0.5	15	70	70

Table 5.6 List of the number of elementary operations ($n = number\ of\ iterations$)

The worst case of this binary search can be observed when this algorithm diverges, which can occur when the solution of the inversion of (3.3) is outside the $[0, 1]$ range. This potential of divergence leads to an upper bound on the number of iterations done during the binary search. Therefore, in order to compute the inversion of (3.3) for most cases, the maximum number of iterations is usually set to 50. Indeed, we implemented the binary search of the inversion and found out that the algorithm converges in 15 iterations for $RTT = 400ms$ and $bandwidth = 1Mbit/s$.

In conclusion, for the worst case it takes 16862 CPU cycles for the initialization process. In our proposal, all of this computational process is achieved at the sender side. Moreover, we have shown in section 5.4 that this simplification entails a congestion-control behaviour that strictly conforms to receiver-based TFRC implementations.

5.6 CONCLUSION

In this chapter, we have presented the design of a sender-based TFRC congestion control mechanism This design is driven by the aim of shifting the computation of the loss rate estimation from the receiver to the sender, in order to alleviate the processing and memory needs of "light" receivers. This shifting requires the sending of loss-resilient feedbacks, and is accomplished through the use of a SACK-like mechanism. This results in a significantly lightened computational load on the receiver which is particularly useful for mobile clients with computation and energy constraints.

We have shown that the proposed sender-based TFRC architecture behaves identically to the official ns-2 implementation and remains friendly to TCP streams. This validation has been accomplished through well accepted metrics which confirmed that our architecture

remains as efficient as the original TFRC. We have also quantified the benefits of this shift from the perspective of computations and memory.

Furthermore, the proposed solution allows the security issues raised in [HFPW03] to be resolved. The way to resolve these issues are not explicitly explained in this chapter, but will be explained in a future work. These security issues are related to the forwarding of false loss event rates by the receiver. Such misbehaviour is no longer possible with our solution when associated with nonce mechanisms and will be detailed in a future extension of the proposed solution. We plan to further validate our proposal by performing a large range of experimental measurements on a multi-hop testbed.

Chapter 6

Understanding and improvement on the same variation

6.1 INTRODUCTION

Previous Chapters have presented two architectural modifications of TFRC. These two contributions aim to adapt the current TFRC mechanism to either the quality of service offered by the network layer or the capability of entities on which the protocol is running. These proposals have been done without modifying any parameter defined in the original mechanism.

Among all these parameters, two of them are closely related to the computation of the packet loss rate of the transmission and by consequence to the sending rate. These two parameters are the initial packet loss rate estimation and the loss history weights (i.e. defined as a set eight real constants). These two variables are important for the computation of the packet loss rate. Indeed, the initialisation of the loss history structure is crucial since at the beginning of communication the loss history structure is empty. When a loss event occurs the loss history has to be initialised. Because of the initial slow-start phase associated to TFRC, we have to initialise the loss history and then process the loss event rate just from the occurrence of the first and single loss event. In order to perform this initialisation, TFRC implementations have to invert the TCP throughput equation taking the received rate as the main parameter of the stopping criterion. Based on the value found by this process the receiver initialise the loss history structure.

The second parameter responsible of the computation of the packet loss rate in TFRC is the set of weight applied to the loss history. These weights allow to decide how long the memory of loss event is kept, via the configuration of the number of loss-event intervals it manages. Furthermore these weights allow the mechanism to decide of the importance of every loss interval that composes the loss-event history. For example, if all the weights were equal, obviously every loss interval would be of the same importance regardless of the

loss occurrences. This could be relevant in the unlikely hypothesis of an invariant network loss distribution. These weights can be tuned but are generally limited to eight and for the definition of their value follow the recommendation of [HFPW03].

In this chapter we first propose an optimisation of the loss history initialisation using a numerical analysis of the TCP throughput equation. We then present a first approach for studying the relation between the loss rate computation and the weight used in TFRC following different lost patterns using a discrete event model of TFRC.

This chapter is organised as follows: Section 6.2 presents the problem of the loss history initialisation, its optimisation and the evaluation of the proposed algorithm. Section 6.3 presents an analysis of the loss history behaviour, then we introduce a simple model of TFRC using the Scilab software, finally we propose a first approach for the study of the weights in TFRC based on a 3-states Gilbert model of network losses. Finally, Section 6.4 concludes and gives some perspectives for this work.

6.2 LOSS HISTORY INITIALISATION

The main characteristic of TFRC is the use of a TCP equation model which provides a much lower throughput variation over time than TCP. As a result, it is more suitable for multimedia applications such as audio/video streaming or voice over IP.

In addition to round trip time and received estimated throughput,TFRC algorithm needs an estimation of the packet loss rate. This loss estimation is computed by the receiver and sent periodically to the sender where the rate control is performed. The initial packet loss estimation is crucial as it determines the sending rate in the congestion avoidance phase from a consistent initialisation of the loss-history structure as described in [HFPW03] and the receiver to correctly initialize its loss history events. Indeed, the subsequent estimations of the packet loss rate are based on a weighted moving average using this history. As a consequence, the initialization of this structure has an impact on the sending throughput and on TFRC overall performances (for further details see section 6 in [HFPW03]).

Since TFRC equation can not be analytically solved due to the higher exponent, the method proposed in [Wid00] is based on a binary search process. In every iteration, this process converges towards the solution of the equation by halving the range of study; by the end, the result will be chosen as the middle of the last range. This method is used by all the early available TFRC implementations such as in ns-2 or DCCP implementations. To the best of our knowledge, no efficient method has been formulated. In this section, we show that the binary search is not efficient and needs a large number of iterations to achieve the 5% accuracy required by the TFRC RFC [HFPW03]. Then, we propose a method faster than the binary search method.

The rest of this section is structured as follows: part 6.2.1 states the problem and explains our proposed method. Part 6.2.2 provides numerical results and analytical analysis of them. Finally, part 6.2.3 provides conclusions and perspectives.

6.2.1 Optimization of the loss rate computation

In this section, we present the initialisation problem of the loss event rate and loss history and provide an algorithm that offers an efficient solution to this issue.

Problem statement

TFRC uses a TCP throughput model given by the equation (3.3).

$$X = \frac{s}{(RTT \cdot \sqrt{\frac{p \cdot 2}{3}} + RTO \cdot \sqrt{\frac{p \cdot 27}{8}} \cdot p \cdot (1 + 32 \cdot p^2))}$$

The sending rate (X) depends on the packet loss rate (p), the mean packet size (s) and the Round Trip Time. RTO refers to the TCP retransmission timeout value.

During the initialization phase, TFRC acts like the TCP slow start algorithm. This slow start phase can also occur during the transfer if the RTO timeout expires [HFPW03]. This phase is followed by a congestion avoidance phase as soon as the receiver detects a loss. In order to compute the sending rate X, TFRC needs an estimation of the current loss event rate. This estimation is achieved using a loss history structure which records the number of packets between successive loss events also called a loss-event interval. From this structure, the loss rate can be computed, as described in [HFPW03].

When the slow start phase is over, the loss history has to be initialized. The number of packets transmitted during the slow start phase cannot be used to estimate the loss event rate since it does not reflect accurantly enough the underlying packet drop rate of the connection [Wid00]. For this reason, existing TFRC implementations use a simple binary search process in which the receiver measures the receiving rate ($X_{measured}$) corresponding to the rate when the first loss occurs and then starts the estimation of the corresponding loss event rate. This estimation is performed by computing the packet loss rate that should have allowed the sender to transmit at the rate $X_{measured}$ using (3.3). We show in the following that the loss history's initialization can be improved with the use of a numerical analysis based on Newton's algorithm of the TFRC equation.

