Explicit Congestion Control Algorithms for Time-Varying Capacity Media

Filipe Lameiro Abrantes

October 27, 2008

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Departamento de Engenharia Electrotécnica e de Computadores

Universidade do Porto

Faculdade de Engenharia

FEUP

Dissertação submetida para a obtenção do grau de
Doutor em Engenharia Electrotécnica e de Computadores

Dissertação realizada sob a supervisão do
Doutor Manuel Alberto Pereira Ricardo,
Professor Associado do
Departamento de Engenharia Electrotécnica e de Computadores,
Faculdade de Engenharia, Universidade do Porto

Porto, Outubro de 2008

Dissertação financiada pela Fundação para a Ciência e a Tecnologia
(Referência BD/19166/2004)

DOCTORAL DISSERTATION COMMITTEE

Prof. Rui Valadas

Full Professor at Instituto Superior Técnico da Universidade Técnica de Lisboa

Prof. Luís Bernardo

Auxiliar Professor at Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Prof. Artur Pimenta Alves

Full Professor at Faculdade de Engenharia da Universidade do Porto

Prof. José António Ruela Simões Fernandes

Associate Professor at Faculdade de Engenharia da Universidade do Porto

Prof. Mário Jorge Moreira Leitão

Associate Professor at Faculdade de Engenharia da Universidade do Porto

Prof. Manuel Alberto Pereira Ricardo

Associate Professor at Faculdade de Engenharia da Universidade do Porto

Abstract

One of the main drivers for Internet stability is the congestion control function embedded in the Transport Control Protocol (TCP) [1]. This function controls the bitrate of TCP connections and has been introduced by Van Jacobson in 1988 [2], after reports of congestion collapse emerged [3]. Since then it has helped the Internet through the years of its exponential growth with remarkable success. But this is coming to an end. The ever increasing bandwidth delay product of networks, and the latency and smoothness requirements of new applications start to unveil the limitations of TCP. In order to overcome them, the research community has dedicated efforts during the last decade to develop new congestion control mechanisms, and a new class of congestion control was introduced by several proposals [4], [5], [6] - eXplicit Congestion Control (XCC); it works based on explicit communication between network elements and end-systems, enabling the former to provide multi-bit feedback to the latter. This high-resolution feedback value allows the end-systems to use the network resources efficiently and have stable and responsive rates, regardless of the bandwidth delay product of the network. However, XCC mechanisms require that a router has an accurate estimate of its output link capacity, restraining their application in time-varying capacity media.

In this thesis we focus on the topic of XCC operation under conditions where the link capacity is unknown, variable, or non-trivial to estimate. The main contribution of this work consists in three alternative router algorithms that do not require the router to have knowledge about the link capacity. These algorithms rely either on queue properties, such as queue speed and

accumulation, or Medium Access Layer (MAC) information such as idle and busy periods. With the proposed algorithms the applicability of XCC is extended to shared access and/or multi-rate media.

Resumo

A função de controlo de congestionamento incluída no *Transport Control Protocol* (TCP) [1] é um dos pilares da estabilidade da *Internet*. Esta função é responsável pelo controlo do débito das ligações TCP, tendo sido introduzida por Van Jacobson em 1988 [2], após alguns relatos de colapso da rede devido a congestionamento [3]. Desde então, ela acompanhou a *Internet* durante os anos de crescimento exponencial evidenciando grande eficácia. Contudo, o infundável crescimento do produto da largura de banda pelo atraso das redes de comunicação e os requisitos de latência e estabilidade de novas aplicações começam a desvendar as limitações do controlo de congestionamento do TCP. De modo a ultrapassar essas limitações, a comunidade científica dedicou-se durante a última década ao desenvolvimento de novos mecanismos, tendo emergido uma nova classe de controlo de congestionamento [4], [5], [6] - Controlo de Congestionamento eXplícito (CCX); esta classe de mecanismos baseia-se na comunicação explícita entre os elementos de rede e os terminais, permitindo que os elementos de rede enviem para os terminais informação de controlo com resolução de vários bits. Esta informação de alta resolução permite aos terminais a utilização dos recursos da rede de forma eficiente, e que o débito das suas ligações seja simultâneamente estável e de rápida adaptação, independentemente do produto da largura de banda pelo atraso da rede. No entanto, os mecanismos de CCX requerem que um encaminhador tenha conhecimento preciso da capacidade das suas ligações, o que torna a sua aplicação em meios de transmissão de acesso partilhado e/ou capacidade variável difícil.

Esta tese foca o tópico da operação de CCX em condições em que a

capacidade das ligações de um encaminhador é desconhecida, variável, ou difícil de estimar. A principal contribuição desta tese consiste num conjunto de três algoritmos de CCX que não necessitam que um encaminhador tenha conhecimento da capacidade das suas ligações. Estes algoritmos baseiam-se em propriedades da fila do encaminhador, como a sua velocidade ou acumulação, ou em informação da Camada de Acesso ao Meio (CAM), como é o caso do período disponível e do período utilizado. Com os algoritmos propostos, a aplicação de CCX pode ser alargada a meios de acesso partilhado e/ou de débito variável.

Resumé

La fonction de contrôle de congestion incluse dans le *Transport Control Protocol* (TCP) [1] est un des piliers de la stabilité de la *Internet*. Cette fonction est responsable par le contrôle du dû des liaisons TCP, ayant été introduite par Van Jacobson, l'année 1988 [2], après quelques rapports de collapsus du réseau dû à congestion [3]. Dès lors, elle a accompagné la *Internet* pendant les années d'accroissement exponentiel mettant en évidence une grande efficacité. Cependant, l'infini accroissement du produit de la largeur de la bande par le délai des réseaux de communication et les conditions de latence et de stabilité demandé par nouvelles applications commencent à montrer les limitations du contrôle de la congestion du TCP. Pour réussir dépasser ces limitations, la communauté scientifique s'est dédié au développement de nouveaux mécanismes, pendant la dernière décade, ayant émergé une nouvelle classe de contrôle de congestion [4], [5], [6] - Contrôle de Congestion explicite(CCX); cette classe de mécanismes a son principe dans la communication explicite entre les éléments du réseaux et les terminaux, permettant que les éléments du réseau envoient pour les terminaux l'information de contrôle avec une résolution de multiple bits. Cette information d'une haute résolution permet aux terminaux l'utilisation des recours du réseau d'une façon efficace, et permet aussi que le dû de leurs liaisons soit en même temps stable et de rapide adaptation, indépendamment du produit de la largeur de la bande par le délai du réseau. Néanmoins les mécanismes de CCX requièrent q'un routeur ait de la connaissance précise de la capacité de leurs liaisons, cela rend son application dans les milieux de transmission d'accès partagé et/ou la capacité variable difficile.

Notre thèse focalise le topique de l'opération de CCX en des conditions où la capacité de liaisons d'un routeur est inconnue, variable, ou difficile d'estimer. La principale contribution de notre thèse consiste dans un conjoint de trois algorithmes de CCX qui n'ont pas besoin q'un routeur connaît la capacité de leurs liaisons. Ces algorithmes ont leurs principes dans des propriétés de la file du routeur, comme leur vitesse ou accumulation, ou dans l'information de la couche de Contrôle d'Accès au Milieu (CAM), comme est le cas de la période disponible et de la période utilisée. Avec les algorithmes proposés, l'application de CCX peut être étendue à des milieux d'accès partagé et/ou de dû variable.

Acknowledgements

I want to thank my supervisor, Manuel Ricardo, for his support and guidance during this four year long adventure. His advice and encouragement are the pillars of this thesis. I feel most fortunate for having João Taveira's collaboration in this thesis. His outstanding skills made the FreeBSD implementation possible, and his comments provided much insight. The contribution from many individuals, in numerous public presentations, paper reviews and mailing list discussions has also helped shape my work, for which I am grateful. I want to thank all my colleagues at INESC Porto, for the constructive working atmosphere in which I was able to develop the thesis, and to my girlfriend Laura, family and friends for their support over the years. To all a big thank you!

To Mom, Dad, and Laura...

Table of Contents

1	Introduction	1
1.1	Problem Statement	2
1.2	Contributions	4
1.3	Organization	4
2	Congestion Control in Packet Networks	5
2.1	The Basics of Congestion Control	5
2.2	Terms and Definitions	9
2.3	Modeling Congestion Control	13
2.4	End-System Congestion Control	18
2.4.1	The Reno Family	19
2.4.2	TCP-Friendly Rate Control (TFRC)	21
2.4.3	Vegas and FAST: Delay-Based Algorithms	22
2.4.4	Binary Increase Control (BIC and CUBIC)	24
2.4.5	Compound-TCP (CTCP)	26
2.4.6	Westwood(+)	27
2.5	Router-Assisted Congestion Control	27
2.5.1	Explicit Congestion Notification (ECN)	28
2.5.2	Active Queue Management (AQM) Mechanisms	28
2.5.3	eXplicit Congestion Control (XCC) Protocols	31
2.6	Summary	40
3	Problem Statement and Related Work	45
3.1	Problem Statement	45
3.1.1	Characterization of the Problem	45

3.1.2	Capacity Estimation Error	46
3.1.3	Simulation Results	50
3.1.4	Problem Statement Revisited	59
3.2	Related Work	60
3.3	Summary	62
4	XCC Algorithms for Time-Varying Capacity	65
4.1	The Queue Based Approach	66
4.1.1	The Blind Algorithm	66
4.1.2	The Error Suppression (ErrorS) Algorithm	77
4.2	The MAC-based Approach	86
4.2.1	The MAC Algorithm	86
4.2.2	Differences to the Wireless eXplicit Control Protocol (WXCP)	87
4.3	Algorithm Comparison	88
4.4	General Applicability to XCC Protocols	90
4.4.1	Blind Algorithm	91
4.4.2	ErrorS Algorithm	91
4.5	Summary	92
5	Validation and Performance Evaluation	95
5.1	Single Bottleneck Scenario	96
5.1.1	Simulation Setup	96
5.1.2	Basic Dynamics	98
5.1.3	Response to Variable Bandwidth	103
5.1.4	Efficiency and Scalability	110
5.1.5	Flow Fairness	113
5.1.6	Queuing Delay	117
5.1.7	Robustness to Traffic Pattern	120
5.2	Multiple Bottleneck Scenario	121
5.2.1	Simulation Setup	121
5.2.2	Basic Dynamics	123
5.3	Experimental Test-Bed	128

5.3.1	Test-Bed Setup	129
5.3.2	Basic Dynamics	130
5.3.3	Response to Variable Bandwidth	133
5.4	Summary	137
6	Deployment and Open Issues	139
6.1	Co-existence With Other Congestion Control Mechanisms . .	140
6.2	Non-XCC Routers	141
6.3	Non-XCC Hosts	143
6.4	Reaction to Packet Loss	144
6.5	Attacks and Security	145
6.6	Virtual Private Networks, Tunnels, MPLS, and IPSec	147
6.7	Pacing Mechanisms	148
6.8	Summary	149
7	Conclusion	151
7.1	Review of Work	151
7.2	Contributions	153
7.3	Limitations	154
7.4	Topics for Future Work	155
A	Stability Analysis of the ErrorS Algorithm	159
B	Recommended Parameter Values	165
C	Pseudo-code of the Blind Algorithm	167
D	Pseudo-code of the ErrorS Algorithm	169

List of Figures

2.1	Network Feedback Types	7
2.2	System model detailed	16
2.3	Simplified system model	17
2.4	TCP NewReno operation.	20
2.5	BIC vs CUBIC.	25
2.6	XCP Header	32
2.7	Number of flows	34
3.1	Compensation queue q_c represented in terms of the delay introduced $\left(\frac{q_c}{C_{real}}\right)$ as a function of the relative estimation error $\left(\frac{\epsilon}{C_{real}}\right)$	48
3.2	Queue time response to different levels of error in the capacity estimation.	51
3.3	Effect of the capacity estimation error $\left(\frac{\epsilon}{C_{real}}\right)$ on the compensation queue q_c : comparison between the simulation results and the theoretical analysis.	52
3.4	Queue time response in the presence of an estimation error $\frac{\epsilon}{C_{real}}$ of 20% for different system delays d_0	53
3.5	Effect of the base delay d_0 on the compensation queue: comparison between the simulation results and the theoretical analysis (Eq. 3.11) for a constant relative capacity estimation error $\frac{\epsilon}{C_{real}} = 20\%$. The compensation queue q_c grows linearly with the system base delay d_0	54

3.6	Time response of the queue to a capacity estimation error $\frac{\epsilon}{C_{real}} = 20\%$ and β values ranging from 0.15 to 1.0.	54
3.7	Effect of the parameter β on the compensation queue: comparison between the simulation results and the theoretical analysis. The compensation queue q_c decreases with increasing β	55
3.8	Queue time response for different values of β and an estimation error of 20%.	56
3.9	Time response of the queue to a 20% capacity estimation error with varied link capacities.	58
3.10	Capacity effect on the compensation queue: comparison between the simulation results and the theoretical analysis. . . .	58
4.1	The dynamics of the queue length $q(t)$, its moving average $\bar{q}(t)$, and the queue target $\kappa(t)$, with different low-pass filters applied to $\kappa(t)$, and $\bar{q}(t)$ with the Blind algorithm.	73
4.2	Excess utilization versus queue length of a system with the recommended values $\alpha = 0.4$, $\beta = 0.226$ and $\rho = 0.22$. The worst case scenario of capacity overshoot.	75
4.3	Left) The maximum value of Q_χ expressed as a fraction of Q_{max} , for various values of β and ρ . Right) The maximum amount of bandwidth distributed in each control interval for different values of β , and using the recommended value of ρ . . .	75
4.4	The effect of the <i>late reaction</i> mechanism.	76
4.5	The dynamics of the queue $q(t)$, its moving average $\bar{q}(t)$, and the queue target $\kappa(t)$, with different low-pass filters applied to $\kappa(t)$, and $\bar{q}(t)$ with the ErrorS algorithm.	82
4.6	Excess utilization versus queue length of a system with the recommended values $\alpha = 0.6$, $\beta = 0.1817$, $\mu = 0.1817$ and $\rho = 0.15$. The worst case scenario of capacity overshoot. . . .	84

4.7	Left) The maximum value of Q_χ expressed as a fraction of Q_{max} , for various values of β, μ and ρ . Right) The maximum amount of bandwidth distributed in each control interval for different values of β , and using the recommended value of ρ	84
4.8	The effect of the <i>late reaction</i> mechanism.	85
5.1	Single bottleneck simulation scenario.	97
5.2	The evolution of the congestion window throughout time for all algorithm combinations.	99
5.3	The evolution of the persistent queue length throughout time for all algorithm combinations.	100
5.4	Throughput measured at the base station (BS). Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.	103
5.5	The evolution of the congestion window throughout time for all algorithm combinations. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.	104
5.6	The evolution of κ and queue dynamics throughout time. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.	105
5.7	The evolution of the error suppression ($\frac{\alpha}{\mu} \cdot \xi(t)$) throughout time in ErrorS. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s. Note that the capacity estimate of the router is set to $C = 0$ Mbit/s.	107
5.8	Throughput measured at the base station (BS). Every 10 seconds one pair of flows leaves the network and another pair of flows enters the network. These pairs of flows use different data rates.	108
5.9	The evolution of κ and queue dynamics throughout time. Every 10 seconds one pair of flows leaves the network and another pair of flows enters the network. These pairs of flows use different data rates.	109