Newton's algorithm rate estimation method

The binary search algorithm is well known for its easily programmable properties but also for its slow convergence. Usually, numerical problems are solved more efficiently using more elaborated algorithms, such as the gradient method, Newton's algorithm or primal-dual method. Because of the relatively simple equation used by TFRC, we propose to set a Newton's algorithm in order to estimate the packet loss rate. To compute this algorithm, during the loss history initialization phase, (3.3) has to be used as a simple function of p as follows:

$$F(p) = RTT \cdot \sqrt{\frac{p \cdot 2}{3}} + RTO \cdot \sqrt{\frac{p \cdot 27}{8}} \cdot p \cdot (1 + 32 \cdot p^2) - \frac{s}{X_{measured}} \tag{6.1}$$

Then, the problem becomes to solve $F(p) = 0$. (6.1) is a algebraic function which can be derived as follows:

$$F'(p) = \frac{RTT \cdot \sqrt{\frac{2}{3}}}{2 \cdot \sqrt{p}} + RTO \cdot \sqrt{\frac{27}{8}} \cdot (1.5 \cdot \sqrt{p} + 112 \cdot p^{\frac{5}{2}}) \tag{6.2}$$

Newton's algorithm is known to converge with a quadratic pace to the solution compared to a linear pace for the binary search algorithm [Gau97]. Thanks to (6.2), we propose to use Newton's algorithm starting with an under solution of p for the first iteration. Newton's iterative process is performed while the convergence criterion is not reached and is computed as follows:

$$p_{i+1} = p_i - \frac{F(p_i)}{F'(p_i)} \tag{6.3}$$

Newton's algorithm is constrained by the existence of $F(p_i)$ and $F'(p_i)$ for all $p_i \in [a, b]$, where $[a, b]$ is the study interval with $a = 0$ and $b = +\infty$. This constraint leads us to exclude values $p_i \leq 0$. In our method, we claim that we have to start the process with the value $a = 10^{-7}$. This value comes from the analysis of the function $F(p)$. Indeed, $F(p)$ has a double concavity. This double concavity can result in a negative value for the next iteration of the algorithm. Nevertheless, this double concavity has a stationary point for $p_0 = 2.84 * 10^{-2}$, for all RTT, s, and $X_{measured}$ under the hypothesis $RTO = 4RTT$ [HFPW03]. In an obvious way, the stationary point is no longer identical for all RTT, s, and $X_{measured}$ if $RTO \neq 4RTT$. According to this stationary point and the restriction $p > 0$ due to the square root, we have to take a starting point inferior to p_0. Thanks to the convergence property of Newton's algorithm [Gau97], we know that it will find the result from any starting point (Theorem 2.7 p 37 for [BFR81]).

In order to compare the two methods, we introduce the following convergence criterion:

$$\left| \frac{X_{measured} - X_{computed}}{X_{measured}} \right| \leqslant \alpha \tag{6.4}$$

where $X_{measured}$ is the rate measured by the receiver, $X_{computed}$ is the rate computed with (3.3) and α is the computation accuracy criterion (also called stopping criterion). According to [HFPW03], this accuracy is recommended to be at least 5%, meaning that when $X_{computed}$ equals to $X_{measured}$ more or less 5% the process stops. The 5% tolerance is introduced for two main reasons: firstly (3.3) is difficult to invert, and secondly the numerical computation cost of p [HFPW03]. The next section will evaluate the convergence pace of both algorithms.

6.2.2 Numerical results and interpretation

Numerical results

This section presents the numerical results of our proposed loss rate computation method compared to the classical binary search method. We have implemented these two algorithms in C++ and evaluated analytically the number of iterations to compute p for given X and RTT values. All the computations have been performed on a Pentium IV processor. The study is performed over a large scale of bandwidth and delay values, with an RTT ranging from $1ms$ to $1000ms$ and a bandwidth ranging from $1KB/s$ to $100MB/s$.

Table 6.1 shows the summary of the number of iterations required to reach the 5% accuracy in this context.

	average	min	max	standard deviation
Binary search	21.7379	1	29	4.1656998
Newton	5.00034	3	13	0.0688248

Table 6.1 Summary of the number of iterations for both algorithms

The results concerning the binary search are expected as this method tends to converge with a number of iterations n [Gau97] with:

$$n = \left\lceil log_2(\frac{|d-c|}{\alpha}) \right\rceil \tag{6.5}$$

where $[a, b]$ is the study interval with $c = 0$ and $d = 1$, α is the convergence criterion and $\lceil x \rceil$ denotes the ceiling of x (i.e., the smaller integer $\geq x$). Nevertheless in (6.5), the convergence criterion α is supposed to be function of the iterative parameter (in our case p). In this study the convergence criterion is function of $X_{computed}$. In order to explain the number of iterations needed by the binary search, we have to use the equation (6.6).

$$n = \left\lceil log_2(\frac{|d-c|}{\alpha'}) \right\rceil \tag{6.6}$$

where α' is the equivalent convergence criterion on p for $\alpha = 0.05$. For the same reason, as it is impossible to solve directly the equation, the translation from α to α' cannot be done with a simple function. In order to illustrate this translation, we give two examples as shown in Table 6.2.

RTT	$X_{measured}$	s	α'
$100ms$	$1Mbit/s$	$8Kbits$	0.0006 ($n = 11$)
$400ms$	$2Mbit/s$	$8Kbits$	0.000015 ($n = 17$)

Table 6.2 Example of α' value

Our large range of numerical experiments tends to show that there is a correlation between RTT, $X_{measured}$ and the new accuracy on p. Indeed, in our experiments, the number of iterations needed to compute the binary search increase with the RTT and $X_{measured}$.

The results concerning Newton's algorithm are also linked to the properties of this algorithm. Indeed, Newton's algorithm converges monotonically from any starting point [Gau97]. Nevertheless, there is no general inferior or superior boundary for Newton's algorithm.

Obviously, the number of iterations is linked to the computation time needed by both methods. However, even if Newton's algorithm requires a stable number of iterations, we need to verify if it is more computationally efficient. In the next section, we therefore compare the computation time of both algorithms. We focus on the range where the binary search algorithm is efficient.

Interpretation and discussion

We define the efficiency criterion ϵ as the ratio of the binary search algorithm computation time (t_{dicho}) to Newton's algorithm computation time (t_{Newton}):

$$\epsilon = \frac{t_{dicho}}{t_{Newton}} \tag{6.7}$$

The results are shown in Figure 6.1. When the efficiency criterion is lower than one it corresponds to a better efficiency of the binary search algorithm.

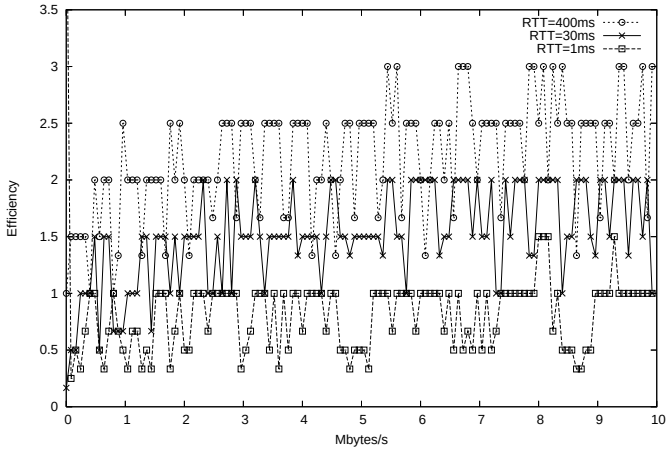


Figure 6.1 Comparison of binary search and Newton's algorithm efficiency

In Figure 6.1, we represent a sample of the study case summarised in Table 6.1. We focus on the smaller range of bandwidth than in Table 6.1 as this range is less favorable to Newton's algorithm. In total study range summarised in Table 6.1, our study shows that Newton's algorithm is more efficient for more than 95% of the cases, as efficient for less than 4% and less efficient for less than 1% of the cases.

In addition, we propose to study the correlation between the number of iterations and the number of CPU cycles needed to reach the 5% of accuracy. In order to explore the way these two characteristics interact, we proceed to an algorithmic study of both algorithms. We have presented in Table 5.6 the number of elementary operations in both algorithms and the theoretical number of CPU cycles for every operation[1].

Usually, the division and the square root need the highest number of CPU cycles. We see that Newton's algorithm needs more elementary operations per iteration. But as shown previously, it also needs less iteration for a given accuracy. Next we study these two algorithms for the worst case. For the binary search, the worst case is when the result of the computation is outside the range $[0, 1]$[2]. In this case, the maximum number of iterations should be fixed *a-priori* as a static variable. According to the results obtained and summarized in Table 6.1, we fixed this variable to 30 iterations. The Newton's worst case has been evaluated to 13 iterations by our computation scheme. In these conditions the worst case study results are presented in Table 6.3.

	Iterations	CPU cycles
Binary search	30	10222
Newton	13	6402.5

Table 6.3 Worst case study

We show that the number of cycles needed in both cases is largely in favor of Newton's algorithm.

Side effect of the Newton Algorithm

One other property of the Newton algorithm is the accuracy improvement toward the solution for each iteration. In order to study this property let's define the efficiency criterion δ as the ratio between the dichotomy algorithm computation accuracy (δ_{dicho}) to the Newton algorithm computation accuracy (δ_{Newton}) such as $\delta = \frac{\delta_{dicho}}{\delta_{Newton}}$. As a result, if δ is lower than 1 it means that the binary search is more efficient the Newton algorithm. The result for large range of RTT and bandwidth value are shown in Figure 6.2.

Figure 6.2 shows an important difference in favor of the Newton algorithm in terms of precision. This difference can be explained by speed of convergence of this algorithm. This figure shows that the Newton algorithm is more precise in 99.5% of cases for the same stopping criterion.