5.10	Efficiency as a function of medium capacity. TCP Reno utilization decreases with the increase of the medium bandwidth, while XCC mechanisms maintain their properties regardless of the bandwidth value.	112
5.11	The Jain's fairness index over time. XCC algorithms vs. Reno.	114
5.12	The Jain's fairness index over time. The difference between Reno with download traffic only and Reno with traffic in both ways.	116
5.13	The amount of traffic transported by flows with different RTTs.	117
5.14	The probability density function of the RTTs measured by the traffic sources. These measurements include the propagation RTT (≈ 80 ms) plus queuing delay.	118
5.15	Network utilization as a function of the percentage of short flows.	120
5.16	Multiple bottleneck simulation scenario.	122
5.17	The evolution of the congestion window throughout time in the chain scenario.	125
5.18	The evolution of the congestion window throughout time in the grid scenario.	126
5.19	The test-bed setup. Flows traverse from node R to the wireless clients $W(i)$	129
5.20	The evolution of the congestion window throughout time using our FreeBSD testbed.	131
5.21	The evolution of the queue length, its moving average and the parameter $\kappa(t)$ using a FreeBSD testbed.	132
5.22	The evolution of the congestion window throughout time using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.	134
5.23	The evolution of the queue length, its moving average and the parameter $\kappa(t)$ using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.	135
5.24	Data throughput throughout time using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.	136

A.1	The ErrorS control loop.	161
A.2	The maximum value of β as a function of n	163

List of Tables

2.1	Comparison of different congestion control mechanisms: Part 1.	42
2.2	Comparison of different congestion control mechanisms: Part 2.	43
5.1	The average throughput in the interval $[20, 40]$ s.	101
5.2	The relationship between the scale factor and the MAC layer parameters.	111
5.3	Jain's fairness index ranges.	115
5.4	The amount of traffic transmitted in the download and upload directions by TCP Reno.	116
5.5	The average value and the 95 th percentile of the persistent queue length, presented in packets, and as a % of the queue buffer size. For easier comparison of the results we have considered the buffer size to be 60 packet in all experiments. . . .	119
5.6	The relationship between the scale factor and the MAC layer parameters.	123
5.7	The standard deviation (in %) of the congestion window of all flows during the interval $[45, 80]$ s.	127
5.8	Comparison of the Blind, ErrorS and MAC algorithms and TCP Reno.	137
B.1	Recommended parameter values for the Blind and the ErrorS algorithms	165

Acronyms

ABR Available Bit Rate

ACK Acknowledgment

AFCT Average Flow Completion Time

AI Additive Increase

AIMD Additive Increase Multiplicative Decrease

AQM Active Queue Management

ARQ Automatic Repeat reQuest

ATM Asynchronous Transfer Mode

AVQ Adaptive Virtual Queue

BER Bit Error Ratio

BDP Bandwidth Delay Product

BIC Binary Increase Control

BS Base Station

BSD Berkeley Software Distribution

CAC Call Admission Control

CLTCP Coupling Logistics Transmission Control Protocol

CSFQ	Core Stateless Fair Queuing
CTCP	Compound Transmission Control Protocol
CTS	Clear To Send
DUPACK	Duplicate Acknowledgment
EC	Efficiency Controller
ECN	Explicit Congestion Notification
EMA	Exponential Moving Average
ErrorS	Error Suppression
FC	Fairness Controller
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
IEEE	Institute of Electrical & Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internetworking Protocol
IPSec	Internetworking Protocol Security
IPv4	Internetworking Protocol version 4
IPv6	Internetworking Protocol version 6
IRTF	Internet Research Task Force
MAC	Medium Access Control
MD	Multiplicative Decrease
MI	Multiplicative Increase

MIMD Multiplicative Increase Multiplicative Decrease

MPLS Multi Protocol Label Switching

MSS Maximum Segment Size

NIC Network Interface Card

OS Operating System

PEP Performance Enhancing Proxy

PF Packet Filter

QoS Quality of Service

QS-TCP Quick Start for Transmission Control Protocol

RCP Rate Control Protocol

RED Random Early Detection

RFC Request For Comments

RTS Request To Send

RTT Round Trip Time

TBF Token Bucket Filter

TCP Transmission Control Protocol

TFRC TCP-Friendly Rate Control

TTL Time To Live

UDP User Datagram Protocol

VCP Variable structure congestion Control Protocol

VJ Van Jacobson

VoIP Voice over Internetworking Protocol

WXCP Wireless eXplicit Control Protocol

XCC eXplicit Congestion Control

XCP eXplicit Control Protocol

Chapter 1

Introduction

One of the main drivers for Internet stability, and for IP-based networks in general, is the congestion control function embedded in the Transport Control Protocol (TCP) [1]. It has been introduced by Van Jacobson (VJ) in 1988 [2], after Internet congestion collapses had been reported [3]. Since then, this function has supported the Internet through its years of exponential growth and, in hindsight, congestion control has been very effective, suffering only small improvements [7], [8], [9] to its original design. However, as the bandwidth delay product (BDP) of networks scales and application diversity grows, Van Jacobson's algorithm is signaling its limits. The root of the problems is the low resolution feedback signal the algorithm receives from the network, which can take the form of packet loss or marking. This leaves too much guessing to be done when setting the congestion window that controls the rate of a flow, resulting in four problems. Firstly, the instantaneous rate used by the TCP connection will have an oscillatory behavior of at least $\frac{1}{3}$ of its mean rate. Secondly, the latency of the network will increase caused by queue build-up at the bottleneck. Thirdly, the flow bandwidth allocation becomes dependent on the round trip time (RTT), and flows with a high RTT are able to obtain only a fraction of the bitrate used by flows with small RTTs. Fourthly, efficiency on high BDP networks is limited, as a result of the fixed nature of the increase rule of VJ's algorithm.

Several approaches have been proposed to tackle the problems described.

During the 90's, the ideas of using delay variation [10], or the pace of the acknowledgment packets [11] as a source of network feedback emerged; however they never appeared as a standard, while VJ's algorithm was subject to numerous improvements [9], [7], [8], towards what is now called TCP New Reno.

More recently, another approach has been explored by the research community which consists in enabling explicit feedback from the network routers to the sources [4], [12], [5], [6]. Simulation results and initial implementations of these mechanisms showed promising results [4], [13], [14], [15], improving the accuracy and the stability of the congestion control function, as well as QoS characteristics of the network such as network latency or jitter. Additionally, an effort has been made on the characterization of the dynamics of these protocols and it was possible to prove analytically whether a protocol would maintain network stability for any link capacity, delay, and number of sources. Most of these protocols require the router to know the available link capacity. In some cases this may not be possible, either because the capacity is difficult to measure, or it is variable. An example of this case is the IEEE 802.11 [16] medium; it is a shared-access medium, meaning that its capacity is shared by all active stations, and this capacity is a function of some variables including the data rate of each station, the handshake mechanisms used for medium access, or the number of access collisions. In this type of media the router will not be able to effectively assess the instantaneous value of the link capacity, thus explicit congestion control mechanisms struggle to perform, if able to work at all, in networks composed of such transmission media.

1.1 Problem Statement

eXplicit Congestion Control (XCC) protocols have shown promising results in early simulation and implementation studies, helping to reduce network latency and jitter, as well as to increase the throughput stability and the efficiency of resource utilization. These characteristics help creating a media-

friendly network, and to keep the network efficiency high even when the bandwidth delay product of networks scale. However, such benefits can only be enjoyed by fixed-capacity networks: the fundamental assumption of the design of XCC algorithms is that the capacity of the transmission medium is fixed or, at least, known by the router.

This thesis explores the operation of XCC protocols in transmission media where, typically, it is non-trivial for a router to infer about their instantaneous media capacities. When this is the case, and because the XCC algorithms were designed assuming that an accurate estimate of the medium capacity is available, the XCC protocols perform either poorly or simply they do not work at all. An XCC router tends to adjust the rates of its incoming flows to match its estimate of the medium bandwidth. If the router underestimates the medium capacity, the medium will result under-utilized, thus limiting the network efficiency. If the router over-estimates the medium capacity, then the queue will build-up to compensate the excess of the router bandwidth estimation; it may happen that the queue buffer is not sufficient to compensate the estimation error and, in this case, a certain degree of packet loss and queue oscillation occurs, in proportion to the level of capacity over estimation. In both cases the network performance is severely impacted and the nice properties of XCC protocols are lost.

This problem becomes even more important as XCC protocols demand the whole network to have XCC functionalities. That is, if a router of the network is not XCC-capable, even if that router represents the bottleneck only for a subset of the flows traversing the router, none of these flows may use XCC; at least not with an optimal solution. So, the problem of operating XCC protocols over time-varying capacity media not only affects the deployment of XCC capabilities over this type of transmission media, but also affects the deployment of XCC on the whole network.

1.2 Contributions

The main contribution of this thesis consists in novel XCC router algorithms that enable the operation of XCC protocols over time-varying capacity media. Our approach differs from early work [17] in this field, by relying solely on the queue dynamics to regulate the network utilization. Our approach enables the router to abstract from the details of the underlying technology, providing a general solution to the problem. Our algorithms are, in particular, designed for the case of the eXplicit Control Protocol (XCP) [4] and the Rate Control Protocol (RCP) [5], two of the most representative XCC protocols; however the key ideas behind our proposed algorithms may be applicable to other XCC protocols. The proposed congestion control mechanisms are scalable and particularly adequate for new wireless technologies, which start to work at hundreds of Mbit/s, e.g. IEEE 802.11n.

During the design process of our algorithms we also characterized the effect that disturbances have on the performance of XCP and RCP, thus contributing to a better understanding of the control loop of these protocols. This result, in particular, can be used to further study and improve the performance and the robustness of XCC mechanisms.

1.3 Organization

This document is structured as follows. Chapter 2 presents the state of the art in congestion control, including a description of current proposals and considerations about their performance. Chapter 3 addresses in detail the problem of capacity mis-estimation and its effect on XCP and RCP, as well as related work. In Chapter 4 we present our main contribution which consists in algorithms applicable to XCC protocols in shared-access, time-varying capacity media. Chapter 5 describes the evaluation of the proposed algorithms made both by simulation and by a real-system implementation. In Chapter 6 a discussion on the open issues and the difficulties envisaged to deploy XCC protocols is presented. Finally, in Chapter 7, we conclude the thesis and point out directions for future work.

Chapter 2

Congestion Control in Packet Networks

This chapter presents the fundamentals and the state of the art on the topic of congestion control in the Internet. We begin by presenting fundamental background on congestion control in Sections 2.1 and 2.2. The topics of these sections include the basic notions of congestion control such as the types of feedback available from the network, the metrics used to benchmark congestion control algorithms and related definitions. In Section 2.3 a mathematical model used to study and design congestion control protocols is introduced. In Sections 2.4 and 2.5 individual congestion control proposals, representative of the state of the art, are presented; they are divided in proposals relying solely on the participation of end-systems, and proposals requiring also the cooperation of network routers. The chapter is summarized in Section 2.6.

2.1 The Basics of Congestion Control

Congestion control is the function responsible for controlling the traffic entry in communication networks and its purpose is, in the first place, to avoid the network to collapse due to congestion. A congestion collapse may occur when the traffic offered to the network exceeds the network capacity in a way that little to none useful communication is possible. The congestion control task

consists in adjusting the offered load to the network capacity which, in the end, implies more than just avoiding congestion collapse; it becomes closely related with rate control, its dynamics, and the fairness of the bandwidth allocation among flows.

Congestion control may be implemented in open-loop or closed-loop. Open-loop congestion control does not include mechanisms for the network to send back information to the sources, so they are unable to adapt their rate to the available network bandwidth. In this case, bandwidth reservation should be deployed in the network and the incoming traffic policed to ensure stable network operation. Bandwidth reservation can be more or less dynamic depending on the integration or not of Call Admission Control (CAC) and network monitoring mechanisms; still, the level of input traffic variation that such mechanisms can handle is limited. For this reason open-loop congestion control is not the best approach to perform congestion control in large-scale networks with dynamic traffic patterns such as the Internet and, for that reason, this type of congestion control is not addressed in this thesis.

On the other hand, if congestion control is implemented in closed loop, feedback from the network is available, which enables the control mechanism to adapt the amount of input traffic to the available network bandwidth. Closed loop congestion control has two major components: network feedback and source control. Source control refers to the methods used by the sources to shape the rate at which they inject traffic into the network. Network feedback consists in the signals received by a source from the network, which can be used to determine the congestion level of the network.

Source control is currently implemented using two approaches: window-based control and equation-based control. In window-based control, the source maintains a *congestion window* (*cwnd*) variable which quantifies the amount of packets that the source may have outstanding in the network within one RTT. Outstanding packets are those for which the sender has not yet received an acknowledgment from the receiver. Upon the reception of an acknowledgment packet, the source adjusts the value of the congestion window and, by doing that, it is adjusting its sending rate. The sending rate R can be obtained by $R = \frac{cwnd}{RTT}$. In equation-based control, the source

Figure 2.1: Different network feedback signals have different per-packet resolutions, and they may be either implicitly inferred from the communication or explicitly sent by network elements.

limits the sending rate by adjusting the interval between sent packets, and this interval is updated with some configured periodicity. Window-based control has the advantage of being self-clocked by the reception of packets, therefore no timers are required for this task, thus removing computational overhead and complexity; as a drawback, it allows bursts of data which may increase the level of jitter in the network. Equation-based is the opposite; it requires timers to count the intervals between packets thus becoming computationally more complex; however it produces a smoother, paced, output that helps maintaining low levels of jitter. Choosing a source control mechanism is mostly a trade-off between computational overhead and smoother data output. However, for high bandwidth delay products, where the *cwnd* may be too large, at least some degree of pacing is desirable to avoid large bursts of data.

Network feedback consists in the congestion signal received from the network, usable by a source to adjust its sending rate. These signals are the core of any congestion control protocol. The accuracy and the convergence time of an algorithm are tightly coupled with the resolution of the information contained in the congestion signal. These signals may be inferred *implicitly*

from the communication itself, e.g. a packet is lost, the delay increases, or they may be *explicitly* sent by network elements.

Packet loss is currently the default signal used by TCP; it represents one of the most rudimentary forms of congestion feedback as it contains a single bit of information per packet (congested vs. not congested). Packet loss is also a form of implicit congestion feedback, since congestion is derived from the loss occurrence rather than an explicit signal of congestion from the network. Relying on implicit signals has the drawback of having to make assumptions; in the case of packet loss, the assumption is that the loss of a packet is caused by congestion, while losses may also happen due to transmission errors. If the ratio between the Bit Error Ratio (BER) and the bandwidth of a medium is large, and if the data link layer does not implement any Automatic Repeat reQuest (ARQ) protocol, then loss-driven algorithms may fail to make an efficient use of the medium bandwidth, because packet loss caused by transmission errors will force the source to reduce its generation rate.

Another form of implicit feedback is delay variation. Unlike packet loss, delay variation is a high resolution signal, being that its resolution is limited only by the resolution of the end-host timer. Being an implicit signal, assumptions have to be made; in this case, the source assumes that an increase of delay is caused by queue build-up in the network. Similarly to the case of packet loss, delay may increase for other reasons than queue build-up. For example, if the data link performs ARQ, and a packet is corrupted during transmission, the re-transmission at the data link layer will be seen as an increase of the delay by the TCP sender. Additionally, there are some concerns among the research community about using delay as indication of congestion. For example, the study presented in [18] concludes that delay does not correlate well with the congestion state of the network.

The cases described above illustrate the drawbacks of using implicit feedback. In order to avoid these drawbacks, the research community started to address methods based on explicit feedback functionality in routers. These proposals range from simple single-bit feedback such as the Explicit Congestion Notification (ECN) [19] to more elaborated multi-bit feedback signals [4],

[5], [12], [20], which require their own header in the TCP/IP stack, usually referred to as the *congestion header*. ECN removes the problem of ambiguity for loss-based mechanisms; however, being still a single bit resolution signal, ECN feedback does not provide sufficient information for a source to adjust its sending rate rapidly and accurately to the network conditions. On the other hand, congestion control mechanisms that employ multi-bit explicit feedback allow the sources to produce a fast and accurate response, but it comes at the cost of increased router complexity and high deployment cost.

In summary, closed-loop congestion control has two major components: source control and network feedback. Source control can be either window-based or equation-based. Window-based control is computationally more efficient but may increase the level of jitter in the network, if the bandwidth delay product of the network becomes too large. For those cases, adjusting the interval between packets, as it is done in equation-based control, is preferable. Network feedback is the other component of congestion control. The higher the resolution of the congestion signal is, the more accurate and fast is the response of the congestion control algorithm. Additionally, if the signal is not explicitly sent by routers, but rather inferred from the communication itself, then there exists ambiguity about the nature of the signal, e.g. packet loss may be caused by transmission errors, and the congestion control algorithm may not lead to an efficient utilization of the network resources.

2.2 Terms and Definitions

The discussion of which metrics should be used to measure congestion control performance has been open since the congestion control topic has started to be addressed. Recently, some progress has been made with the publication of the informational RFC 5166 [21], and with the creation of a common evaluation framework [22], [23]. In this section we present the metrics relevant to the performance of a congestion control algorithm, and on which our evaluation of the mechanisms presented later on is based. The relevance of each metric as well as related definitions are also presented.

- **Utilization** metric, also referred to as **efficiency**, is usually measured at the bottleneck router as the percentage of the link capacity that an algorithm is able to utilize, although it may also be obtained by measuring the individual flow throughput at the source system. It is one of the most relevant metrics as it measures the amount of bandwidth that an algorithm wastes. It may be useful to analyze utilization in conjunction with queuing delay and loss ratio as there exists a trade-off between these 3 metrics. If the bottleneck queue is always occupied at high levels, it is able to absorb oscillations of bandwidth demand from sources, thus maintaining high utilization; however queuing delay is directly proportional to the level of queue build-up. The loss-ratio is also related with the average queue length in the case of most congestion control algorithms.
- **Throughput** consists in the absolute value of data that a source, or aggregate of sources, is able to send per second. It is also a component of the utilization metric. Sometimes, a distinction is made between goodput and throughput as goodput combines both throughput, loss-ratio, and other sources of inefficiency. The goodput is measured as the throughput of data at the receiver, that is, it only considers the amount of data that is correctly received by the destination.
- **Queuing Delay** represents the delay introduced by queuing along a path. It may be measured at the bottleneck router(s) as the time a packet waits in the router queue, or at the source of the flows by measuring the time between sending a data packet and receiving its acknowledgment. Sometimes it might be referred to solely as *delay* since the propagation delay of a path is considered to be constant. The delay is crucial in congestion control. Not only it affects directly the interactivity experienced by applications, but it also impacts all aspects of the congestion control algorithm - delay controls the clocking of the algorithms, thus it impacts on the rate at which the control algorithm is able to adapt to the changing network conditions.

- **Scalability** is the metric that characterizes how the congestion control algorithm performance is affected by the increase (or decrease) of the network bandwidth delay product. Scalability can be characterized, for example, as the utilization in function of bandwidth delay product. Some algorithms, usually those applying linear increase/decrease functions such as TCP Reno, are only suitable up to a limit value of the bandwidth delay product. As the bandwidth delay product grows beyond the limit these algorithms have their efficiency reduced. A long-term congestion control solution must be scalable.
- **Fairness** measures how the bandwidth is distributed among different flows. Although fairness is still a controversial topic [24], it is widely considered as beneficial that a congestion control algorithm distributes bandwidth evenly among all flows to ensure a minimum level of fairness in the network [25]. The most used metric of fairness is the Jain Fairness Index [26] applied to the throughput of individual flows x_i through:

$$J = \frac{(\sum_{i=1}^n \bar{x}_i)^2}{n \cdot \sum_{i=1}^n \bar{x}_i^2} \quad (2.1)$$

where n is the number of simultaneous flows. Another aspect related with fairness that has been gaining importance as a number of new algorithm proposals appear, is how fair the newly proposed algorithms are to the currently deployed algorithms, namely to TCP Reno.

- **Throughput Stability** measures how stable the throughput produced by the congestion control algorithm is, in steady-state. There is no consensual procedure for measuring throughput stability, but using the throughput variance is common. Throughput stability relevance is twofold: 1) it impacts the jitter level and the stability of the network; 2) applications using streaming buffers, e.g video-streaming, benefit from high throughput stability since it helps reducing the buffer size required.

- **Convergence Time/Responsiveness** is a metric of how fast the congestion control algorithm adapts to changing network conditions, which may be the change of the capacity of the bottleneck link, the change of the path round trip time, or the entrance or departure of flows. There is no universal definition of how to measure the convergence time. In many cases, the convergence time is measured as the time the algorithm takes to achieve 90% of the target throughput range, and it is expressed in terms of the per-flow bandwidth delay product. Note that for many congestion control algorithms it is even difficult to define a target throughput range as their rate tends to oscillate heavily, e.g. TCP Reno. Responsiveness is important to ensure network stability and also to enable the fast completion of short flows. Usually, responsiveness and throughput stability are somehow related and a trade-off between the two may apply.
- **Loss Ratio** represents the ratio of packets that are dropped due to queue overflow. Optionally it may also include packet loss due to corruption.
- **Deployability** measures the barrier to deploy a congestion control mechanism in the Internet. It characterizes the modifications required into the existing infrastructure to deploy the mechanism, and the difficulty of co-existence with the currently deployed control mechanisms. Three levels of modifications are identified: host-only modifications, network-only modifications, and host and network modifications. Host-only modifications are easier to deploy, while network modifications are more difficult to deploy since they require a higher up-front investment.
- **Security/Robustness in Uncooperative Environments** represents the level of robustness of a mechanism in the presence of mis-behaving users. If a mechanism is vulnerable to attacks generated on end-systems its deployment in the Internet poses a risk to its stability. The robustness metric is measured as the risk of the vulnerabilities associated with the mechanism. Vulnerability examples include the possibility of

a user achieving a throughput higher than his fair share or conducting denial-of-service (DoS) attacks to other users.

- **Media Friendliness** refers to the suitability of a congestion control mechanism to multimedia applications. Multimedia applications usually require low delay and stable throughput. The lower the queuing delay and the more stable the throughput are, the more *media friendly* the congestion control mechanism is. Low delay enhances the interactivity experienced by applications; it is beneficial to applications involving two-way interaction such as interactive multimedia applications, e.g. VoIP, but also to other types of interactive applications, such as web-browsing. Throughput stability is important for multimedia streaming applications. These applications usually require the management of a playback buffer; if the throughput varies too much during a session, either the application buffer is big enough to accommodate those variations or the user will experience hick-ups during media playback.
- **Safety** refers to how safe deploying a congestion control mechanism in the Internet is with regards to stability and security. Stability analysis, either theoretical or real-case studies, is fundamental to estimate the impact of the deployment of a mechanism in an Internet scale. Simulation studies alone are not able to determine whether an algorithm maintains network stability under general conditions. Security is also important; when deploying a congestion control mechanism in the Internet, one has to assume that users will misbehave and exploit the vulnerabilities of the mechanism, so the risk level of those vulnerabilities should be characterized.

2.3 Modeling Congestion Control - a Control Theoretic Approach

Congestion control in the Internet is one of the largest ever man-made feedback control systems. Furthermore, the scale and the dynamic nature of the

Internet makes it virtually impossible to replicate in simulation. For this reason, mathematical modeling and analysis plays an important role in the design and evaluation of congestion control algorithms. Such analysis allows us to determine the dynamic properties of the algorithms, such as stability and convergence time, under general conditions. This knowledge aids the algorithm designer in the task of parameter tuning, but also in the prediction of the network response. In this section, congestion control systems are characterized formally from a control theoretic perspective.

Congestion control can be regarded as a delayed feedback control system where the sources adapt their rate according to delayed feedback they receive from the network. Before describing the system, the following variables are defined:

- $y(t)$ is the aggregate rate generated by the sources;
- $d(t)$ is the system delay, while d_0 is the path propagation delay;
- $p(t)$ is the congestion signal received by the source;
- $q(t)$ is the bottleneck queue length;
- C is the capacity of the bottleneck link.

Abstracting from packet granularity, the dynamic system may be described by the set of equations:

$$\dot{y}(t) = F(p(t-d)) \quad (2.2)$$

$$p(t) = f(y(t) - C, q(t)) \quad (2.3)$$

$$\dot{q}(t) = y(t) - C \quad (2.4)$$

$$d(t) = d_0 + \frac{q(t)}{C} \quad (2.5)$$

Equation 2.2 states that the variation of the aggregate bandwidth of the sources is a function F of the delayed congestion signal $p(t-d)$. The function F can be seen as the core of the congestion control algorithm, as it defines how

the rate of the sources adapts to the congestion signal $p(t)$ received from the network. Equation 2.3 states that the congestion signal $p(t)$ is a function of the excess utilization, $y(t) - C$, and of the length $q(t)$ of the bottleneck queue. Equation 2.4 describes the physical relationship between the bottleneck queue length and the excess bandwidth. Equation 2.5 describes the delay $d(t)$ of the system as the sum between the path propagation delay d_0 and the queuing delay at the bottleneck queue. Applying the Laplace transformation [27] the system represented by the above set of equations may be represented by the control loop shown in Fig. 2.2. Note that a delay d is represented by e^{-sd} in the Laplace domain. Note also that the interaction between queue length and queuing delay is hidden in this model. Fig. 2.2 contains two models that are equivalent. One represents the congestion control algorithm F implemented in the source, while the second model represents the algorithm implemented in the router. A simplified, and equivalent version of these two models is shown in Fig. 2.3.

The system model represented in Fig. 2.3 can be studied using control theory. For instance, by working over the system open-loop transfer function we can study the system stability. Using the closed-loop transfer function, the system response $(y(t) - C)$ can be analyzed. Among the characteristics that can be analyzed are the fundamental frequency of oscillation of the system response, or the initial and final values of the response in the time domain. The open-loop transfer function is defined as:

$$Z(s) = G(s) \cdot H(s) \quad (2.6)$$

while the closed-loop can be expressed by:

$$\frac{C(s) - Y(s)}{C(s)} = \frac{G(s)}{1 + G(s) \cdot H(s)} \quad (2.7)$$

The stability of the system can be analyzed by determining either the phase margin ϕ or the gain margin Ω of the system. These margins must be positive, and they may be obtained by solving $\phi = \pi - \angle(G(s) \cdot H(s))|_{s=jw_z}$ or $\Omega = |(G(s) \cdot H(s))|_{s=jw_\pi} - 1$, where w_z represents the frequency at which the

(a) Source

(b) Router

Figure 2.2: The detailed congestion control models. The models represent two design alternatives, implementing the control algorithm F in the sources (a) or in the routers (b).

Figure 2.3: The simplified feedback system model.

gain is unitary, and w_π represents the frequency at which the phase of the open-loop response is $-\pi$. Putting it more simply, these margins express how far the the open-loop response $G(s) \cdot H(s)$ is from the value -1 . If $G(s) \cdot H(s) \leq -1$ then the closed-loop response will be ∞ , and the system results unstable. The frequency w_z for which the open-loop response gain is unitary also represents the closest point of the open-loop response to -1 . This frequency is called the fundamental frequency of the system, since it is the most amplified frequency by the control loop. Additionally, the final value theorem can be used to identify the value at which the time response of the system will stabilize. This theorem states that:

$$\lim_{t \rightarrow \infty} C - y(t) = \lim_{s \rightarrow 0} s \cdot \frac{G(s)}{1 + G(s) \cdot H(s)} \cdot C(s) \quad (2.8)$$

Ultimately, one can determine the exact time-response expression given a

certain input signal, by applying the inverse Laplace transform to the output signal:

$$C - y(t) = \mathcal{L}^{-1} \left[\frac{G(s)}{1 + G(s) \cdot H(s)} \cdot C(s) \right] \quad (2.9)$$

However, this is usually less tractable and used only to predict the exact system response for a specific input function.

In summary the congestion control system can be modeled as a delayed feedback control loop. Using the tools provided by control theory, the system can be studied with regard to stability, convergence dynamics, and steady-state response. However, the model presented has some limitations: it considers negative queue lengths, which are physically impossible; it also hides the impact of queuing delay on the overall system delay; and it considers all flows to have a common RTT. The negative queue aspect amplifies the oscillation amplitude of the system response. For this reason, system analysis of the model produces conservative values regarding system stability, as it is shown in [28]. The impact of queuing delay may be ignored if queuing delay is small relative to the overall system delay d ; however when the queuing delay becomes large this has to be considered when determining d . The assumption of a uniform RTT among all connections is caused by the difficulty in modeling the behavior of different RTTs. When studying a system where connections have different RTTs, the average RTT of all connections should be used as the system base delay. Of course, if the RTTs are too heterogeneous, then the accuracy of the model may be compromised and further experimental tests should be performed to complement the theoretic study.

2.4 End-System Congestion Control

This section focuses on the congestion control algorithms present at the end-systems. Typically, they are embedded in TCP and often the congestion control and TCP terms are used as synonyms. These algorithms rely on the ability of the end-system to measure the congestion state from signals such as packet loss (or marking), delay variation, or the ACK rate. Currently, the most common and widely accepted algorithm is NewReno; however, some

systems are starting to shift to other algorithms. Linux for example, now uses by default the CUBIC [29] algorithm, while Windows Vista uses its home brewed Compound-TCP (CTCP) [30]. Both are still under revision and standardization in the IRTF and IETF, and the precocious deployment of these newly developed congestion controllers raises interoperability and stability concerns as they lack wide scale experimentation. Anyway, changes are occurring in what was seen as one of the immutable pillars of the Internet only a few years ago, meaning that now a driving force exists for the change.

2.4.1 The Reno Family

NewReno [8] is currently the most widely deployed and commonly accepted algorithm of congestion control in the Internet. It is the result of the small improvements to the Tahoe [31], and Reno [7] algorithms. The NewReno algorithm has 4 phases: congestion avoidance, slow start, fast retransmit and fast recovery. Slow start occurs at the beginning of the connection or after a timeout, and it consists in the exponential growth of the sending rate to probe network for resources. Fast recovery and fast retransmit are mechanisms that allow NewReno to deal with the retransmission of a lost packet efficiently, while congestion avoidance is the core of the rate control algorithm. During congestion avoidance NewReno adopts an Additive Increase Multiplicative Decrease (AIMD) rule. In the absence of congestion, the rate is increased by a fixed amount of 1 Maximum Segment Size (MSS) divided by the RTT of the flow, while if congestion is detected the rate is cut to half. The MSS is usually the maximum packet size allowed in the path of the flow. The rate adjustment decision is taken upon the reception of each Acknowledgment (ACK) packet, or after a timeout for the reception of the ACK. If the ACK denotes the loss or marking of a packet in the network, the source will interpret it as a sign of congestion; otherwise the network is perceived as non-congested by the source. An ACK signals the loss of a packet if it represents the third duplicate ACK (DUPACK). A DUPACK is the acknowledgment of a data packet received out of order.

The NewReno algorithm has a few undesirable characteristics. As seen in

Figure 2.4: TCP NewReno operation.