Although the TFRC Request For Comments [HFPW03] recommends a computation with a precision of at least 5%, we show that the Newton algorithm can reach a higher precision. This higher accuracy in the sending rate computation is expected to impact the effective throughput of the transmission significantly. For example, as the sending rate more accurately reflects the bandwidth available to the flow, the chances of the sending process

[1] According to Intel PIV documentation.

[2] These rare cases have been voluntary excluded from the Table 6.1.

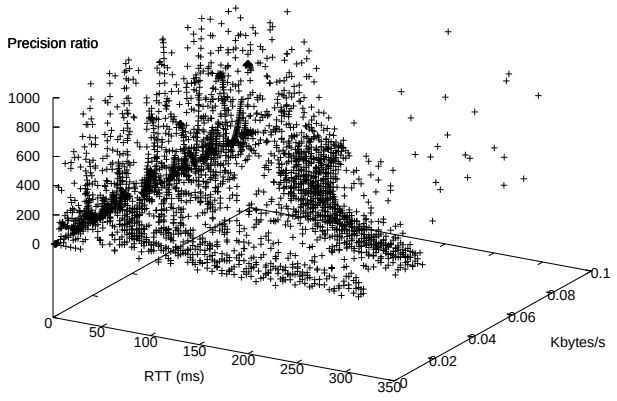


Figure 6.2 Comparison of binary search and Newton's algorithm accuracy

exceeding its fair share (possibly resulting in packet losses), are minimised. We computed this initialization process with a stopping criterion set to 0.1%. In average, the dichotomy requires 90% more time than it would need to reach 5% precision.

6.2.3 Conclusion

In this section, we have presented an efficient method to compute the packet loss rate of the TFRC equation. We showed that this method outperforms the 5% accuracy in terms of number of iterations and computation time. The initial results presented in this section have demonstrated the efficiency of the proposed method. Due to its low computation needs, it is particularly well-suited to mobile devices with low processing power. Furthermore, due to the low standard deviation of the number of iterations, it is also well-suited to real time processing. In future work, presented in the following section, we will evaluate the performance implications of using this mechanism in a TFRC implementation.

6.3 FUTURE WORK: STUDY OF THE LOSS HISTORY

In the previous section we have presented an optimisation of the initialisation of the loss history structure. This optimisation lead us to investigate the impact of the initialisation on the overall behaviour of TFRC. Indeed, we have seen that as a side effect the Newton's algorithm can find the solution with a better accuracy than the recommended 5% without an excessive number of iterations. In order to study the possible correlation between this initialisation (and the value of the stopping criteria) and TFRC behaviour, we propose in this section the first step toward an approach to better understand the importance of the various TFRC's parameters through a model of TFRC receiver.

This work focuses on the structure used to compute the packet loss rate, i.e. the loss history structure. This work follows the notations introduced in [VB05, RX05, XH06]. We consider in this work the comprehensive model of [VB05] and we will adapt this model to a Markov chain.

The study will be organized as follows:

- Proposition of the Markov chain model;
- model of TFRC inside scilab/scicos software;
- presentation of the 3-states Gilbert model we will use;
- conclusion and future work.

This approach aims to help the designer to better understand the behaviour of the receiver and of the streaming in TFRC. Indeed, this model can help to have a state-oriented model of TFRC. In the rest of this section we propose in a first model for the use of the loss history structure, we then present the integration of TFRC inside the scilab/scicos software in which we have integrated different loss models.

6.3.1 A first model of the lost history utilisation

In this section we propose a first model of the loss history structure and its relation with the network behaviour. This model is based on a Markov chains. This network will be configured by the following parameters:

- the probability that a packet is lost in an RTT round,
- the number of packets received since the last loss event $\theta(t)$,
- the actual weight of the $n-1$ interval in the loss history ω_{n-1},
- the value of the n^{th} interval in the loss history $(\theta_n)_n$,
- and t the time since the beginning of the transmission.

The different use of the loss history

In [HFPW03] the estimation of the loss rate is computed regarding a specific structure on which is applied a sliding window weighted average computation scheme. This scheme is:

$$p = \frac{\sum_{i=0}^{n-1} \omega_i}{\max(\sum_{i=0}^{n-1} \omega_i \cdot \theta_{n-i}, \sum_{i=1}^{n} \omega_{i-1} \cdot \theta_{n-i})} \tag{6.8}$$

This scheme can be illustrated by the Figure 6.3.

In this context we have θ_t = number of packets received since the last loss event, $\omega_0 = 1$, θ_n = number of packet received between the $(n+1)^{th}$ loss event and the n^{th} loss event or $\frac{1}{p_{est}}$, and $\omega_n = 1 - \frac{n+1}{n+3}$. The previous notation has been introduced in [VB05] and used in [RX05, XH06].

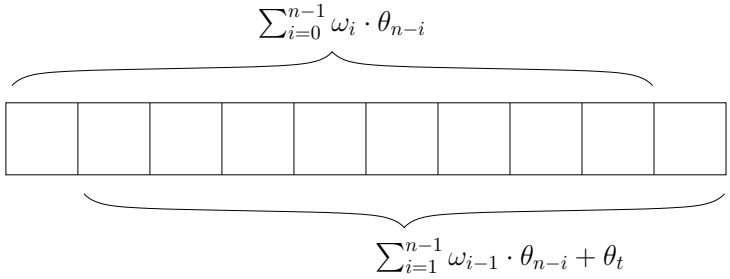


Figure 6.3 Use of the loss history to compute the loss rate

Equivalent Markov Chain

Based on the presentation of the use of the loss history structure depicted in the previous section we were able to produced an equivalent Markov chain. This Markov chain is shown in Figure 6.4.

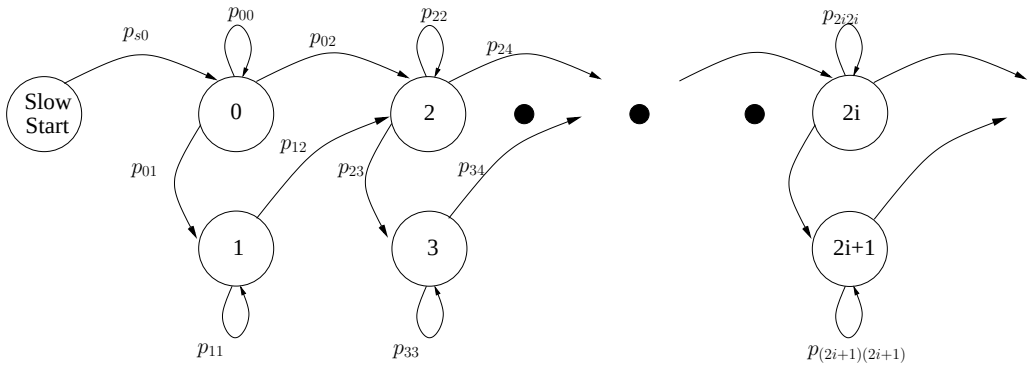


Figure 6.4 Equivalent Markov Chain

In this model there is three particular states:

- the slow start state: this state represents the behaviour of the protocol during its slow start phase. The two other kind of states represent the behaviour of the protocol during the congestion avoidance phase.
- the $(2_i,\ \forall i \in \mathbb{N}^+)$ states represent the behaviour of the protocol when it does not take into account the number of packets arrived since the last loss event;
- the $(2_i + 1,\ \forall i \in \mathbb{N}^+)$ states represent the behaviour of the protocol when it takes into account the number of packets arrived since the last loss event;

The transition matrix of this model is of infinite dimension with the following pattern:

$$\begin{bmatrix}
p & p_{s0} & 0 & & & & & \\
0 & p_{00} & p_{01} & p_{02} & 0 & & & \\
\vdots & 0 & p_{11} & p_{12} & 0 & & & \\
 & \vdots & 0 & \ddots & \ddots & \ddots & & \\
 & & \vdots & 0 & p_{2i2i} & p_{2i(2i+1)} & p_{2i(2i+2)} & 0 \\
 & & & \vdots & 0 & p_{(2i+1)(2i+1)} & p_{(2i+1)(2i+2)} & 0 \\
 & & & & \vdots & 0 & \ddots & \ddots
\end{bmatrix}$$

In our study we consider the following simplifying hypothesis:

***Hypothesis* 6.3.1** *even if a loss is detected the receiver only computes the packet loss rate once an RTT.*

In the previous transition matrix, the transition probabilities can be classified as follows:

$$p_{s0} = \mathbb{P}(\theta_0 > t) \tag{6.9}$$

$$p_{2i(2i+2)} = p_{(2i+1)(2i+2)} = \mathbb{P}(\theta_{i+1} > t) \tag{6.10}$$

where, $p_{2i(2i+2)}$ and $p_{(2i+1)(2i+2)}$ represents the probability of a loss event. Furthermore since the transition matrix that represents the system is infinite and is not positive recurrent (if all probabilities are different from 1 and 0) we know that there is not a stationary distribution associated to it [Bre99].

The weights $\omega_i, i \in [0, N]$ are computed as recommended in [HFPW03]. In [HFPW03], the authors recommend to compute these weights following the algorithm as described in Fig. 6.5.

```
for{i=1; i<=n; i++}{
  if(i>n/2){
    w_i=(2N-2i)/(n+2)
  }else{
    w_i=1
  }
}
```

Figure 6.5 Computation scheme of the different weights in TFRC

Based on this first analysis of the loss history behaviour, we decide to investigate the relation between the proposed Markov chain, the value of the weight applied to the different loss intervals and some probability distributions for the loss pattern probability distributions. To achieve the evaluation of the possible correlation between these three parameters, we have decided to implement the TFRC mechanism inside an adequate tool kit. We have chosen the scilab/scicos toolkit for its modularity and the availability of its open source code

6.3.2 Use of scilab/scicos

In order to study the previous Markov chain, we have chosen an analytical approach since the transition matrix is infinite, therefore we cannot find its stationary distribution.

A model of TFRC and the network

We have modelled the TFRC sender and receiver algorithm inside scicos toolkit. These two blocks are therefore linked with a block representing the network between the receiver and the sender. This general model is represented in the Figure 6.6.

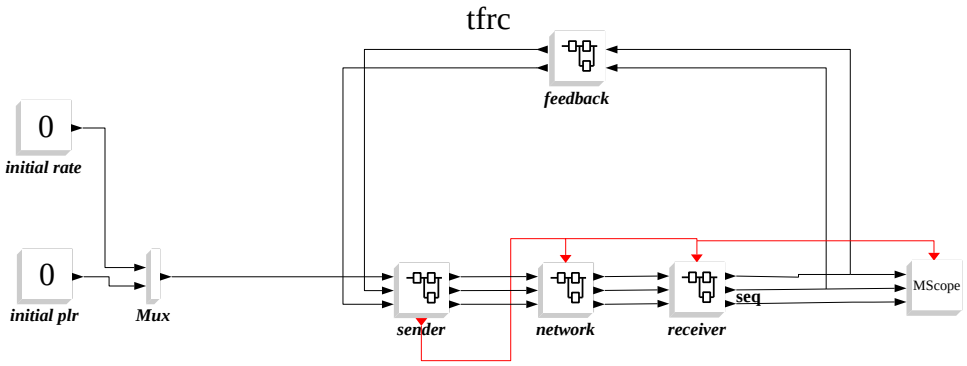


Figure 6.6 General Model of TFRC inside a network

In this model we make one important assumption. This assumption concerns the one way delay between the sender and the receiver. This one way delay is set to $0ms$ in order to better use the possibilities of scicos. Indeed, in scicos we want to use the input and output event to better drive our simulations.

The receiver and sender have also been modelled in scicos. In the receiver model we have included the slow start phase as we want to study the impact of the initialisation of the receiver.

In this model the network is modelled with a simple function that looses packets. These losses can be driven via two methods. In the first one we simply loose packets according to measurements obtained either with ns 2 or real test bed. In the second one, we loose packet following some probability models. We have implemented three kinds of probability of losses:

- simple Gilbert model;
- multiple states Gilbert model;
- Bernoulli model.

We present in the following section a 3-states Gilbert model

Multiple states Gilbert model

In order to clarify this model we will first study the four states Gilbert model. This model is a Markov chain with transition matrix M:

$$\begin{bmatrix} 1-p_1 & p_1 & 0 & 0 \\ 1-p_2 & 0 & p_2 & 0 \\ 1-p_3 & 0 & 0 & p_3 \\ 1-p_4 & 0 & 0 & p_4 \end{bmatrix}$$

In this model the state 1 represents the network when you just had a packet lost. The state 2 represents the successful transmission of one packet. The state 3 represents the successful transmission of one more packet. Finally the state 4 represents the successful transmission of more than three packets and continuing sending packets without losses. In this model there is only one class except when $p_4 = 1$ and the other probabilities are different to zero, and obviously when one probability equal to zero there is more than one class. Therefore the class is irreducible and all the states are transient. A simple study of this model can provide us a stationary distribution $\pi = [\pi_1\ \pi_2\ \pi_3\ \pi_4]$ where

$$\begin{cases} \pi_1 &= \frac{(1-p_4)}{\alpha} \\ \pi_2 &= \frac{(p_1-p_1p_4)}{\alpha} \\ \pi_3 &= \frac{(p_1p_2-p_1p_2p_4)}{\alpha} \\ \pi_4 &= \frac{(p_1p_2p_3)}{\alpha} \end{cases}$$

Where $\alpha = 1 - p_4 + p_1 - p_1p_4 + p_1p_2 - p_1p_2p_4 + p_1p_2p_3$. In this Markov model, the chain is irreducible and admits a stationary distribution then we know that every π_i for $i \in \{1, 2, 3, 4\}$ is equal to the inverse of the average return time of the i^{th} state [Bre99].

6.3.3 Preliminary conclusion

We have implemented the Gilbert model inside the network model of scicos. We are currently studying the impact of the different parameters on TFRC flow. The introduction of such probabilities distribution should allow us to compare the theoretical result and the packet loss rate computed as described before. Indeed, the use of scilab allows us to compute the Euclidean distance between the computed packet loss rate and the theoretical value. Based on this distance we should be able therefore to adjust the value of the weight (or the way to compute them) and repeat the experiment until the distance between these values is minimised.

We have presented in this section a first approach to study the behaviour of TFRC under numerous probability distributions. Nevertheless, this first approach did not allow us to produce any significant results yet, but thanks to the ease of use scilab and scicos toolkit, we think that we will be able to propose in the future a more complete study that could quantify the impact of loss history initialisation and also according to well known stochastic distribution optimise the computation of the weight applied to the computation of the packet loss rate.

6.4 CONCLUSION

In this chapter we have presented one contribution and a promising future work. We have shown that based on a numerical analysis of the equation used in TFRC mechanism we can lighten and enhance the initialisation of the loss history structure. This solution is based on the use of the well known Newton algorithm for solving the inversion problem of the TFRC equation.

We have quantified the benefit of this algorithm compared to the usually used binary search algorithm and we have shown that our proposal, even if it required more elementary operations per iterations, outperforms the original algorithm in terms of computation time and CPU consumption. Furthermore, the proposed method can invert the TCP throughput equation with a better accuracy and without loss of performance. This is due to the better convergence pace of the Newton algorithm. We plan to integrate this new algorithm inside the Chameleon Protocol in order to quantify the potential benefit of this method on the communication process.

In the second part of this chapter, we described an in-progress approach that aims to analyse the correlation between the use of the loss history, the applied weights and a defined loss probability model. We have first proposed a Markov chain representing the loss history behaviour and its associated transition matrix. Due to the complexity introduced by this model we have decided to analyse it through the integration of the TFRC mechanism inside a discrete event toolkit.

In a future work we plan to pursue the use of this model in order to try to answer the following questions:

- how to quantify the impact of probability models on the way the packet loss rate is computed in TFRC?
- Is it possible to optimize the weights associated to the loss interval, in order to be closer to the probability distribution?
- If this is possible, how can we distinguished between wired and wireless losses?

Chapter 7

Conclusion

This chapter gives an overview of the contributions of this thesis and some directions on future research that would benefit from this thesis.

7.1 PROBLEMS SUMMARY

In this thesis, we try solve two problems observed in current transport protocols thanks to respectively the specialisation and the adaptation of the congestion control TCP Friendly Rate Control (TFRC). These problems are:

1. the poor performances of current transport protocols over bandwidth-guaranteed network;
2. CPU and memory consumption of transport protocols when used on entities with limited resources.

In addition to these two general problems we identified and corrected some drawbacks in the current architecture of the TFRC. This resolution has been done with the optimisation of TFRC internal algorithms.

7.1.1 Transport protocols performance over bandwidth guaranteed networks

In chapters 2, 3 and 4, we have presented the limitations of current transport protocols to fully benefit from QoS-oriented network-layer services. We mainly focus on bandwidth-guaranteed networks and in particular the DiffServ/AF class. In these networks, as shown in previous work [SNP99, PC04a], TCP cannot reach the negotiated bandwidth for all network conditions and it is not fairly sharing the out-profile bandwidth. The only way

to ensure TCP to reach its target rate is to introduce complex conditioners at the edge of the network. Nevertheless, because these conditioners and TCP are not using the same measures of the network behaviour it still remains some cases where TCP cannot reach its target rate.