Fig. 2.4, the output rate has a high degree of oscillation, as a result of (1) its aggressive reaction to congestion and (2) the poor resolution of the congestion signal which does not detail the degree of congestion in the network. Such oscillatory behavior is especially harmful to applications that require buffering, e.g. video streaming, since they are unable to quickly assess the size of buffer required to absorb the oscillations, existing the possibility of the user experiencing hiccups during the transmission. An additional disadvantage is the increased latency due to queue build-up. NewReno requires queues on the network to build up to a point where packets need to be dropped or marked and that reduces the interactivity experienced by network applications. Applications for which interactivity plays a crucial role include HTTP, online gaming or VoIP. Hidden in the dynamics of NewReno there are two more major drawbacks: 1) as the aggressiveness of the increase rule is tied to

the inverse of the flow RTT; flows with higher RTTs usually get less bandwidth than those with lower RTTs, and, 2) since the increase δ during the congestion avoidance phase is fixed, NewReno loses efficiency as the network bandwidth delay product scales. All these drawbacks are now seen as limitations that NewReno congestion control imposes to a high-speed, stable, and media-friendly Internet. These drawbacks have triggered the development of new congestion control algorithms and approaches, particularly during the last decade.

2.4.2 TCP-Friendly Rate Control (TFRC)

The TCP-Friendly Rate Control (TFRC) algorithm [32] has been derived from an early effort [33] of characterizing analytically TCP's (Reno) throughput. TFRC suggests that the output rate is set by using an equation that estimates the average throughput of TCP Reno over a certain period of time, under a given packet loss probability. TFRC is said to be TCP-friendly because it will achieve the same rate as TCP (Reno) on average, if measured over a sufficiently large period of time. The equation used by TFRC is as follows [33]:

$$T = \frac{s}{RTT \cdot \sqrt{\frac{2p}{3}} + t_{RTO} \cdot \left(3\sqrt{\frac{3p}{8}} \cdot p \cdot (1 + 32p^2)\right)} \quad (2.10)$$

where T denotes the maximum rate of the sender, p is the measured packet loss probability, s represents the packet size, and t_{RTO} represents the retransmission timeout, usually approximated by $4 \cdot RTT$. The mechanism used to calculate the loss probability p is the key aspect of the operation of TFRC as it defines its dynamic properties. Currently, the TFRC standard [34] states that the loss probability should be averaged over the period of the last 8 losses. By transforming loss events into a loss probability, TFRC trades responsiveness for throughput stability. Packet loss is a signal with low resolution (1-bit per packet), while loss probability has multi-bit resolution but with low frequency (multi-bit information per period of measurement of the loss probability). This means that TFRC will be able to choose its

rate more accurately, producing a stable throughput, but it will take more time to react to changing network conditions. For this reason, TFRC is more suitable for applications requiring a stable throughput, e.g. media streaming. However, TFRC will struggle when network conditions vary rapidly such as when vertical handovers occur, as seen in [35] and [36]. It has also been reported [37], [38] that TFRC may fail to mirror Reno's throughput if certain conditions are observed. Finally, TFRC has one inherent limitation, which is its assumption that most of the traffic with which it is competing is actually controlled using the Reno algorithm.

2.4.3 Vegas and FAST: Delay-Based Algorithms

The Vegas algorithm was proposed in 1994 [10]. It suggested to implement congestion control based on the amount of queued packets, which is equivalent to say queuing delay. Vegas inferred how many packets were buffered in the network by computing the difference between the expectable amount of data to be received in a certain interval and the actual amount of data received. If the amount of buffered packets (or bytes) was within the interval $[\alpha, \beta]$, the rate would remain unchanged, but if it was either above or below the interval's limits, the rate would be linearly decreased or increased, respectively. The interval $[\alpha, \beta]$ was the amount of data that was considered *right* for the network buffers to hold, and consisted in the equilibrium point of the algorithm. Using the amount of queuing in the path as a signal of congestion allowed Vegas to detect congestion before packets were dropped; by slowing down as queuing increased, Vegas was able to achieve a lower packet drop ratio than the family of Reno algorithms.

More recently, the concept used by Vegas was extended to make it more flexible, and achieve faster response times. The resulting algorithm is known as FAST [39]. The equation used by FAST to update the sender congestion window w is the following:

$$w_{n+1} = \min \left[2 \cdot w_n, (1 - \gamma) \cdot w_n + \gamma \cdot \left(\frac{baseRTT}{RTT} w_n + \alpha \right) \right] \quad (2.11)$$

where $baseRTT$ denotes the minimum RTT observed during the connection time, and γ, α are tunable constants of the algorithm. FAST adjusts the congestion window according to the amount of queuing delay in the path. The value of $\left(1 - \frac{baseRTT}{RTT}\right) \cdot w_n$ denotes the amount of packets (or bytes) currently queued in the network, while $\frac{baseRTT}{RTT} w_n$ reflects the value of the congestion window required if the sender wants to drain the outstanding queue of the bottleneck. But because empty queues are a very thin point of equilibrium, FAST, as Vegas does, allows some queue to build on the bottleneck, and then achieve equilibrium on that point. This equilibrium point is adjustable by the α parameter, which controls the amount of packets/bytes that each source is allowed to queue on the bottleneck. The update of the congestion window of FAST uses a mechanism of Exponential Moving Average (EMA) following the target value of the window. The EMA mechanism can also be regarded as a low-pass filter that may be used to produce smoother transitions and also filter the noise inherent to network and traffic fluctuations. The γ parameter controls the smoothness of this filter, being that $\gamma \in]0, 1]$. Lower values of γ increase the smoothness of the response (but also lower the responsiveness to network changes). Additionally, FAST clamps the growth of the congestion window to 100% in consecutive intervals to preserve stability (it is the equivalent to the slow start phase of the Reno family). Stability properties of the FAST algorithm have been derived analytically in [39].

Although queuing delay is theoretically a more accurate measure of congestion than packet loss or loss probability, there are still some concerns about its applicability and feasibility in practice. In fact, there exist reports [40], [41], [18] denoting the weak correlation between increased delays and congestion losses. Other experiments [42], [43] show that reverse traffic severely affects FAST throughput, which is understandable since it sees any increase of the path delay as a sign of congestion, even if it is on the reverse path. Additionally, queue build-up in the bottleneck is proportional to the α parameter, but mostly to the number of concurrent flows, meaning that the queued buffer might be significant in the presence of a large number of flows. The α parameter itself requires tuning.

2.4.4 Binary Increase Control (BIC and CUBIC)

The Binary Increase Control (BIC) [44] algorithm faces the rate adjustment problem as a binary search problem, in which it tries to find the *right* sending rate through binary answers from the network. The binary answers from the network consist in information regarding the congestion state of the network, i.e. congested vs. not congested, which, as expected, take the form of packet loss or marking. Through these answers, BIC travels along the rate solution space until the solution error interval becomes small enough. BIC connections start using a slightly modified slow start phase, until the network drops or marks a packet. When a packet is dropped, the window is reduced by a factor β . The size of the window right before the window cut is set to be the maximum W_{max} of the next search interval, and the window size right after the cut is set as the minimum W_{min} of the interval. The window is then set to the middle point of that interval. If the network does not drop packets with the updated window value, the middle point becomes now the minimum of the interval and the process goes on until the difference between the maximum and the minimum of the interval reaches a certain minimum value S_{min} called the *minimum increment*. If that minimum interval is reached, the window is set to W_{max} , and the search continues but for values above W_{max} , and starting with an interval of $2 \cdot S_{min}$. BIC establishes one more rule to improve TCP-friendliness and stability that consists in limiting the maximum increase of the window to a certain *maximum increment* S_{max} . This results that, for large window cuts, the initial period of window adaptation looks like the Reno family algorithms with an additive parameter of S_{max} . The essential difference in BIC from Reno, is that it tries to stabilize throughput, rather than increase linearly until congestion occurs. The rate increase of BIC is a logarithmic function of the window size, and by that it is able to approach full utilization more gently, resulting more stable. It also performs better in high bandwidth delay products, because the increase when the window is low is faster than that of Reno. Still, it produces large standing queues in the network, and the convergence times vary greatly.

CUBIC [29], by the same authors, enhances the BIC window growth

Figure 2.5: BIC and CUBIC window growth functions.

function. It proposes that a cubic function having properties similar to the window function of BIC (Fig. 2.5) is used to update the congestion window value. This function takes as arguments the time offset t from the last packet loss event, the window size right before the loss event W_{max} and the constants C, K :

$$W_{CUBIC} = C \cdot (t - K)^3 - W_{max} \quad (2.12)$$

C is a scaling factor, while $K = \sqrt[3]{W_{max} \cdot \frac{\beta}{C}}$ (β is the reducing factor of the window upon packet loss). By using this equation, CUBIC greatly improves the fairness of the distribution of bandwidth among flows by making it independent of the RTT. Because the function depends on t , which represents absolute time rather than a number of control steps (RTTs), CUBIC is able to decouple its response from the flow RTT. However, this solution

raises concerns about stability because in a delayed control loop, the response dynamics should be related with the loop delay. That is, control decisions should be more conservative when the feedback delay is high, while the control may be more aggressive when the feedback loop delay is small; if this rule is not applied, system stability may not be guaranteed for any system delay. As a final note let us point out that CUBIC is currently the default congestion control mechanism of the Linux OS.

2.4.5 Compound-TCP (CTCP)

The Compound-TCP approach [30] proposes to use delay and loss as a combined congestion signal. The congestion window $cwnd$ will have two components, one related with the loss signal $lwnd$, and another with the delay signal $dwnd$. The loss related component evolves following the regular Reno algorithm, while the delay component ($dwnd$) will evolve according to the queuing delay. The resulting window is the sum of both components. The delay component will grow exponentially while the amount of packets in the network buffers is below a certain threshold γ . When this amount of buffered packets grows above the threshold, $dwnd$ is decreased linearly. The algorithm is characterized by the set of equations (note that the notation used here is slightly different from that used by the CTCP authors):

$$cwnd_n = lwnd_n + dwnd_n \quad (2.13)$$

$$lwnd_{n+1} = \begin{cases} lwnd_n + \frac{1}{lwnd_n + dwnd_n} & \text{if !loss} \\ lwnd_n \cdot (1 - \beta) & \text{if loss} \end{cases} \quad (2.14)$$

$$dwnd_{n+1} = \begin{cases} dwnd_n + (\alpha \cdot cwnd_n^k - 1) & \text{if diff} < \gamma \\ dwnd_n - \psi \cdot diff & \text{if diff} \geq \gamma \\ cwnd \cdot (1 - \beta) - \frac{cwnd}{2} & \text{if loss} \end{cases} \quad (2.15)$$

where $diff$ represents the estimation of packets buffered in the network, calculated as $diff = \left(\frac{cwnd}{baseRTT} - \frac{cwnd}{RTT} \right) \cdot baseRTT$, and the γ threshold is the number of packets that a source is allowed to buffer in the network. α , k are

tunable parameters to control the exponential gain of the delay component when *diff* is below the allowed threshold γ . β is another tunable parameter that controls the window cut upon the loss of a packet. The resulting output rate of CTCP is argued by the authors to be scalable to high bandwidth delay product networks, while being more fair to Reno than most novel congestion control proposals. CTCP also shares most of the characteristics of FAST, including its drawbacks such as the susceptibility to reverse traffic, or the queue scaling with the number of flows.

2.4.6 Westwood(+)

Westwood [11] or Westwood+ as the Westwood/NewReno combination is known, adopts an additional feedback signal that consists in the rate at which acknowledgment (ACK) packets arrive to the sender. The main difference in Westwood is that when a loss is detected, the rate is reduced to the value of the bandwidth estimation rather than being cut to half. This measure allows Westwood to avoid the conservative cut present in Reno algorithms, and with that it is able to scale to higher bandwidth delay product networks, as well as producing a more stable throughput. However, Westwood still lacks significant research regarding its safety and fairness, and it does not provide small queuing delays.

2.5 Router-Assisted Congestion Control

This section describes mechanisms implemented by the network to enhance the congestion control function. The Active Queue Management (AQM) and the Explicit Congestion Notification (ECN) marking mechanisms are pieces of a congestion control system, used to complement traditional end-system congestion control mechanisms that rely on packet loss or marking. Other approaches, grouped under the eXplicit Congestion Control (XCC) class, implement different, explicit congestion signals and require a dedicated algorithm running on the end-systems that is able to transform the enhanced congestion signal into a sending rate.

2.5.1 Explicit Congestion Notification (ECN)

The Explicit Congestion Notification mechanism [19] is a simple mechanism that allows routers to signal congestion to sources *before* routers have to drop a packet. Additionally, explicit notification helps to disambiguate the nature of the congestion signal, since packets may get dropped not only due to congestion but also due to transmission errors; different action will be taken for these cases what contributes to maintain low standing queues. The ECN mechanism was standardized in the IETF and a 2-bit field was defined in the IP header to carry the ECN code. The 2 bits are used both to carry the congestion information, in a binary form (congested vs. not congested), and to negotiate the ECN capability of a path. The congestion information contained in the ECN field plugs naturally in loss-based end-system algorithms. Note that routers along a path may not support ECN marking. ECN was the first step in enabling explicit communication between the end-systems and the network, for the purpose of congestion control.

2.5.2 Active Queue Management (AQM) Mechanisms

Active Queue Management (AQM) consists in a control mechanism implemented by routers to selectively drop packets, rather than just drop a packet when the buffer is full. These mechanisms, in conjunction with the source algorithm, may help to improve the fairness or the delay of the network. This section describes two AQM algorithms, representative of this class of mechanisms: RED which aims to improve fairness and delay, and AVQ which achieves near-zero queuing delay in conjunction with the Reno algorithm.

2.5.2.1 Random Early Detection (RED)

Random Early Detection [45] is a queue management technique that aims to reduce queuing delay and minimize unfairness and inefficiency caused by source synchronization [46]. Source synchronization is observed when sources send packets with a fixed periodicity, which may happen due to TCP's self-clocking properties, causing certain sources to have more packets dropped

than other sources, thus obtaining a lower share of bandwidth. The idea behind RED is that packets are dropped (or marked) before the queue is full, allowing sources to perceive congestion before the bottleneck queue overflows. Additionally, RED adds a random factor to choose the packets to drop, which causes drops to be more distributed than with the DropTail drop policy, which reduces the effect of source synchronization. RED defines two queue threshold levels, Q_{min} and Q_{max} , which define the interval at which the queue will try to stabilize. RED tracks the queue congestion through an exponential moving average of the queue length. Upon the arrival of a packet, three situations may occur: 1) the queue average $\bar{q}$ is above the interval $[Q_{min}, Q_{max}]$, indicating persistent congestion - in this case the packet gets dropped, 2) the queue average is below the interval minimum Q_{min} , perceived as low level of congestion by the queue manager - in this case the packets is not marked nor dropped, and 3) the queue average is within the interval $[Q_{min}, Q_{max}]$ - in this case the packet is dropped with probability p :

$$p = \frac{\bar{q} - Q_{min}}{Q_{max} - Q_{min}} \quad (2.16)$$

note that p increases linearly with the average of the queue $\bar{q}$. Using this strategy, RED is able to reduce the average queue length over time without discriminating bursty sources. By computing an weighted average of the queue length rather than using the instantaneous queue length, RED allows bursts of packets without considering them persistent congestion. The extent of the burst allowed by RED is configurable using the weight parameter w_q associated to the current sample in the moving average calculation:

$$\bar{q} = w_q \cdot q + (1 - w_q) \cdot \bar{q} \quad (2.17)$$

where q represents the instantaneous queue length and $\bar{q}$ is the exponential moving average of the queue length.