The introduction of new transport protocols and new congestion controls did not solve this incompatibility between the QoS networks and the transport layer. Indeed, in SCTP [SA00] and DCCP/CCID 2 [FK06], the congestion control is similar to the one used by TCP. In DCCP/CCID 3 [FKP06], the congestion control in use is TFRC [HFPW03]. This congestion control uses a model of TCP AIMD congestion control, as a result this kind of congestion control does not perform better than TCP over a QoS network [KK03]. Nevertheless, TFRC and rate-based congestion controls, contrary to TCP, are sending packets according to the inter-packets time which constitutes a metric at the edge routers.

7.1.2 Resources limited entities

The last five years have seen more and more small communication entities emerge. These entities are characterised by less CPU power and memory storage than usual personal computers. Therefore the lightening of recurrent communication processing is a critical issue for increasing the performance and autonomy of mobile end systems. One of the main costs of the TFRC mechanism comes from the periodic computation of both the RTT and the loss rate of data carried by a connection. In particular, RFC 3448 [HFPW03] proposes the loss rate estimation to be done on the receiver side. It also suggests that this computation could also be done on the sender-side: *"It would be possible to implement a sender based variant of TFRC where the receiver uses reliable delivery to send information about packet losses and the sender computes the packet loss rate and the acceptable transmit rate".*

7.1.3 Drawbacks of TFRC implementations

Throughout this thesis we tried to solve different problems related to TFRC congestion control. These studies led us to discovery some computational possible improvements inside this mechanism. The first one was the computation of the loss history initialisation. Indeed, in every implementation we analysed, this particular function was done with a binary search algorithm which is the simplest algorithm to implement but also poorly efficient one. Furthermore, this algorithm cannot certify the resolution of the problem if the solution is outside the initial boundaries.

The second problem we observed was the computation of the packet loss rate. Indeed, this computation is done using a weighting moving window average of the loss event intervals. Nevertheless, the weight have been defined in the case of wired network therefore, nowadays these weights may not be adapted to the different new network technology, especially the wireless networks.

7.2 CONTRIBUTION SUMMARY

7.2.1 Design and implementation of QoS-aware Transport Protocol

In this thesis we have designed and implemented a transport protocol able to reach the negotiated bandwidth whatever the network conditions. This successful design is based on the introduction of a new congestion control mechanism especially designed for QoS-oriented network services. We called this mechanism *g*TFRC (for guaranteed TFRC) since it consists of a specialisation of TFRC congestion control mechanism in order to make it aware of the target rate. We have demonstrated the efficiency of this mechanism through its implementation inside a Java framework and its evaluation over a DiffServ/AF testbed. The results show that this mechanism allowes the transport protocol to reach its target rate whatever the network conditions. Nevertheless, this mechanism still does not allow the transport protocol to share fairly the out-profile traffic.

After the evaluation of this congestion control, we compose it with a reliable mechanism in order to provide a transport service similar to TCP. This reliable service is provided by a SACK-like mechanism and the introduction of flow control especially designed for datagram-oriented and rate-based transport protocol. The results of this composition have confirmed the previous results and have shown that, at the application level, the results remain similar. Furthermore, we have shown that this mechanism is not only efficient with a DiffServ/AF class using a RIO mechanism but also with on top of a generic class-based network service.

7.2.2 Lightened version of TFRC

Our second contribution has focused on the design and implementation an adaptation of TFRC for mobile and light end-system via the introduction of a new version of a sender-based architecture. This TFRC architecture has aimed to provide a light version of TFRC and to solve the problem of the selfish receiver. This proposal is based SACK-like information to inform the sender about lost packets and therefore has introduced a new sender-oriented definition of loss events. Indeed, in this new version a loss event is no longer detected according to the estimated arrival time of the lost packet but according to the sending time of this lost packet corrected by a ratio factor between the sending and receiving rate.

The proposed solution has been evaluated regarding numerous metrics, confirming that it stays compatible both with TCP and the original TFRC. Furthermore, we have quantified this proposal and shown that it allows light entities to save a substantial number of CPU cycles, memory space and therefore increases substantially their autonomy.

7.2.3 Optimisation and tool box for TFRC

Finally we have introduced an additional contribution and sketche a promising future work. The contribution has targetted the optimisation of the loss history initialisation. This optimisation uses a Newton algorithm based on the numerical analysis of the equation used in TFRC. This proposal allows the receiver to invert the equation with a stable

number of iterations and less CPU cycles than the currently used algorithm. Furthermore, this algorithm allows a better accuracy in the resolution of the inversion of the equation without using many more iterations.

Based on this first optimisation, we investigated a way to better quantify how TFRC loss event rate is computed and how to optimise it. This study can be done due to the introduction of a tool box that reproduces the behaviour of TFRC and where probability model for the losses can be easily integrated.

7.3 FUTURE RESEARCH DIRECTIONS

During this thesis, we have shown the efficiency of the combination of a rate-based congestion control and a SACK-like mechanism that integrates a flow control.

In this thesis, we have built transport protocol mechanisms that aim to improve the QoS perceived by the user, as well as the autonomy of light entities. These new mechanisms are merely elementary components of a versatile transport protocol and they demonstrate the pressing need for a transport protocol that can be simultaneously adapted to the application and underlying network services. Indeed, facing the great diversity of access-network features and application requirements, it seems impossible to define an universal protocol that would encompass the entire combination of these features. Therefore, service-oriented transport protocols, able to be dynamically configured, appear to be the future of new generation communication architectures. These new approaches dispose of the traditional layered approach and raise numerous issues, such as security, performance, service identification, localisation and composition, and the reliability and consistency of composed services. In a future work, we plan to define a global architecture for dynamic protocol composition.

Recent studies have shown that a rate-based congestion-control mechanism should improve the throughput for multi-hop ad-hoc networks, in particular for Vehicular Ad-Hoc Network (VANET). The studies presented in this paper provide substantial evidence for the potential benefits of such a composition. In future, we plan to specify the entire chameleon protocol and its different versions in order to implement it at the kernel level, since we have observed certain limitations in terms of throughput when the prototype was used at the application level. This implementation should allow us to test the properties of such a protocol over a vehicular network testbed.

This implementation at the kernel level will integrate all the different versions of the chameleon protocol as synthesised in the Table 7.1

	reliable	non-reliable
TFRC normal	Yes	Yes
QoS-aware	Yes	Yes
Sender-based	Yes	Yes

Table 7.1 Different configuration of Chameleon

In all these versions of Chameleon Protocol, the optimisations proposed in chapter 6 will be integrated.

Based on the work done with the Scilab-Scicos software, we plan to propose more complete studies of the computation of the loss event rate in TFRC. Future research will attempt to better understand the problems raised in [VB05, RX05]. In the first instance, future research will use the same hypothesis concerning the constant value of the RTT. However, as demonstrated in [ABR05, BCD+06] this hypothesis is no longer valid when the studied network contains a wireless access network. Therefore, we will integrate a non-constant RTT in order to better study TFRC mechanism.

In the future, based on the findings of this thesis, we will attempt to adapt the transport layer for global computing P2P application. Indeed, we have shown in [JE05] that this kind of application, with the use of asynchronous algorithms, allows an improvement for the pace of convergence of the algorithm. Furthermore, the use of asynchronous algorithms ensures a certain robustness and the convergence of the resolution [BG97]. A study of this kind of algorithm using a transport protocol, whether able or not to be configured to provide reliability, should improve P2P global computing applications.

In order to integrate such kinds of transport protocols inside a global computing application, some frameworks allow the study of these applications over an heterogeneous network [Bou05]. One possible study could be the optimisation of the sending process according to the network estimations and the class of asynchronous algorithm used by the application (e.g. gradient, Gauss-Siedel).

List of Publications

[1] Ernesto Exposito, Guillaume Jourjon, Emmanuel Lochin, and Nicolas Van Wambeke. Enhanced transport protocols. In Fernando Boavida, Torsten Braun, Michel Diaz, Jose Enrique, and Thomas Staub, editors, *End-to-End Quality of Service Over Heterogeneous Networks*. 2008.

[2] Design, Implementation and Evaluation of a QoS-aware Transport Protocol, Guillaume Jourjon, Emmanuel Lochin, Patrick Sénac *Accepted for publication in Elsevier Computer Communications 2008*

[3] Optimization of Loss History Initialization, Guillaume Jourjon, Emmanuel Lochin, Laurent Dairaine *IEEE Communication Letters - Volume 11, Issue 3 - March, 2007, pp 276-278.*

[4] IREEL: Remote Experimentation with Real Protocols and Applications over Emulated Network, Laurent Dairaine, Guillaume Jourjon, Emmanuel Lochin and Sébastien Ardon, *Inroads, the SIGCSE Bulletin, Volume 39, Issue 2, June 2007, pp 92-96*

[5] gTFRC, a TCP Friendly QoS-aware Rate Control for Diffserv Assured Service, Emmanuel Lochin, Laurent Dairaine, Guillaume Jourjon, *Springer Telecommunication Systems Journal, 10.1007/s11235-006-9004-2, ISSN : 1018-4864 (Print) 1572-9451 (Online), Volume 33, Numbers 1-3 / December, 2006, pp 3-21.*

[6] Towards sender-based TFRC, Guillaume Jourjon, Emmanuel Lochin and Patrick Sénac, *IEEE International Conference on Communications, Glasgow, 24-28 June 2007, pp 1588-1593.* **Best paper award of the Multimedia Communications and Home Services Symposium of ICC 2007**

[7] Study and enhancement of DCCP over DiffServ Assured Forwarding class, Emmanuel Lochin, Guillaume Jourjon and Laurent Dairaine, *In Proceedings of the Fourth European Conference on Universal Multiservice Networks (ECUMN'07), pp. 250-262.*

[8] Towards a Versatile Transport Protocol, Guillaume Jourjon and Emmanuel Lochin and Patrick Sénac, *Student workshop of CONext06 in cooperation with ACM Sigcomm, December 2006.*

[9] Implementation and performance analysis of a QoS-aware TFRC mechanism, Guillaume Jourjon, Emmanuel Lochin, Laurent Dairaine, Patrick Sénac, Tim Moors and Aruna Seneviratne, *Proceedings of 14th IEEE ICON 2006 (International Conference on Networking), October 2006.*

[10] gTFRC: a QoS-aware congestion control algorithm, Emmanuel Lochin, Laurent Dairaine and Guillaume Jourjon, *Proc. of the 5th International Conference of Networking, 2006.*

[11] IREEL: Remote Experimentation with Real Protocols and Applications over Emulated Network, Laurent Dairaine, Guillaume Jourjon, Ernesto Exposito, *Poster Session of ACM ITiCSE 2006 June 2006.*

[12] Modeling, Simulation, and Emulation of QoS Oriented Transport Mechanisms, Guillaume Jourjon, Ernesto Exposito and Laurent Dairaine, *Student workshop of CONext05 in cooperation with ACM Sigcomm, October 2005.*

[13] Some Solutions for Peer-to-Peer Global Computing, Guillaume Jourjon and Didier El Baz, *Proceeding of the PDP '05: Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'05), pp 49-58.*

Bibliography

[ABR05] Eitan Altman, Chadi Barakat, and Victor M. Ramos. Analysis of AIMD protocols over paths with variable delay. *Computer Networks*, 48(6):960–971, August 2005.

[AHA97] O. Ait-Hellal and E. Altman. Analysis of tcp-vegas and tcp-reno. In *IEEE International Conference on Communications (ICC 97)*, pages 495–499, 1997.

[AP03] G. Anastasia and A. Passarella. Towards a Novel Transport Protocol for Ad Hoc. In *Personal Wireless Communications (PWC)*, September 2003.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Request For Comments 2475, IETF, December 1998.

[BCB06] J. Babiarz, K. Chan, and F. Baker. Configuration Guidelines for DiffServ Service Classes. Request For Comments 4594, IETF, August 2006.

[BCD+06] G. Boggia, P. Camarda, A. D'Alconzo, L. A. Grieco, S. Mascolo, E. Altman, and C. Barakat. Modeling the AIADD Paradigm in Networks with Variable Delays. In *Proceedings of Conext* , December 2006.

[BCS94] R. Braden., D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. Request For Comments 1633, IETF, June 1994.

[BFR81] Richard L. Burden, J. Douglas Faires, and Albert C. Reynolds. *Numerical Analysis*. Pindle, Weber and Schmidt, Boston, Mass., 1981.

[BG92] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1992.

[BG97] D. Bertsekas and R. Gallager. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[Bou05] Julien Bourgeois. *Réseau : Transport, Modélisation et Optimisation*. PhD thesis, University of Franche-Comté, 2005.

[BP95] L. S. Brakno and L. L. Peterson. TCP Vegas: End to End congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, (13):1465–1480, 1995.

[Bre99] Pierre Bremaud. *Markov Chain, Gibbs Field, Monte Carlo Simulation and Queues*. Springer-Verlag, New York, 1999.

[Bri06] Bob Briscoe. Flow Rate Fairness: Dismantling a Religion. Internal report, ACM SIGCOMM CCR, October 2006.

[Cho99] Kenjiro Cho. Managing traffic with ALTQ. *Proceedings of USENIX Annual Technical Conference: FREENIX Track*, June 1999.

[CM05] XiaoLin Chang and Jogesh K. Muppala. On improving bandwidth assurance in af-based diffserv networks using a control theoric approach. *Computer Networks*, 49(6):816–839, March 2005.

[CMX+05] C. Cicconetti, M. Garcia-Osma, X. Masip, J. Sa Silva, G. Santoro, G. Stea, and H. Taraskiuk. Simulation model for end-to-end QoS across heterogeneous networks. In *Proc. of International Workshop on Internet Performance, Simulation, Monitoring and Measurement (IPS-MoMe)*, 2005.

[CNV04] K. Chen, K. Nahrstedt, and N. Vaidya. The Utility of Explicit Rate-Based Flow Control in Mobile Ad Hoc Networks. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, March 2004.

[CT90] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM '90: Proceedings of the ACM symposium on Communications architectures & protocols*, pages 200–208, New York, NY, USA, 1990. ACM.

[Del00] Christophe Deleuze. *Qualité de Service dans l'Internet : problèmes liés au haut débit et au facteur d'échelle*. PhD thesis, Université Pierre et Marie Curie, January 2000.

[Egg07] Lars Eggert. Experimental Specification of New Congestion Control. Internet Draft ion-tsv-alt-cc.txt, IETF, May 2007.

[EGS02] M.A. El-Gendy and K.G. Shin. Assured Forwarding Fairness Using Equation-Based Packet Marking and Packet Separation. *Computer Networks*, 41(4):435–450, 2002.

[ELD07] Guillaume Jourjon Emmanuel Lochin and Laurent Dairaine. Study and enhancement of DCCP over DiffServ Assured Forwarding class. In *In Proceedings of the Fourth European Conference on Universal Multiservice Networks (ECUMN'07)*, pages 250–262, April 2007.

[EMR+03] Ernesto Exposito, Mathieu Gineste, Romain Peyrichou, Patrick Sénac, and Michel Diaz. XQOS: XML-based QoS Specification Language. In *Proc. of International Multimedia Modelling Conference (MMM)*, 2003.

[EPM04] Ernesto Exposito, Patrick Sénac, and Michel Diaz. UML-SDL modelling of the FPTP QoS oriented transport protocol. In *Proc. of International Multimedia Modelling Conference (MMM)*, 2004.

[Exp03] Ernesto Exposito. *Specification and Implementation of a QoS Oriented Transport Protocol for Multimedia Applications.* PhD Thesis, LAAS-CNRS/ENSICA, December 2003.

[FA07] Sally Floyd and Mark Allman. Specifying New Congestion Control Algorithms. Internet Draft draft-ietf-tsvwg-cc-alt-04.txt, IETF, June 2007.

[FF96] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, July 1996.

[FF99] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[FHG04] S. Floyd, T. Henderson, and A. Gurtov. The newreno modification to tcp's fast recovery algorithm. Request For Comments 3782, IETF, April 2004.

[FHPW00] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. Equation-based Congestion Control for Unicast Applications. In *Proc. of ACM SIGCOMM*, pages 43–56, Stockholm, Sweden, August 2000.

[FJ95] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[FK06] Sally Floyd and Eddie Kohler. Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control. Request For Comments 4341, IETF, March 2006.

[FKP06] Sally Floyd, Eddie Kohler, and Jitendra Padhye. Profile for DCCP Congestion Control ID 3: TRFC Congestion Control. Request For Comments 4342, IETF, March 2006.

[FKSS98] W. Feng, Dilip Kandlur, Debanjan Saha, and Kang S. Shin. Adaptive Packet Marking for Providing Differentiated Services in the Internet. In *Proc. of the IEEE International Conference on Network Protocols - ICNP*, October 1998.

[FL03] C. P. Fu and S. C. Liew. Tcp veno: Tcp enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications*, 21(2):216–229, February 2003.

[Flo03] S. Floyd. Highspeed tcp for large congestion windows. Request For Comments 3649, IETF, December 2003.

[FMMP00] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An extension to the selective acknowledgement (SACK) option for TCP. Request For Comments 2883, IETF, July 2000.

[For07] Bryan Ford. Structured stream: a new transport abstraction. In *Proc. of SIGCOMM 07*, Kyoto, August 2007.

[FRK00] Azeem Feroz, Amit Rao, and Shivkumar Kalyanaraman. A TCP-Friendly Traffic Marker for IP Differentiated Services. In *Proc. of IEEE/IFIP International Workshop on Quality of Service - IWQoS*, June 2000.

[FSa00] W. Fang, N. Seddigh, and al. A time sliding window three colour marker. Request For Comments 2859, IETF, June 2000.