As noted above, RED is able to both reduce the average queue length and the synchronization effects when used in cooperation with the Reno algorithm. However, it represents a small improvement as the individual flow

dynamics maintain the characteristics of Reno, such as throughput instability and network inefficiency for high bandwidth products. It has been argued that the gains introduced by RED are thin [47]. Furthermore, tuning RED's parameters is difficult as it depends on the link capacity and on traffic characteristics such as burstiness or number of sources, and those are unlikely to remain constant over time.

2.5.2.2 Adaptive Virtual Queue (AVQ)

Adaptive Virtual Queue (AVQ) management [48] is an interesting approach that provides very low queuing delay in the network, by sacrificing a marginal fraction of the throughput. AVQ maintains two queues. One is the real, physical buffer where packets are stored. The other is a virtual queue whose length is calculated by the queue controller, as if the link bandwidth were slightly lower than the actual link capacity. This is called the *virtual capacity* or $\tilde{C}$. The growth of the virtual queue length will outpace that of the real queue, and because packets will start to get dropped once the virtual queue overflows, the physical queue length is kept at a very low level. The virtual capacity $\tilde{C}$, which cannot exceed the link capacity C , is updated using the following differential equation:

$$\tilde{C}(t) = \min(\tilde{C}, \gamma \cdot C) \quad (2.18)$$

$$\dot{\tilde{C}}(t) = \alpha \cdot (\gamma \cdot C - \lambda) \quad (2.19)$$

where the value of γ defines the desired degree of utilization of the output link, while α is just a control gain parameter. λ is the current utilization of the link (in bit/s), which the authors propose to calculate upon the arrival of a packet as the ratio between the packet size and the interval between the arrival of the last two packets. The system will stabilize once $\dot{\tilde{C}}(t) = 0$, which occurs when $\gamma \cdot C = \lambda$ or, in other words, when utilization is γ .

The AVQ mechanism achieves very low queuing delay, by reducing utilization from 2 to 5% [49]. Such trade-off of suboptimal utilization for reduced

queuing latency might be desirable in some cases. Still, AVQ as all AQM techniques, has limited benefits since it is intended to be used in conjunction with Reno-family algorithms in the end-systems; therefore AVQ still suffers from unstable flow throughput, RTT-dependent fairness, and inefficiency in high bandwidth delay product networks.

2.5.3 eXplicit Congestion Control (XCC) Protocols

During the last half-decade a significant research effort has focused on congestion control mechanisms that rely on explicit multi-bit communication between the network elements, i.e. routers, queue controllers, and the end-systems. Such approach has been reported to improve throughput and network stability, latency, efficiency and convergence times when compared with the end-system and AQM approaches. Even more interesting is the fact that these mechanisms do not require any per-flow state on the network. However, the gain comes at the cost of higher deployment cost since modifications are required both to end-systems and network elements. In this section we describe and analyze the properties of some of the most important proposals in this field. It is fair to say that the approaches presented here build on the congestion control mechanisms [50], [51], [52], [53] developed by the ATM forum for the ATM Available Bit-Rate (ABR) profile [54], and on the Core Stateless Fair Queuing (CSFQ) architecture [55], which already implemented congestion control based on explicit communication between network elements.

2.5.3.1 eXplicit Control Protocol (XCP)

The eXplicit Control Protocol (XCP) [4], [56] was the first proposal of explicit signaling between end-systems and routers for congestion control purposes. It introduces a new header in the TCP/IP stack, called the *congestion header*, shown in Fig. 2.6, placed in between the network (IP) and the transport headers (e.g. TCP). This header carries information about the instantaneous flow throughput (X field) and the flow RTT, removing the need for the router to keep per-flow state. The X field actually contains the instantaneous inter-

Figure 2.6: The congestion header used by XCP.

packet interval at the sender which, used in conjunction with the packet size s , may be used to calculate the instantaneous flow throughput ($R = \frac{s}{X}$). The header contains two more fields, **Delta_Throughput** and **Reverse_Feedback**, which are used to carry the actual rate variation required by the bottleneck router.

An XCP algorithm is composed of two controllers. The Efficiency Controller (EC), responsible for the overall link utilization, and the Fairness Controller (FC), responsible for the bandwidth distribution among individual flows. The EC follows a Multiplicative Increase Multiplicative Decrease (MIMD) rule that achieves a fast convergence to the network conditions and, thus high efficiency. The FC follows an Additive Increase Multiplicative Decrease (AIMD) rule that is known to converge to fairness resulting in a max-min fair bandwidth distribution among flows.

The EC periodically calculates an amount of bandwidth F to be distributed in the following period. This period is referred to as the *control interval*, and usually it is set to the average RTT of all flows. The amount of bandwidth F is calculated as:

$$F = \alpha \cdot (C - \text{input_bw}) - \beta \cdot \frac{q}{d} \quad (2.20)$$

where *input_bw* is the incoming bandwidth, *C* is the link capacity, *q* is the length of the persistent queue and *d* is the duration of the control interval. The duration of the control interval is typically set to match the average RTT of the flows traversing this router in order to preserve stability. α and β are system constants dimensioned also to preserve system stability. The persistent queue represents the lowest queue length observed within the control interval. As said, *F* represents the amount of bandwidth that is going to be distributed in the following control interval.

The FC is responsible for deciding how *F* is divided among the flows traversing this controller. Before serving a packet from the queue, the FC calculates the portion of *F* assigned to that packet, the *delta_throughput* or, in other words, the variation of bandwidth the flow to which the packet belongs to should be applied in the next control interval. The calculation of the *delta_throughput* comes as (this is a simplified version of the equation for better comprehension):

$$\text{delta_throughput} = \begin{cases} \frac{F}{N \cdot d} \cdot \frac{s}{R} & \text{if } F > 0 \quad (\text{Additive Increase}) \\ F \cdot \frac{s}{\text{input_bw} \cdot d} & \text{if } F \leq 0 \quad (\text{Multiplicative Decrease}) \end{cases} \quad (2.21)$$

where *s* represents the packet size (*bytes*), *R* is the throughput of the flow to which the packet belongs to (*bytes/s*), and *N* is the estimate of the number of flows traversing the controller. *s* is obtained by examining the length field of the IP header [57], [58], *R* is obtained from the *X* field of the congestion header ($\frac{s}{X}$), whilst *N* is a bit trickier to obtain. Fig. 2.7 shows an example of how the XCP controller estimates the number of currently active flows. Basically it is able to identify the portion of a flow that each packet represents (using the *X* field value) and, based on that, the number of flows can be found. Eq. 2.21 above can be simplified so that it does not involve any per-packet divisions. The division operation is computationally more complex and should be avoided on a per-packet calculation. Considering that the *X*

Figure 2.7: Example of the calculation of the number of current active flows by XCP.

field of the congestion header carries the average time interval between sent packets, estimated by the source, then $X = \frac{s}{R}$. By replacing $X = \frac{s}{R}$ in Eq. 2.21, we note that no per-packet divisions are required since now X is used, and $\frac{F}{input_bw}$ is a fixed fraction calculated only once at the beginning of the control interval. In practice, the calculation of *delta_throughput* is a bit more complicated since when utilization exceeds 90%, an additional 10% of the utilized bandwidth *input_bw* is redistributed among flows. The purpose of this additional distribution is to avoid the stagnancy of the rates when utilization is high.

By enabling the router to define the bandwidth allocation, XCP achieves high efficiency, high stability, near-zero queuing delay, and one order of magnitude faster convergence times than pure end-system mechanisms. The router is able to establish such an accurate performance because it has ac-

cess to very high resolution information about the network state of congestion, and it is able to pass it to the sources. However, this mechanism has limitations and drawbacks as well. Firstly, to operate it requires that all bottlenecks and end-systems are XCP-enabled, so incremental deployment is possible but difficult. Secondly, the router is required to have knowledge about the link capacity, which may be difficult in certain type of transmission media, e.g. IEEE 802.11. Thirdly, utilization may be clamped down to a minimum of 80% [59], [60]. Because bandwidth allocation is done proactively by the router to all flows traversing it, it will force bandwidth allocation even to flows that are limited elsewhere in the network, the *alien flows*. This allocation results in a waste of bandwidth and, as said above, it can limit utilization down to 80% if the percentage of alien flows is significant. Additionally it has been reported [61] that XCP may become unstable when RTT heterogeneity among flows is high. This problem, however, is overcome if the control interval is set using the maximum flow RTT rather than the average RTT. Concluding, besides its flaws, XCP has taken a fresh approach to congestion control showing how much the accuracy of the congestion control function can be improved when very high resolution information is available.

2.5.3.2 Rate Control Protocol (RCP)

The Rate Control Protocol (RCP) [5] works similarly to XCP, differing only in the way it allocates bandwidth among flows. It emulates processor sharing of flows by calculating a common sending rate R to be used by all flows. In XCP, flows converge to the same rate smoothly over a period of time, whilst in RCP flows converge to a common rate within one control interval. The rate R is updated based on the amount of unused bandwidth and the current standing queue, as in XCP. Because the new flows converge to the common rate R within one control interval, short flows are able to complete quicker than in XCP, where those flows take more time to achieve the common rate. The rate R at control interval n is updated by the RCP controller according to the following equation:

$$R_n = R_{n-1} + \left(\frac{\alpha \cdot (C - \text{input_bw}) - \beta \cdot \frac{q}{d}}{\hat{N}} \right) \quad (2.22)$$

where $\hat{N}$ is the controller estimate of the number of currently active flows, obtained by:

$$\hat{N} = \frac{C}{R_{n-1}} \quad (2.23)$$

RCP senders use a congestion header, as in XCP. Before releasing the packet from the queue, the RCP controller fills in the feedback field with the value of R_n . Additionally, the header contains the RTT of the flow estimated by the source, so that the controller is able to calculate the average RTT of the flows. The average RTT consists in the delay of the feedback loop; it is used to clock the update cadence, maintaining stability.

RCP builds on the XCP approach, however it puts the emphasis on the metric of Average Flow Completion Time (AFCT) [62], rather than delay stability. It is a fact that RCP reduces the AFCT of short sessions, such as typical web requests, by one order magnitude (compared to Reno), or even two orders of magnitude (compared to XCP) [5], greatly improving the interactivity experienced by the users. RCP achieves this by sacrificing delay (and queue) stability, as it allows temporary capacity overload whenever the number of new flows surpasses that of the flows leaving the network. The authors argue that the effect of flows arrival will not have a major impact on the RCP system [63], in most scenarios of utilization. Another study [64] defines the limits of the surge of flows in the system for which RCP is able to maintain stable performance. RCP is also computationally less complex than XCP. No per-packet operations are required since the rate R , calculated once at the control interval timeout, is the only feedback given to the sources.

2.5.3.3 MaxNet

MaxNet [6] also builds on the XCP framework, allowing routers to communicate high precision congestion state of the network to the sources. It differs from XCP in the sense that it leaves the rate decision task to the sources.

MaxNet routers compute their congestion state $p(t)$ as the integral of the excess bottleneck utilization over time:

$$p(t + dt) = p(t) + dt \cdot \frac{y(t) - \gamma \cdot C}{C} \quad (2.24)$$

where γ represents the target utilization of the link, C is the link capacity and $y(t)$ represents the input bandwidth. Additionally $p(t) \in [0 : 1]$. This congestion state signal is communicated to sources through a congestion header. The source updates its sending rate $x(t)$ using a demand function $x(t) = D(p(t))$:

$$x(t) = x_{max} \cdot e^{\xi(t) - \alpha \cdot \frac{p(t)}{d}} \quad (2.25)$$

resulting in a decreasing exponential function with the increase of the congestion measure $p(t)$. d represents the RTT of the flow and x_{max} is the maximum source rate. $\xi(t)$ and $\alpha \cdot \frac{p(t)}{d}$ are the two members of the exponent. $\alpha \cdot \frac{p(t)}{d}$ dominates the transient response and the dynamic stability of the system, while $\xi(t)$ is responsible for the convergence to fairness on the long-run. $\xi(t)$ is further complex; however we will not expand it here the reader should refer to [65] for a more detailed description. MaxNet was shown to be stable [65] as long as the slope of the demand function does not exceed a certain threshold.

MaxNet, like RCP, reduces drastically the number of per-packet operations required. In the case of MaxNet, those operations are moved to the source, while in RCP they are simply unnecessary. MaxNet on the other hand has fewer experimentation reports, and recently it is believed to have turned commercial by FASTSOFT [66].

2.5.3.4 Coupling Logistic TCP (CLTCP)

Coupling Logistic TCP (CLTCP) [12] is another approach built on top of the XCP framework. The idea of CLTCP is to improve the stability of the congestion control system, so that the throughput of individual flows is more stable in the presence of significant variance in the rate of flow arrivals. It

achieves that by approximating the convergence of the rate of flow to its target rate by a *sigmoid* function. It is inspired by the Logistic Model [67], a model which characterizes the growth of a population over time. Initial growth of the population, when resources are plentiful, is exponential. When resources become scarce, the growth diminishes exponentially, but still the population grows, converging to a certain stable size. The result is that the size of the population in the initial phase is pretty small, and the growth, in absolute terms, is also gentle. Afterwards, a sort of explosion period occurs, and as soon as resource nears its limits (i.e. link utilization approaches 100%), the growth is quickly reduced. CLTCP has developed analytic expressions both for the router control and the source control. As in MaxNet, CLTCP moves the bandwidth allocation task to the source. The congestion signal p_n (at control interval n) is calculated by the CLTCP router using the following equation:

$$p_n = p_{n-1} \cdot e^{\beta \cdot T2 \cdot \left(1 - \frac{y_n + (q_n - q_0)/T}{C}\right)} \quad (2.26)$$

where β and T are parameters that control system dynamics and stability. y_n represents the incoming bandwidth during control interval n , and q_n is the queue length measured in that same period. q_0 is the queue length desired on steady-state, C is the link capacity and $T2$ is the sampling period duration. The value of p is then communicated to the sources, which use it in the following equation to update its sending rate x :

$$x_n = x_{n-1}^{1-\alpha \cdot d} \cdot p_{n-1}^{\alpha \cdot d} \quad (2.27)$$

where d is the flow estimated RTT, and α is a control gain parameter.

CLTCP is reported [12] to improve significantly the stability of the system in the presence of highly dynamic flow arrival distributions. This high stability is obtained by severely reducing the initial rate that flows are allowed to use. It can be regarded as the opposite idea of RCP. On the bright side it almost eliminates per-packet operations at the router, as RCP and MaxNet do.

2.5.3.5 Variable-structure congestion Control Protocol (VCP)

The Variable-structure congestion Control Protocol (VCP) [68] argues that *one more bit is enough*. This statement is based on the claim that only 3 levels (2 bits) of congestion are required to produce smooth rates, and achieve high utilization of the network. The 3 levels of congestion would be: under-utilized, slightly under-utilized, and over-utilized. The source would react differently to each of these levels of congestion. If under-utilized, the source would adopt a Multiplicative Increase (MI) strategy, resulting in an exponential growth of the sending rate during the period of under-utilization. As the link steps into the slightly under-utilized region, the source switches to an Additive Increase (AI) strategy. When the link becomes over-utilized, the source adopts the Multiplicative Decrease (MD) rule, but with a multiplicative factor less conservative than Reno.

The VCP authors have shown that VCP is stable and able to achieve a fair and efficient utilization of network resources. The drawback is revealed by the time required for flows to converge to the same rate, which may be significant [68]. The authors have shown that the convergence time was $O(P \cdot \log \Delta P)$, where P is the bandwidth delay product (BDP) per-flow, and ΔP is the maximum difference between the BDP of the different flows converging. XCP, for example, converges in $O(\log \Delta P)$, and RCP has instant convergence time - $O(1)$.