[Gau97] Walter Gautschi. *Numerical Analysis An Introduction*. Birkhäuser Boston, 1997.

[GG05] M. Georg and S. Gorinsky. Protecting tfrc from a selfish receiver. In *Proc. of Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS 2005)*, October 2005.

[GG07] Yunhong Gu and Robert Grossman. Udt: Udp-based data transfer for high speed wide area networks. *Elsevier Computer Network*, 51(7), August 2007.

[GSW+01] M. Gerla, M. Y. Sanadidi, Ren Wang, A. Zanella, C. Casetti, and S. Mascolo. TCP Westwood: congestion window control using bandwidth estimation. In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, pages 1698–1702, 2001.

[HBF02] Ahsan Habib, Bharat Bhargava, and Sonia Fahmy. A Round Trip Time and Time-out Aware Traffic Conditioner for Differentiated Services Networks. In *Proc. of the IEEE International Conference on Communications - ICC*, New-York, USA, April 2002.

[HBWW99] Juha Heinanen, Fred Baker, Walter Weiss, and John Wroclawski. Assured forwarding PHB group. Request For Comments 2597, IETF, June 1999.

[HFPW03] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control TFRC: Protocol Specification. Request For Comments 3448, IETF, January 2003.

[HG99] J. Heinanen and R. Guerin. A single rate three color marker. Request For Comments 2697, IETF, September 1999.

[HJ98] M. Handley and V. Jacobson. Sdp: Session description protocol. Request For Comments 2327, IETF, April 1998.

[HK00] S. Hassan and M. Kara. Simulation-based performance comparison of tcp-friendly congestion control protocols. In *Proceedings of the 16th Annual UK Performance Engineering Workshop (UKPEW2000)*, pages 199–210, 2000.

[HKLdB05] R. Hancock, G. Karagiannis, J. Loughney, and S. Van den Bosch. Next steps in signaling (nsis): Framework. Request For Comments 4080, IETF, June 2005.

[HP91] Hutchinson and Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions Software Engineering*, 17 (1), 1991.

[IT96] ITU-T. End-user multimedia qos categories. Technical Report G.1010, ITU, 1996.

[IT04] ITU-T. Framework recommendation for multimedia services. Technical Report F.700, ITU, 2004.

[Jac88] Van Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM*, pages 314–329, Stanford, CA, August 1988.

[Jai91] R. Jain. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling.* Wiley-Interscience, 1991.

[JB88] V. Jacobson and R. Braden. Tcp extensions for long-delay paths. Request For Comments 1072, IETF, 1988.

[JBB92] V. Jacobson, R. Braden, and D. Borman. Tcp extensions for high performance. Request For Comments 1323, IETF, 1992.

[JE05] Guillaume Jourjon and Didier El Baz. Some solutions for peer-to-peer global computing. In *Proceeding of the PDP '05: Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'05)*, Italy, February 2005.

[JPBF94] Van Jacobson, L. Peterson, L. Brakmo, and S. Floyd. Problems with arizona's vegas. Technical report, mailing list, end2end-tf, 1994.

[JWL04] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: motivation, architecture, algorithms, performance. In *IEEE Infocom '04*, Hongkong, 2004.

[KAJ01] K. Kumar, A. Ananda, and L. Jacob. A memory based approach for a tcp-friendly traffic conditioner in diffserv networks. In *Proc. of the IEEE International Conference on Network Protocols - ICNP*, Riverside, California, USA, November 2001.

[Kel03] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks. *Computer Communication Review*, 32(2), April 2003.

[KHF06] Eddie Kohler, Mark Handley, and Sally Floyd. Datagram Congestion Control Protocol (DCCP). Request For Comments 4340, IETF, March 2006.

[KHR02] Dina Katabi, Mark Handley, and Chalrie Rohrs. Congestion control for high bandwidth-delay product networks. In *proceedings on ACM Sigcomm*, 2002.

[KK03] Young-Gook Kim and C.-C. Jay Kuo. TCP-Friendly Assured Forwarding (AF) Video Service in DiffServ Networks. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Bangkok, Thailand, May 2003.

[KK05] V. Kawadia and P. R. Kumar. Experimental Investigations into TCP Performance over Wireless Multihop Networks. In *In proceeding of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, August 2005.

[L. 97] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communications Review*, 27(1), jan 1997.

[LAF05a] Emmanuel Lochin, Pascal Anelli, and Serge Fdida. AIMD Penalty Shaper to Enforce Assured Service for TCP Flows. In *Proc. 4th International Conference on Networking (ICN 2005)*, La Reunion, France, April 2005.

[LAF05b] Emmanuel Lochin, Pascal Anelli, and Serge Fdida. Penalty Shaper to Enforce Assured Service for TCP Flows. In *IFIP Networking*, Waterloo, Canada, May 2005.

[LDJ06a] Emmanuel Lochin, Laurent Dairaine, and Guillaume Jourjon. gTFRC: a QoS-aware congestion control algorithm. In *Proc of the 5th International Conference on Networking*, Mauritius, April 2006.

[LDJ06b] Emmanuel Lochin, Laurent Dairaine, and Guillaume Jourjon. gtfrc, a tcp friendly qos-aware rate control for diffserv assured service springer telecommunication. *Springer Telecommunication Systems Journal*, 33(1-3):3–21, December 2006.

[Loc04] Emmanuel Lochin. *Qualité de Service dans l'Internet : Garantie de débit dans la claaAF*. PhD thesis, Université Pierre et Marie Curie, December 2004.

[LSF+06] A. Leiggener, R. Schmitz, A. Festag, L. Eggert, and W. Effelsberg. Analysis of Path Characteristics and Transport Protocol Design in Vehicular Ad Hoc Networks. In *In Proceedings of the 63. IEEE Semiannual Vehicular Technology Conference*, May 2006.

[MBYSG+07] Xavi Masip-Bruin, Marcelo Yannuzzi, Rene Serral-Gracia, Jordi Domingo-Pascual, Jose Enríquez-Gabeiras, María Ángeles Callejo, Michel Diaz, Florin Racaru, Giovanni Stea, Enzo Mingozzi, Andrzej Beben, Wojciech Burakowski, Edmundo Monteiro, and Luís Cordeiro. The EuQoS system: A solution for QoS routing in heterogeneous networks. *IEEE Communication Magazine*, 2(45):96–103, February 2007.

[MHT07] Lefteris Mamatas, Tobias Harks, and Vassili Tsaoussidis. Approaches to congestion control in packet networks. *Journal of Internet Engineering*, 1(1):22–33, January 2007.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. Request For Comments 2018, IETF, October 1996.

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, jul 1997.

[NPE00] B. Nandy, P.Pieda, and J. Ethridge. Intelligent traffic conditioners for assured forwarding based differentiated services networks. In *IFIP High Performance Networking*, Paris, France, June 2000.

[ns2] http://www.isi.edu/nsnam/ns/.

[PC04a] Eun-Chan Park and Chong-Ho Choi. Proportional Bandwidth Allocation in DiffServ Networks. In *Proc. of IEEE INFOCOM*, Hong Kong, March 2004.

[PC04b] Eun-Chan Park and Chong-Ho Choi. Proportional bandwidth allocation in diffserv networks. In *Proc. of IEEE INFOCOM*, pages 2038–2049, March 2004.

[PFTK98] Jitedra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proc. of ACM SIGCOMM*, pages 303–314, Vancouver, CA, 1998.

[Pos80] John Postel. User datagram protocol. Request For Comments 768, IETF, 1980.

[Pos81] John Postel. Transmission control protocol: Darpa internet program protocol specification. Request For Comments 793, IETF, 1981.

[RFB01] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to ip. Request For Comments 3168, IETF, September 2001.

[RHE99] Reza Rejaie, Mark Handley, and Deborah Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Proc. of IEEE INFOCOM*, pages 1337–1345, New York, March 1999.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol. Request For Comments 3261, IETF, June 2002.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. Request For Comments 3031, IETF, January 2001.

[RX05] Injong Rhee and Lisong Xu. Limitations of equation-based congestion control. *SIGCOMM Comput. Commun. Rev.*, 35(4):49–60, October 2005.

[SA00] R. Stewart and Al. Stream control transmission protocol. Request For Comments 2960, IETF, October 2000.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Request For Comments 1889, IETF, January 1996.

[SM05] Sarah Sharafkandi and Naceur Malouch. Simple and Effective End-to-End Approach to Increase TCP Throughput over Ad-hoc Networks. In *The 19th International Teletraffic Congress*, August 2005.

[SNP99] N. Seddigh, B. Nandy, and P. Pieda. Bandwidth assurance issues for TCP flows in a differentiated services network. In *Proc. of IEEE GLOBECOM*, page 6, Rio De Janeiro, Brazil, December 1999.