2.5.3.6 Quick-Start for TCP (QS-TCP)

Quick Start for TCP (QS-TCP) [69] takes advantage of explicit communication between routers and sources to determine the initial sending rate of the source. It can be regarded as *jump-start*, replacing the *slow start* phase of Reno. QS-TCP proposes that communication between sources and the network is done using the options field [57] of the IPv4 header, or the extension headers [58] of IPv6. The source initiates the field with the desired sending rate (which can be ∞), and the routers along a path either approve that sending rate, or they fill in the field with the maximum approved rate. This information travels back to the source in the ACK packet, which will

use this value as its initial window size. Additionally, QS-TCP implements a TTL-based mechanism to determine if all routers in a path have evaluated the bandwidth request. The source only uses the feedback received if all routers have participated in the process. The router algorithm, used to approve Quick Start requests, is not fully defined; however some guidelines can be found in [70]. These guidelines suggest the usage of the moving average of the load factor of the link, and a counter of the recently approved Quick Start request, to decide about the approval of incoming requests.

QS-TCP is an improvement that can greatly benefit short flows, reducing its Average Flow Completion Time (AFCT). Its benefits are limited to this sole metric, as most of TCP behavior is maintained.

2.6 Summary

The family of Reno algorithms is currently the major responsible for the congestion control function in the Internet. Characteristics like inefficiency in high bandwidth delay networks, throughput instability, high network latency, RTT-dependent fairness or slow convergence speed have led to a number of alternative algorithms. Some of these alternatives require only modifications of the end-systems, whilst others go further suggesting that routers take an active role in the congestion control function. End-system only algorithms are certainly easier to deploy, however they are unable to suppress all the undesired characteristics of Reno. Most of these proposals successfully improve the scalability and the throughput stability of Reno up to some degree. However only the FAST algorithm improves queuing delay [71], [42] as well, and only under some conditions. Adding AQM techniques to the control loop, helps improving fairness and network latency for loss-based algorithms; however it is hard to tune an AQM mechanism to achieve optimal performance and stable operation. The main problem with the end-system approach is that end-systems only have access to limited information. The eXplicit Congestion Control (XCC) approaches solve this problem by making it possible that network elements communicate explicitly the congestion state of the

network to the sources. With precise information about the network state, the sources become able to set their sending rate much more accurately. The XCC approach is able to achieve high efficiency regardless of the network BDP, accurate max-min fairness, near-zero queuing delay, and also fast convergence times. One of the constraints of XCC mechanisms, and of many AQM mechanisms as well, is that routers are required to know the link capacity, which limits its applicability to links with fixed, and deterministic capacity. In this thesis we dig into this problem, exploring alternative XCC algorithms that can be used when the capacity is either variable or unknown. A comparison of the currently available congestion control solutions is presented in Tables 2.1 and 2.2.

	Scalability	Queuing Delay	Fairness	Convergence Time (RTTs)
Reno	Low	High	Inaccurate RTT- dependent	Proportional to BDP
TFRC	Low	High	Inaccurate, RTT- dependent	Proportional to BDP+
FAST	High	Low to Medium	Accurate	Proportional to $\log(\text{BDP})$
CUBIC	High	High	Average	Proportional to BDP
CTCP	High	Medium	Average	Proportional to BDP
WestWood+	High	High	Inaccurate, RTT- dependent	Proportional to BDP
XCP	High	Near-Zero	Accurate	Max of ≈ 20 RTTs
RCP	High	Low to Medium	Accurate	1 RTT
VCP	High	Low	Accurate, RTT- dependent	Proportional to BDP
MaxNet	High	Near-Zero	Accurate	Max of few tens of RTTs
CL-TCP	High	Low	Accurate	Proportional to $\ln(\text{BDP}) \cdot \frac{1}{RTT}$

Table 2.1: Comparison of different congestion control mechanisms: Part 1.

	System Stability	Deployability	Security
Reno	Highly oscillatory throughput No stability proof under general conditions	Already deployed	No major vulnerabilities
TFRC	Slightly oscillatory throughput No stability proof under general conditions	Changes to end-system stack	No major vulnerabilities
FAST	Stable throughput Theoretically proven stability	Changes to end-system stack	No major vulnerabilities
CUBIC	Fairly oscillatory throughput No stability proof under general conditions	Changes to end-system stack	No major vulnerabilities
CTCP	Fairly oscillatory throughput No stability proof under general conditions	Changes to end-system stack	No major vulnerabilities
WestWood+	Fairly oscillatory throughput No stability proof under general conditions	Changes to end-system stack	No major vulnerabilities
XCP	Stable throughput Theoretically proven stability	Changes to end-system and router stack	Vulnerable to both lying receivers and lying senders
RCP	Stable throughput Theoretically proven stability	Changes to end-system and router stack	Vulnerable to lying receivers
VCP	Fairly stable throughput Theoretically proven stability	Changes to end-system and router stack	Vulnerable to lying receivers
MaxNet	Stable throughput Theoretically proven stability	Changes to end-system and router stack	Vulnerable to lying receivers
CL-TCP	Stable throughput Theoretically proven stability	Changes to end-system and router stack	Vulnerable to lying receivers

Table 2.2: Comparison of different congestion control mechanisms: Part 2.

Chapter 3

Problem Statement and Related Work

In this chapter we describe the problem of operating eXplicit Congestion Control (XCC) mechanisms in links with varying or unknown capacity. This thesis proposes to solve this problem and, by doing that, extends the benefits of XCC to this type of links. Section 3.1 presents the thesis problem and, in Section 3.2, related work in this field is introduced. The chapter is summarized in Section 3.3.

3.1 Problem Statement

First, a high level description of the problem is presented in Section 3.1.1. In Section 3.1.2 the problem is characterized analytically in greater detail; our analysis is based on two of the most important XCC mechanisms: XCP and RCP. In Section 3.1.3 our analytical analysis is validated through simulation. Finally, in Section 3.1.4, conclusions of the problem statement are drawn.

3.1.1 Characterization of the Problem

eXplicit Congestion Control (XCC) mechanisms require that routers along a communication path know the capacity of their output links. In some transmission media, namely on those providing shared-access or variable data

rates, e.g. IEEE 802.11, it may be hard for a router to have an exact estimate of the output capacity. For example, the capacity of an IEEE 802.11 medium depends on the data and basic rates used by each station, the average size of the packets being sent, the amount of transmission collisions, the handshake mechanisms used, e.g. RTS/CTS, or the wireless interference conditions. Thus it is complex for a wireless router to compute the bandwidth it can use. Moreover, the capacity is likely to change over time. For these reasons, it is likely that the router estimation of the output capacity contains an error. Intuitively, we can deduce that if the router estimate is below the actual link capacity, the link will result under-utilized while, if it is above, over-utilization of the link will occur. In fact, in the case of an over-estimation of the medium capacity - as it will be shown in Section 3.1.2 - an XCC controller is able to compensate a (limited) estimation error by building up the queue. But in any case, the performance of XCC mechanisms is severely affected by the error of the capacity estimate, which may cause XCC mechanisms to under-utilize network resources, generate large packet loss or excessive queuing delay. This section aims to answer the question *To what extent does an erroneous estimate of capacity impact the operation of XCC mechanisms?*, and also to provide the motivation for the development of alternative XCC router algorithms that are able to track the medium capacity efficiently.

3.1.2 Capacity Estimation Error

We approach the problem of operating eXplicit Congestion Control (XCC) mechanisms in variable/unknown capacity media by understanding how these mechanisms respond to a (fixed) capacity estimation error. As noted before, we focus on two of the most representative mechanisms of this class: XCP and RCP. In order to understand the impact of the capacity estimation error, the Efficiency Controller (EC) of these mechanisms is analyzed. The EC periodically calculates the amount of bandwidth F to be distributed among the active flows. F can be either positive or negative, and it is a function of the current level of utilization and of the standing queue. In each interval the EC distributes a portion α of the unused bandwidth, while draining a

portion β of the standing queue. As seen in Chapter 2, F is calculated as:

$$F(t) = \alpha \cdot (C - y(t)) - \beta \cdot \frac{q(t)}{d} \quad (3.1)$$

where d is the feedback delay (the average RTT of the flows), $y(t)$ is the bitrate of the aggregate of the incoming flows, and $q(t)$ is the queue length. In XCP, F is explicitly calculated but, in RCP, F is embedded in the calculation of the common rate R . The calculation of R , the rate given to a flow in RCP, takes the form:

$$R(t) = R(t - d) + \frac{F(t)}{\hat{N}} \quad (3.2)$$

where $\hat{N} = \frac{C}{R(t-d)}$ is the router estimate of the number of active flows, as shown in Section 2.5.3.2.

Considering that the estimation of the link capacity C contains a certain error ϵ , C can be written as:

$$C = C_{real} + \epsilon \quad (3.3)$$

where C_{real} represents the actual link capacity. Using this new definition of C , F can be written as:

$$F(t) = \alpha \cdot [C_{real} + \epsilon - y(t)] - \beta \cdot \frac{q(t)}{d} \quad (3.4)$$

The feedback system characterized by Eq. 3.4 will reach equilibrium when the incoming bandwidth matches the actual link capacity, and router feedback is null. That implies the following steady-state conditions:

$$F(t) = 0 \quad (3.5)$$

$$\dot{y}(t) = \frac{F(t-d)}{d} = 0 \quad (3.6)$$

$$C_{real} = y(t) \quad (3.7)$$

applying these conditions to Eq. 3.4, we obtain the equilibrium property:

Figure 3.1: Compensation queue q_c represented in terms of the delay introduced $\left(\frac{q_c}{C_{real}}\right)$ as a function of the relative estimation error $\left(\frac{\epsilon}{C_{real}}\right)$.

$$q_c = \frac{\alpha}{\beta} \cdot \epsilon \cdot d \quad (3.8)$$

which tells us that the queue length q will have to grow to compensate the estimation error ϵ present in C . The queue length required to compensate the capacity estimation error - the *compensation queue* or q_c - is proportional to the error ϵ itself, the ratio $\frac{\alpha}{\beta}$, and the system delay d . Note that the condition expressed by Eq. 3.8 hides the effect of queuing delay in the overall system delay d . To obtain a closed-form expression of the queue length in equilibrium, d should be described in terms of the minimum propagation delay d_0 , and the queuing delay. We present two alternatives to decompose d : 1) by using the general expression of d , or 2) by considering the series of cumulative delays, both providing the same results:

1. Using the general expression of d with $q = q_c$:

$$\begin{aligned}
 d &= d_0 + \frac{q_c}{C_{real}} \\
 d &= d_0 + \frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \cdot d \\
 d &= d_0 \cdot \frac{1}{1 - \left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \right)}
 \end{aligned} \tag{3.9}$$

2. Using a Taylor series decomposition of the partial queuing delays:

$$\begin{aligned}
 d &= d_0 + \left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \right) \cdot d_0 + \left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \right)^2 \cdot d_0 + \dots + \left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \right)^n \cdot d_0 \\
 d &= d_0 \cdot \sum_{n=0}^{\infty} \left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \right)^n \\
 d &= d_0 \cdot \frac{1}{1 - \left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \right)}, \text{ if } \frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} < 1
 \end{aligned} \tag{3.10}$$

Using the decomposition of d presented above, the compensation queue q_c can be expressed in a closed-form, as a function of the minimum propagation delay d_0 . Furthermore, it allows us to derive an additional stability condition $\left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} < 1 \right)$. Analyzing these results, we conclude that queue build-up will only be able to compensate the estimation error if $\epsilon < \frac{\beta}{\alpha} \cdot C_{real}$, as shown in Fig. 3.1. If the error ϵ exceeds this threshold, the queuing delay $\frac{q_c}{C_{real}}$, and consequently the required compensation queue q_c , will grow indefinitely, hindering the system from reaching the equilibrium. The closed-form expression of the compensation queue q_c can now be obtained by merging Eq. 3.8 and Eq. 3.9/3.10:

$$q_c = \frac{\alpha}{\beta} \cdot \epsilon \cdot d_0 \cdot \frac{1}{1 - \left(\frac{\alpha}{\beta} \cdot \frac{\epsilon}{C_{real}} \right)} \quad \text{if } \epsilon < \frac{\beta}{\alpha} \cdot C_{real} \tag{3.11}$$

The conclusion of the steady-state analysis is that XCP and RCP systems are able to compensate a capacity estimation error ϵ through queue build-up, if the error is bounded by $\epsilon < \frac{\beta}{\alpha} \cdot C_{real}$. The queue length required to compensate the error is proportional to the system minimum delay d_0 , the ratio $\frac{\alpha}{\beta}$, and it grows exponentially with the estimation error (Fig. 3.1). Our analysis results are consistent with several experimental tests [13], [14], [15],

and the theoretical analysis made in [72].

3.1.3 Simulation Results

In this section our analysis of the problem is validated using discrete event simulation techniques. For this purpose, the ns-2 simulator [73], which includes an implementation of XCP, is used. The simulation scenario consists of a network with a single bottleneck and a dumbbell topology, which is enough to assess the correctness of our problem analysis. The bottleneck link is configured with capacities of 1, 10 and 100 Mbit/s, while the bottleneck XCP router is configured with different capacities in different experiments. The difference between the link capacity and the capacity configured in the XCP router is the capacity estimation error ϵ . The propagation delay of the flows d_0 is set to 50, 100, 200, and 400 ms in different experiments. The system is tested with $\alpha = 0.4$, $\beta = 0.226$ as recommended in the original XCP proposal [4], but we also vary β to test the stability limits of the system. β is the system parameter that controls the aggressiveness with which the controller drains queue build-up.

3.1.3.1 Effect of the Capacity Estimation Error

We study the effect of the estimation error by analyzing the response of the bottleneck queue. The bottleneck link capacity is set to 10 Mbit/s, and the estimation error ϵ is varied from 500 kbit/s to 6 Mbit/s, i.e. $\frac{\epsilon}{C_{real}}$ varying from 5 to 60%. In this experiment, the values of $\alpha = 0.4$, $\beta = 0.226$ are maintained as recommended by the XCP authors. The system base delay d_0 is set to 100 ms. The theoretic maximum estimation error that the system is able to compensate through queue build-up with these settings is $\epsilon = \frac{\beta}{\alpha} \cdot C_{real} = 0.565 \cdot C_{real}$ - according to Eq. 3.11. Fig. 3.2 shows the time response of the bottleneck queue for various levels of estimation error. The compensation queue in steady state obtained through our analytical analysis is plotted as horizontal lines. The plot above (Fig. 3.2(a)) shows the response to capacity estimation error ranging from 5 to 40%, well below the stable limit (56.5%). It is possible to observe that the queue converges to

(a) Below the stable limit ($\frac{\epsilon}{C_{real}} = \frac{\beta}{\alpha} = 0.565$). (b) Near the stable limit ($\frac{\epsilon}{C_{real}} = \frac{\beta}{\alpha} = 0.565$).

Figure 3.2: Queue time response to different levels of error in the capacity estimation. Horizontal lines represent the theoretic steady-state values obtained using Eq. 3.11, while the other curves are obtained through simulation.

the length obtained using the theoretical analysis, shown in Eq. 3.11. The plot below (Fig. 3.2(b)), shows the time response of the bottleneck queue for error levels near the error limit that the system is able to compensate, that

Figure 3.3: Capacity estimation error effect on the compensation queue: comparison between simulation results and the theoretical analysis. The compensation queue q_c grows with the estimation error.

is, $\frac{\epsilon}{C_{real}} = \frac{\beta}{\alpha}$. The limit, for this experiment's settings is 56.5%, as stated above. Fig. 3.2(b) shows the response to errors of 50 and 60%. For an error of 50%, the system is stable, allowing the queue to converge, while for an error of 60% the system is not stable, and the queue grows indefinitely. These results, although not confirming the exact limit of 56.5% obtained through our analytic analysis, confirm that the limit of the estimation error that can be compensated is within the interval of]50%, 60%[. Fig. 3.3, compares the steady-state results obtained through simulation and our previous analysis. The experimental curve includes an interpolation between the marked points. The difference between the experimental results and our analysis is negligible, and it confirms that the compensation queue q_c grows with the relative estimation error $\left(\frac{\epsilon}{C_{real}}\right)$.

3.1.3.2 Delay effect

We now focus on the effect that the system base delay has on the compensation queue. For this purpose the bottleneck link capacity is set to 10 Mbit/s, and the estimation error ϵ is fixed to 2 Mbit/s, that is, $\frac{\epsilon}{C_{real}} = 20\%$. The

Figure 3.4: Queue time response in the presence of an estimation error $\frac{\epsilon}{C_{real}}$ of 20% for different system delays d_0 . Horizontal lines represent the theoretic steady-state values obtained using Eq. 3.11, while the other curves are obtained through simulation.

parameters $\alpha = 0.4$, $\beta = 0.226$ are the same used in the previous experiment. The system base delay d_0 is varied in the range of $[20, 400]$ ms. Fig. 3.4 shows the bottleneck queue response for different base delays. The steady state results obtained through our analytical analysis (Eq. 3.11) are plotted again as horizontal lines. The steady state results of these experiments are plotted in Fig. 3.5, where they are compared to the theoretical analysis. The similarity between the two resulting curves is high, confirming that the compensation queue is proportional to the system base delay.

3.1.3.3 β Effect and Stability Limits

We analyze the effect of the β parameter of feedback function F (Eq. 3.1) on the behavior of the compensation queue q_c . The β parameter controls the percentage of the queue that the controller drains in each control interval; the higher β is the more aggressive the controller will be in draining the accumulated queue length. Most of the previous experiment settings are maintained: bottleneck link bandwidth of 10 Mbit/s, estimation error of

Figure 3.5: Effect of the base delay d_0 on the compensation queue: comparison between the simulation results and the theoretical analysis (Eq. 3.11) for a constant relative capacity estimation error $\frac{\epsilon}{C_{real}} = 20\%$. The compensation queue q_c grows linearly with the system base delay d_0 .

Figure 3.6: Time response of the queue to a capacity estimation error $\frac{\epsilon}{C_{real}} = 20\%$ and β values ranging from 0.15 to 1.0.

20%, thus $\epsilon = 2$ Mbit/s, and system base delay $d_0 = 100ms$. The experiment is repeated for a number of β values, ranging from 0.05 to 1.2. Note that

Figure 3.7: Effect of the parameter β on the compensation queue: comparison between the simulation results and the theoretical analysis. The compensation queue q_c decreases with increasing β .

XCP recommends $\beta = 0.226$. Fig. 3.6 shows the queue time response for β values in the interval $[0.15, 1.0]$; the steady state theoretical values are drawn again as horizontal lines. For these values of β the queue converges to the length obtained using Eq. 3.11. Fig. 3.7 compares the steady state results obtained from this set of experiments with the theoretical analysis presented previously. The experimental results confirm that the compensation queue decreases with the increase of β , validating Eq. 3.11.

Next, the stability limits associated with β are analyzed. We test both the steady-state and the dynamic stability limits. First the steady-state stability limit, which is given by the condition of Eq. 3.11, is analyzed. That condition states that the system is able to reach equilibrium if $\beta > \alpha \cdot \frac{\epsilon}{C_{real}}$, which for this set of experiments translates to $\beta > 0.08$. The results shown on the Fig. 3.8(a) confirm this limit. The plot shows the queue time response for $\beta = 0.15, \beta = 0.10$, and $\beta = 0.05$. For $\beta = 0.15$ and $\beta = 0.10$, the queue converges to a finite value after some time; for $\beta = 0.05$, the queue grows indefinitely. Next, we test the dynamic stability limit of the system. A system is dynamically stable if, given the existence

(a) Values of β near the stable limit in steady state ($\beta = \alpha \cdot \frac{\epsilon}{C} = 0.08$).

(b) Response to a range of high-values of β . The system becomes dynamically unstable for $\beta > 1$, interestingly, the system is dynamically more robust in the presence of an estimation error.

Figure 3.8: Queue time response for different values of β and an estimation error of 20%.

an equilibrium point, the system is able to converge to that point, or to a bounded vicinity. In [28] it was shown that, for $\alpha = 0.4$, the linear model of XCP and RCP, without estimation error, was dynamically stable if $\beta \leq$

0.32. This study also showed that for the more realistic, non-linear, model of these systems, β could be much higher, and the system would still be stable. However, the conclusions of the non-linear analysis of [28] do not apply in the case of a positive capacity estimation error because the queue is always positive, thus leading to a linear behavior of the queue. The experiments show that the system with a capacity estimation error allows higher values of β than those required to ensure system stability when there is no capacity estimation error. The plot of Fig. 3.8(b) shows the queue time response for $\beta = 0.8, \beta = 1.0$, and $\beta = 1.2$. The queue is able to stabilize for $\beta = 0.8, \beta = 1.0$, while for $\beta = 1.2$ the queue oscillates heavily around its middle value. The dynamic stability limit for this experiment is, thus, inside the interval $[1.0, 1.2]$, well above the limit of 0.32, found for the linear system with no capacity estimation error. The results of this experiment are not meant to be general, i.e. valid for any delay or estimation error, as we only want to highlight that, when in the presence of a capacity estimation error, the β parameter may be set more aggressively than proposed by the XCP authors in [4]. To obtain general results, a complete analytical study, or more extensive experiments would be required. The fact that β can be set more aggressively can be explained by the role that β has in the feedback function. β controls the break of the system; when there is a positive capacity estimation error the system will be able to break with more intensity, because the system is also accelerating faster than it normally would.

3.1.3.4 Capacity Effect

In this experiment, we aim at verifying that the system capacity does not impact the queue response, as long as the estimation error is kept constant in terms of percentage of the link capacity $\frac{\epsilon}{C_{real}}$. The system base delay is set to $d_0 = 200$ ms, $\alpha = 0.4, \beta = 0.226$, and the error is fixed at 20% of the link capacity, $\epsilon = 0.2 \cdot C_{real}$. The link capacity is set to 1, 10, and 100 Mbit/s in different simulation runs. Fig. 3.9 shows the time response of the queue in the various simulation runs. In Fig. 3.10 the results obtained through simulation are plotted overlapped with our analysis results, showing

Figure 3.9: Time response of the queue to a 20% capacity estimation error with varied link capacities.

Figure 3.10: Capacity effect on the compensation queue: comparison between simulation results and the theoretical analysis. The compensation queue q_c is not affected by the system capacity as long as the error in terms of capacity percentage is kept constant.

the independence of the link capacity on the queue response. Note that the queue results are normalized to the capacity estimation error (ϵ), to mute the

effect of the estimation error in the results. The capacity estimation error had to be changed in every run in order to maintain the capacity estimation error constant, in terms of percentage of the link capacity, that is $\frac{\epsilon}{C_{real}} = constant$.

3.1.4 Problem Statement Revisited

We have characterized the problem of operating eXplicit Congestion Control (XCC) mechanisms in variable or unknown capacity media by analyzing the effect that capacity estimation errors have on the overall performance of these mechanisms. If the router estimate of the medium capacity contains an error ϵ , one of 3 situations will occur:

- If the error ϵ is negative, i.e. the estimated capacity is below the actual capacity, the medium will result under-utilized. The amount of bandwidth wasted is equal to the error ϵ .
- If the error ϵ is positive, i.e. the estimated capacity is above the actual capacity, but does not exceed the limit $\epsilon < \frac{\beta}{\alpha} \cdot C_{real}$, the bottleneck queue will build-up in order to compensate the error. This will cause a queuing delay of $\frac{q_c}{C_{real}} = \frac{\alpha}{\beta} \cdot \epsilon \cdot d_0 \cdot \frac{1}{C_{real} - \frac{\alpha}{\beta} \cdot \epsilon}$ (seconds).
- If the error ϵ is positive, i.e. the estimated capacity is above the actual capacity, and exceeds the limit $\epsilon \geq limit$, then the queue will grow indefinitely. Real-world queues are limited, so the bottleneck queue will be in permanent overflowing state, resulting both in excessive packet loss and high queuing delay. Depending on the sources reaction to packet loss, oscillatory behavior of the queue might also occur.

In a real-world scenario, for example an IEEE 802.11 based network, the operation of current XCC mechanisms would result impractical. In such networks the bandwidth fluctuates heavily and probably the 3 situations described above would occur: wasted bandwidth, high queuing delay, and excessive packet loss. In order to enable the operation of XCC mechanisms over time varying capacity media, alternative algorithms must be developed so that an XCC router is either not required to have knowledge of the medium capacity, or it is able to track the medium capacity accurately.

3.2 Related Work

In [17], Wu *et. al* propose the Wireless eXplicit Control Protocol (WXCP). It consists in an adaptation of XCP for operation in IEEE 802.11 transmission media. It solves the problem of capacity estimation in this type of media by using MAC layer properties such as the MAC idle and busy time, or the number of link layer retransmissions. In each control interval, a WXCP router distributes an amount of bandwidth F according to:

$$F = \alpha \cdot \frac{B}{n+1} - \beta \cdot \frac{q}{d} - \zeta \cdot \frac{\bar{R}}{d} \quad (3.12)$$

where B is the estimation made by the router of the available bandwidth, n is the number of active neighbor stations during the control interval, q is the persistent queue length, and $\bar{R}$ is a running average of the number of link-layer retransmissions. These parameters are obtained as follows:

1. B is an estimation of the available bandwidth based on the MAC idle (d_{free}) and busy (d_{busy}) portions of time of the control interval d . B is given by:

$$B = \frac{d_{free} \cdot bw}{d} \quad (3.13)$$

where bw represent the current output rate of the station. It is obtained by dividing the number of transmitted bytes $b_{transmitted}$ during the last time interval, by the air time d_{self} used by the station to transmit that amount of information:

$$bw = \frac{b_{transmitted}}{d_{self}} \quad (3.14)$$

Because each station uses its own output bandwidth, the WXCP algorithm provides air time fairness. This means that stations will get the same amount of air time to transmit, and the stations using higher data rates will also obtain higher output bandwidths.

2. n is the number of active neighbor stations during the last interval; it

is obtained by inspecting the MAC source address of all packets sent to the medium - even those not destined to the monitoring station - and counting the number of unique addresses.

3. q is the persistent queue length of the node, that is, the minimum length of the queue observed during the control interval.
4. $\bar{R}$ is the average number of link layer retransmissions. The authors do not specify how to obtain this value, hence we assume that it consists in a moving average of the number of link layer retransmissions performed by that station alone in each control interval.
5. α, β , and ζ - are system tunable parameters. α controls the amount of unused bandwidth that is distributed in each control interval, β controls the amount of built-up queue that is drained in each control interval, and ζ controls the portion of link layer retransmissions eliminated in each control interval. The authors propose $\alpha = 0.2$, $\beta = 0.1$, and $\zeta = 67$. However they do not provide general support nor stability proof for these values.

Additionally, WXCP proposes a pacing mechanism and a loss recovery mechanism which improve throughput smoothness and robustness to packet loss. These mechanisms, in our opinion, are minor improvements to the basic WXCP since the core of the algorithm behavior will be driven by the explicit feedback.

The WXCP feedback function is similar to that of XCP, differing on the calculation of the available bandwidth and on the inclusion of an additional congestion metric, the average number of link layer retransmissions. It consists in an alternative algorithm, designed for XCP, but applicable to other XCC protocols such as RCP, which enables the operation of XCC protocols in IEEE 802.11, a variable capacity transmission medium.

From our point of view WXCP has some limitations. Firstly, it requires stations to constantly monitor the medium and inspect the MAC header of every packet - even of the packets which are not destined to the monitoring station. Additionally, it requires the maintenance of a table with all

active sources within a control interval, which requires a table look up, and possibly an insertion operation in that table per incoming packet. These requirements add on to the computational cost of the algorithm, but also impair energy saving mechanisms, because the station has to monitor the medium constantly. Secondly, the WXCP algorithm relies on specific details of the IEEE 802.11 technology, conditioning its application to other media with unknown or variable capacity. Finally, the WXCP algorithm was tested only on a couple of limited scenarios, and no stability proof was provided by the authors. Further, extensive testing would be required to consider WXCP for deployment in real networks.

3.3 Summary

The research problem addressed in this thesis has been rigorously characterized. It consists in the operation of explicit congestion control (XCC) mechanisms in transmission media with variable or unknown capacity. An XCC router requires precise knowledge of the link capacity, otherwise the performance of the system is severely degraded. In order to fully characterize and understand our research problem, two of the most representative protocols of this category have been studied: XCP and RCP. We concluded that whenever the router estimate of the link capacity contains an error, queue build-up will occur to compensate this error. The queue required to compensate the capacity estimation error is proportional to the system delay, the estimation error itself, and to the ratio $\frac{\alpha}{\beta}$. If the error reaches a significant percentage of the link capacity, the queuing delay introduced in the path is also significant, greatly reducing the advantages of using XCC mechanisms. Furthermore, we have shown that the amount of capacity estimation error ϵ amendable through queue build-up is limited; if surpassed, this error will cause the queue to grow indefinitely, rendering the system unusable. If the estimation error is negative, i.e. the capacity estimate is below the actual link capacity ($\epsilon < 0$), the queue required to compensate the error will also be negative, which is obviously physically impossible; in that case, the

erroneous capacity estimate translates into medium under-utilization. For these reasons, algorithms that enable the promising XCC mechanisms to operate in networking environments with variable or unknown capacity, such as the popular IEEE 802.11 wireless media, are required. A first proposal for such an algorithm has been presented in [17], however it contains several limitations.

Solving this problem of XCC mechanisms will enable the operation of this class of congestion control algorithms in transmission media characterized by shared-access and variable bitrate, of which IEEE 802.11 is a popular example. Among the advantages brought by XCC are low-delay, throughput stability and scalability. The first two are important for multimedia and interactive applications, while scalability is becoming increasingly important as the network BDP continues to grow, especially in wireless transmission technologies.

Chapter 4

XCC Algorithms for Time-Varying Capacity Media

In this chapter we present three alternative and original feedback algorithms applicable to eXplicit Congestion Control (XCC) protocols. Two of these algorithms - Blind and ErrorS - are based on a novel approach which uses relative information such as queue length variation or queue accumulation over time. This approach enables an XCC router to control the link utilization simply by monitoring its queue state. The third algorithm presented, referred to as MAC, is a modified version of the Wireless eXplicit Control Protocol (WXCP), thus it is based on previous work. The MAC approach relies on parameters obtained from the Medium Access Control (MAC) layer such as the transmission and idle time periods. The modifications we introduce to WXCP have the objective of making it simpler and more practical to implement, and also to improve its convergence properties. These 3 algorithms are designed having in mind the two most relevant XCC protocols to date - XCP and RCP. However, in Section 4.4 it is explained how they can be applied to other XCC protocols, and give the concrete example of their applicability to CLTCP.

The chapter is organized as follows. In Section 4.1 we present the queue-based algorithms, Blind and ErrorS, while in Section 4.2 the MAC algorithm is introduced. In Section 4.3 the three different approaches to XCC are

compared. In Section 4.4 it is shown how Blind and ErrorS may be applied to other XCC protocols, giving the example of their applicability to CLTCP. Finally, the chapter summary is presented in Section 4.5.