[SNV+02] Prasun Sinha, Thyagarajan Nandagopal, Narayanan Venkitaraman, Raghupathy Sivakumar, and Vaduvur Bharghavan. WTCP: A reliable transport protocol for wireless wide-area networks. *Wireless Networks*, 8(2-3):301–316, 2002.

[Soc05] IEEE Computer Society. IEEE Std 802.15.1 - 2005 IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. - Part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs). Technical Report 802.15, IEEE, 2005.

[Soc07] IEEE Computer Society. EEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Technical Report 802.11, IEEE, June 2007.

[SPF04] Manpreet Singh, Prashant Pradhan, and Paul Francis. MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS. In *Proc. of the IEEE International Conference on Network Protocols - ICNP*, Berlin, Germany, October 2004.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Request For Comments 2326, IETF, April 1998.

[Uni99] International Telecommunication Union. International Mobile Telecommunications-2000 (IMT-2000). Technical Report IMT-2000, ITU, 1999.

[VB05] Milan Vojnovi and Jean-Yves Le Boudec. On the long-run behavior of equation-based rate control. *IEEE/ACM Trans. Netw.*, 13(3):568–581, 2005.

[WBB04] J. Widmer, C. Boutremans, and J.Y. Le Boudec. End-to-end Congestion Control for TCP-Friendly Flows with Variable Packet Size. *ACM SIGCOMM Computer Communication Review*, 34(2):137–151, April 2004.

[Wid00] Jörg Widmer. *Equation-Based Congestion Control*. PhD thesis, University of Mannheim, February 2000.

[WKST04] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via tcp: An analytic performance study. In *Proc. of ACM Multimedia*, October 2004.

[XH06] Lisong Xu and Josh Helzer. Media streaming via TFRC: an analytical study of the impact of TFRC on user-perceived media quality. In *Proc. of IEEE INFOCOM*, April 2006.

[XHR04] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control for fast long-distance networks. In *INFOCOM 2004. Twenty-third*

AnnualJoint Conference of the IEEE Computer and Communications Societies, pages 2514–2524, March 2004.

[XSSK05] Yong Xia, Lakshminarayanan Subramanian, Ion Stoica, and Shivkumar Kalyanaraman. One more bit is enough. In *ACM SIGCOMM*, 2005.

Glossary

ADU	*Application Data Unit*
AF	*Assured Forwarding*
AFL	*Application Level Framing*
ALTQ	*ALTernative Queuing*
API	*Application Programming Interface*
AS	*Assured Service*
BE	*Best Effort*
BSD	*Berkeley Software Developpement*
CBQ	*Class Based Queuing*
CBR	*Constant Bit Rate*
CNRS	Centre National de la Recherche Scientifique
CP	*Chameleon Protocol*
DCCP	*Datagram Congestion Control Protocol*
ECN	*Explicit Congestion Notification*
EF	*Expedited Forwarding*
FIFO	*First In First Out*
FPTP	*Fully Programmable Transport Protocol*
GNU	*GNU's Not Unix*
GPS	*Generalized Processor Sharing*
GS	*Guaranteed Service*
gTFRC	*guaranteed TFRC*
IETF	*Internet Enginneering Task Force*
IP	*Internet Protocol*
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes
LAN	*Local Area Network*
MTU	*Maximum Transmission Unit*
OSI	*Open Systems Interconnection*
PDU	*Protocol Data Unit*
PHB	*Per Hop Behavior*
PQ	*Priority Queuing*
QoS	*Quality of Service*
QSTP	*Quality of Service Transport Protocol*
RED	*Random Early Detection*
RFC	*Request For Comment*
RIO	*RED with IN and OUT*

RR	*Read Rate*
RSVP	*Resource ReserVation setup Protocol*
RTSP	*Real Time Streaming Protocol*
RTT	*Round Trip Time*
SLA	*Service Level Agreemment*
SCTP	*Stream Control Transmission Protocol*
TBM	*Token Bucket Marker*
TCP	*Transmission Control Protocol*
TFRC	*TCP Friendly Rate Control*
TFRC_FC	*TCP Friendly Rate Control with a Flow Control*
TFRC_FC_SACK	*TCP Friendly Rate Control with a Flow Control and SACK mechanism*
TFRC_SACK	*TCP Friendly Rate Control with the SACK mechanism*
TOS	*Type Of Service*
TRTCM	*Two Rate Three Color Marker*
TSW	*Time Sliding Window*
TSW3CM	*Time Sliding Window Three Color Marker*
TTL	*Time To Live*
UDP	*User Datagram Protocol*
WAN	*Wide Area Network*
YAVOI	*Yet Another Victim Of Infocom*
YAAAAA	*Yet Another Annoying And Ambiguous Acronym*

List of Figures

List of Tables

Toward a Versatile Transport Protocol

Résumé : Les travaux présentés dans cette thèse ont pour but d'améliorer la couche transport de l'architecture réseau de l'OSI. La couche transport est de nos jour dominée par l'utilisation de TCP et son contrôle de congestion. Récemment de nouveaux mécanismes de contrôle de congestion ont été proposés. Parmi eux *TCP Friendly Rate Control (TFRC)* semble être le plus abouti. Cependant, tout comme TCP, ce mécanisme ne prend pas en compte ni les évolutions du réseau ni les nouveaux besoins des applications. La première contribution de cette thèse consiste en une spécialisation de TFRC afin d'obtenir un protocole de transport avisé de la Qualité de Service (QdS) spécialement défini pour des réseaux à QdS offrant une garantie de bande passante. Ce protocole combine un mécanisme de contrôle de congestion orienté QdS qui prend en compte la réservation de bande passante au niveau réseau, avec un service de fiabilité totale afin de proposer un service similaire à TCP. Le résultat de cette composition constitue le premier protocole de transport adapté à des réseau à garantie de bande passante.
En même temps que cette expansion de service au niveau réseau, de nouvelles technologies ont été proposées et déployées au niveau physique. Ces nouvelles technologies sont caractérisées par leur affranchissement de support filaire et la mobilité des systèmes terminaux. De plus, elles sont généralement déployées sur des entités où la puissance de calcul et la disponibilité mémoire sont inférieures à celles des ordinateurs personnels. La deuxième contribution de cette thèse est la proposition d'une adaptation de TFRC à ces entités via la proposition d'une version allégée du récepteur. Cette version a été implémentée, évaluée quantitativement et ses nombreux avantages et contributions ont été démontrés par rapport à TFRC.
Enfin, nous proposons une optimisation des implémentations actuelles de TFRC. Cette optimisation propose tout d'abord un nouvel algorithme pour l'initialisation du récepteur basé sur l'utilisation de l'algorithme de Newton. Nous proposons aussi l'introduction d'un outil nous permettant d'étudier plus en détails la manière dont est calculé le taux de perte du côté récepteur.
Mots clés : Protocôle de transport, Contrôle de congestion, Qualité de Service, Architecture légère, Optimisation algorithmique.

Toward a Versatile Transport Protocol

Abstract: This thesis presents three main contributions that aim to improve the transport layer of the current networking architecture. The transport layer is nowadays overruled by the use of TCP and its congestion control. Recently new congestion control mechanisms have been proposed. Among them, TCP Friendly Rate Control (TFRC) appears to be one of the most complete. Nevertheless this congestion control mechanism, as TCP, does not take into account either the evolution of the network in terms of Quality of Service and mobility or the evolution of the applications.
The first contribution of this thesis is a specialisation TFRC congestion control to propose a QoS-aware Transport Protocol specifically designed to operate over QoS-enabled networks with bandwidth guarantee mechanisms. This protocol combines a QoS-aware congestion control, which takes into account network-level bandwidth reservations, with full reliability in order mechanism to provide a transport service similar to TCP. As a result, we obtain the guaranteed throughput at the application level where TCP fails. This protocol is the first transport protocol compliant with bandwidth guaranteed networks.
At the same time the set of network services expands, new technologies have been proposed and deployed at the physical layer. These new technologies are mainly characterised by communications done without wire constraint and the mobility of the end-systems. Furthermore, these technologies are usually deployed on entities where the CPU power and memory storage are limited. The second contribution of this thesis is therefore to propose an adaptation of TFRC to these entities. This is accomplished with the proposition of a new sender-based version of TFRC. This version has been implemented, evaluated and its numerous contributions and advantages compare to usual TFRC version have been demonstrated.
Finally, we proposed an optimisation of actual implementations of TFRC. This optimisation first consists in the proposition of an algorithm based on a numerical analysis of the equation used in TFRC and the use of the Newton's algorithm. We furthermore give a first step, with the introduction of a new framework for TFRC, in order to better understand TFRC behaviour and to optimise the computation of the packet loss rate according to loss probability distributions.

Keywords: Transport Protocol, Congestion Control, Quality of Service, Light Architecture, Algorithmic Optimisation.