4.1 The Queue Based Approach

In this section we present the Blind and the Error Suppression (ErrorS) algorithms. Both algorithms replace the core feedback router function with a function that uses only queuing properties and does not rely on the value of the link capacity. The main idea behind the Blind variant is to calculate the available bandwidth through queue speed, instead of measuring the difference between link capacity and incoming bandwidth. The ErrorS algorithm, on the other hand, infers and compensates the error present in the capacity estimation using queue accumulation properties over time.

4.1.1 The Blind Algorithm

The Blind variant proposes an alternative method for calculating available bandwidth on a link, based on queue speed. To better understand it, let us recall the aggregate feedback function used by XCP and RCP:

$$F(t) = \alpha \cdot [C - y(t)] - \beta \cdot \frac{q(t)}{d} \quad (4.1)$$

F represents the amount of bandwidth distributed by the router among sources in a control interval. During each control interval, a router distributes a portion α of the available bandwidth $C - y(t)$, while draining a fraction β of the current standing queue length. The problem is that C , the link capacity, is either unknown or variable, thus it cannot be used. The Blind algorithm takes advantage of the relationship between the available bandwidth and queue speed. This relationship can be expressed by:

$$\dot{q}(t) = y(t) - C \quad (4.2)$$

which denotes that queue speed is actually a mirror of available bandwidth. The Blind proposal introduces this directly into the feedback function F :

$$F(t) = -\alpha \cdot \dot{q}(t) - \beta \cdot \frac{q(t)}{d} \quad (4.3)$$

which successfully removes C from the router feedback function.

Before continuing, let us define clearly two terms that are fundamental to understand the rest of this section:

- We refer to **full utilization** as the system operating point where the incoming bandwidth $y(t)$ is within a small vicinity of the medium capacity C_{real} , i.e. $y(t) \approx C_{real}$. At this operating point, the persistent queue is likely to start to build-up or oscillate.
- We refer to **under utilization** as the system operating point where the incoming bandwidth is below the vicinity of the medium capacity, i.e. $y(t) < C_{real}$. At this operating point, the persistent queue is likely to be empty or at very low levels.

The feedback function shown in Eq. 4.3 is only usable when there is queue build-up, that is when the medium is near or at full utilization, otherwise queue speed is not measurable. For this reason, the router has to somehow drive the system to full utilization, and maintain it near that point of operation. The Blind algorithm achieves this by introducing a new parameter, κ , which represents the target value of queue length or, in other words, the length at which the queue will stabilize when the system reaches steady-state. In fact, κ may vary over time, thus we also refer to it as $\kappa(t)$. The introduction of $\kappa(t)$ in Eq. 4.3 is straightforward:

$$F(t) = -\alpha \cdot [\dot{q}(t) - \dot{\kappa}(t)] - \beta \cdot \frac{q(t) - \kappa(t)}{d} \quad (4.4)$$

$\kappa(t)$ is responsible for two aspects of the algorithm: 1) drive the system to the point of full link utilization, and 2) stabilizing queue length at some positive value so that queue speed can be measured from variations around that value. When driving the system to full utilization, i.e. when the system is under-utilized and the persistent queue is empty or at very low levels, $\kappa(t)$ will take

a high value so that more bandwidth is distributed; the ceiling for the value of $\kappa(t)$ will be such that it will prevent queue overflow in the subsequent control intervals. Note that, since the higher the value of $\kappa(t)$ is the more bandwidth is distributed during each control interval, a high value of $\kappa(t)$ will enable the system to reach full-utilization faster. After the link reaches full utilization, i.e. when the persistent queue starts to build-up, $\kappa(t)$ will take the minimum value required to absorb the bandwidth fluctuations. By doing this, queuing delay is kept to a minimum. During the full utilization period, $q > 0$, we set $\kappa(t)$ to match the standard deviation of the queue length, which is a good estimate of the effect of regular bandwidth fluctuations on queue length. An approximation of the standard deviation of the queue length may be obtained by averaging the deviation of the queue length to its mean value $|q(t) - \bar{q}(t)|$. In order to perform this averaging function, $\kappa(t)$ is implemented as an exponential moving average, which also helps to filter rapid, spurious behavior of the queue. The calculation of $\kappa(t)$ comes as:

$$\kappa(t) = \begin{cases} \rho \cdot |q(t) - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } q > 0 \\ \rho \cdot |Q_\chi - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } q = 0 \end{cases} \quad (4.5)$$

where ρ is the weight given to the current sample of the exponential moving average, Q_χ is the constant that controls the maximum value of $\kappa(t)$ when driving the link to full utilization, and $\kappa(t - d)$ represents the value of $\kappa(t)$ in the previous control interval. $\bar{q}(t)$ represents the exponential moving average of the queue length, calculated also using the parameter ρ :

$$\bar{q}(t) = \rho \cdot q(t) + (1 - \rho) \cdot \bar{q}(t - d) \quad (4.6)$$

where $\bar{q}(t - d)$ represents the value of the exponential moving average of the queue length in the previous control interval. The calculation of queue speed is approximated by:

$$\dot{q}(t) - \dot{\kappa}(t) \approx \frac{[q(t) - \kappa(t)] - [q(t - d) - \kappa(t - d)]}{d} \quad (4.7)$$

4.1.1.1 Increasing Robustness and Stability

Due to transient bandwidth fluctuations, the queue may be empty in some intervals, which will cause the controller to erroneously perceive under-utilization of the medium. To avoid this false positive identification of under-utilization, and the subsequent oscillations in $\kappa(t)$, due to the frequent change of the controller between the two components of Eq. 4.5, we implement a mechanism of *delayed reaction* to queue drainage. Waiting for the queue to be empty for a few control intervals before actually considering the link under-utilized greatly reduces the number of unnecessary oscillations, both in the $\kappa(t)$ function, and in the queue itself. The expression of $\kappa(t)$ including the *delayed reaction* feature becomes:

$$\kappa(t) = \begin{cases} \rho \cdot |q(t) - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } counter < \eta \\ \rho \cdot |Q_\chi - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } counter \geq \eta \end{cases} \quad (4.8)$$

where *counter* is a counter of the number of consecutive intervals during which the queue has been completely empty, and η is the threshold level for considering the link in under-utilization. Alternatively, the identification of under-utilization may be implemented using the exponential average of the queue, already calculated by the router:

$$\kappa(t) = \begin{cases} \rho \cdot |q(t) - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } \bar{q} \geq \tau \cdot \kappa(t - d) \\ \rho \cdot |Q_\chi - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } \bar{q} < \tau \cdot \kappa(t - d) \end{cases} \quad (4.9)$$

where τ is a parameter having the following approximate relationship with η :

$$\tau \approx (1 - \rho)^\eta \quad (4.10)$$

Another aspect of the algorithm that can lead to improved robustness is the type of units used to measure the queue. Considering queue length in terms of packets, rather than bytes, suits shared-access media better,

particularly in the case where the medium provides fair access in terms of packets, e.g. IEEE 802.11. A station contending for access in this type of transmission medium, when granted medium access, is only able to send one packet, not a certain amount of bytes. For this reason, specifying κ in terms of bytes is a bad choice. Imagine that the router has been queuing packets 1500 byte long, and that $\kappa = 4500$ byte. If that router starts receiving small packets (e.g. 40 byte long TCP Acknowledgments), it will allow a large number of these small packets to be queued. This number of small packets will cause large queuing delay because the router is only able to dispatch one packet when it grabs the medium for transmission (assuming the medium is saturated). We propose to measure all queue-related variables in terms of packets. The translation to bytes is performed during the calculation of F :

$$F(t) = -\alpha \cdot \dot{q}(t) \cdot s - \beta \cdot \frac{q(t) - \kappa(t)}{d} \cdot s \quad (4.11)$$

through the inclusion of s , which represents the average size of the packets currently queued. s is obtained by:

$$s = \frac{q_{bytes}}{q_{packets}} \quad (4.12)$$

The method used for measuring queue length plays a crucial role on the algorithm's performance. We adopt the strategy presented by the authors of XCP in [4]. They used the concept of *persistent queue* which consists in the lowest queue length observed during a control interval. Such measure of queue length is able to remove the noise created by the burstiness typical of traffic sent by asynchronous and window-based controlled sources. In this thesis, whenever we refer to queue length, what we actually mean is the persistent queue, and that is how it should be read.

4.1.1.2 Parameter Setting and Discussion

The Blind algorithm includes five tunable parameters: α , β , ρ , Q_χ , and τ :

1. α and β regulate the core operation of the algorithm. α controls the portion of spare bandwidth that is distributed in each control inter-

val, while β controls the fraction of the queue that is to be drained in each control interval. α and β should be as high as possible so that adaptation to network conditions is fast, while queue is rapidly drained whenever it builds-up. On the reverse side of the coin, these two parameters should be low enough so that system stability is preserved. In [4], and later in [28], the stability of XCP and RCP systems is analyzed. The first study [4], done by the XCP authors, analyses the system through its linearized model (it considers negative queue length), and provides a rule for setting α , and β so that the resulting system is provably stable. The latter study [28], done by the authors of RCP, extends this analysis by providing the full region of α , β for which the system is stable. Both studies are applicable to the Blind variant of the algorithm, mostly because it maintains the original structure of the feedback function F . The main difference in Blind is the inclusion of κ ; however, and as it is explained in the dimensioning of the parameter ρ , the low pass filter applied to κ is such that it does not impact the dynamic stability of the system. Therefore the results from previous analysis are still valid for Blind. We also adopt the set of values recommended by the XCP authors, $\alpha = 0.4$, $\beta = 0.226$, which produce a proved stable system.

The RCP authors further extend their study by analyzing the system using its non-linear model (does not consider negative queue length). They conclude that the analysis of the linearized system results in a conservative dimensioning of the parameter β , and for that reason, they propose to set β more aggressively. The analysis of the non-linear model, however, does not apply to the Blind algorithm, therefore setting β does not work for Blind. The non-linear model is not applicable because, in the Blind algorithm, queue length may actually turn negative. Not in the strict sense of course, because that is physically impossible, but because the Blind algorithm sets a positive target length - κ - for the queue; when queue is below that target queue length, it is perceived as negative. Using a more aggressive value of β , as the

non-linear analysis suggests, would impair the Blind algorithm from stabilizing queue length. This fact was confirmed in our simulation runs.

2. ρ controls the cut-off frequency of the low-pass filter applied to $\kappa(t)$. The low-pass filter is implemented through an exponential moving average, where ρ represents the weight given to the current sample of $\kappa(t)$. The purpose of applying a low-pass filter to $\kappa(t)$ is to prevent $\kappa(t)$ from interfering with system stability. One can look at $\kappa(t)$ as the moving target length of the queue. It has to move sufficiently slow so the system has time to converge to that target length before the target moves away again. To achieve that, the cut-off frequency present in the $\kappa(t)$ signal must be n times lower than that of the system output response. From [4], and as discussed in Section 2.3, the cut-off frequency of an XCP (and RCP) system is given by:

$$w_{XCP} = \frac{\beta}{\alpha \cdot d} \quad (4.13)$$

The cut-off frequency of $\kappa(t)$, w_κ , is made n times lower:

$$w_\kappa = \frac{w_{XCP}}{n} \quad (4.14)$$

where n should be as low as possible to allow fast adaptation of $\kappa(t)$ but, at the same time, sufficiently high so the queue itself has time to follow $\kappa(t)$. Fig. 4.1 shows the plot of the evolution of the queue length $q(t)$, its moving average $\bar{q}(t)$, and the queue target length $\kappa(t)$ over time, for various values of n . These results were obtained through a model of an XCP system implemented in MATLAB/Simulink [74]. We observe that the higher n is, the smoother is the evolution of $\kappa(t)$. Moreover, if $n < 2$, the queue exhibits a high level of oscillation, because $\kappa(t)$ moves too fast. For this reason, n should not be below this threshold. Using this rule, $n \geq 2$, ρ can be dimensioned. ρ is the weight given to the current sample of the moving average of $\kappa(t)$. The moving average is actually a low-pass filter, and the relation of ρ with the cut-off frequency

Figure 4.1: The dynamics of the queue $q(t)$, its moving average $\bar{q}(t)$, and the queue target $\kappa(t)$, with different low-pass filters applied to $\kappa(t)$, and $\bar{q}(t)$. **Left to right, top to bottom:** $n = 0.1, n = 1, n = 2, n = 3, n = 4$, and $n = 6$.

w_κ of the filter is given by [75]:

$$\rho = \frac{d}{d + \frac{1}{w_\kappa}} \quad (4.15)$$

Applying the rule $n \geq 2$ we can replace $w_\kappa \leq \frac{w_{XCP}}{2}$ resulting in:

$$\rho \leq \frac{1}{1 + 2 \cdot \frac{\alpha}{\beta}} \quad (4.16)$$

Using the recommended values of $\alpha = 0.4$, $\beta = 0.226$ we finally get $\rho \leq 0.22$. Using the lower bound $\rho = 0.22$ is recommended, in order to achieve the fastest adaptation possible for $\kappa(t)$.

3. Q_χ controls the maximum value of κ . When the system is under-utilized, the router cannot measure queue speed or queue variance. During this period, the router increases κ in order to maximize bandwidth distribution, and Q_χ defines the maximum amount of bandwidth to be distributed within a control interval. The maximum amount of bandwidth F distributed in each control interval is given by:

$$F = \beta \cdot \frac{Q_\chi}{d} \quad (4.17)$$

The higher Q_χ is, the faster the system will be driven to full utilization, but also the more severely the system throughput will overshoot the system capacity. This overshoot is inevitable, as it is the associated queue spike. Q_χ is dimensioned so the queue spike never exceeds a certain value Q_{max} . We treat this problem graphically, and numerically using our MATLAB implementation of the Blind algorithm control loop [76]. Fig. 4.2 shows the system trajectory where the excess utilization ($y(t) - C_{real}$) is plotted versus the queue length, for a system using the recommended values $\alpha = 0.4$, $\beta = 0.226$, and $\rho = 0.22$. A queue peak can be identified at $1.848 \cdot Q_\chi$. In order to prevent the queue from exceeding the limit Q_{max} we derive $Q_\chi \leq \frac{Q_{max}}{1.848} = 0.541 \cdot Q_{max}$, for a system with the recommended values. Fig. 4.3 (Left) shows this same limit of Q_χ , but for a wider set of values of β , and ρ . Note that ρ does not impact the results much, at least not in the region of admissible values of ρ - the line composed of triangles separates the region of admissible values of ρ (the left region of the plot) from the region of non-admissible values of ρ (the region on the right side of the plot). β , on the other hand, causes Q_χ to decrease, however that does not mean that less bandwidth is distributed, on the contrary. The plot on the right of Fig. 4.3 shows the maximum amount of bandwidth distributed in each control interval as a function of β , using the recommended values of ρ ; this plot demonstrates that bandwidth distribution actually grows with the increase of β .

Figure 4.2: Excess utilization versus queue length of a system with the recommended values $\alpha = 0.4$, $\beta = 0.226$ and $\rho = 0.22$. The worst case scenario of capacity overshoot.

Figure 4.3: **Left)** The maximum value of Q_χ expressed as a fraction of Q_{max} , for various values of β and ρ . **Right)** The maximum amount of bandwidth distributed in each control interval for different values of β , and using the recommended value of ρ .

4. τ is a threshold queue length the value of the exponential moving average of the queue needs to cross downward for the medium to be considered under-utilized. Implicitly, τ defines the number of control intervals η during which the queue needs to be empty before considering the medium under-utilized - which is the key part of the *late reaction* mechanism described in the Section 4.1.1. The relationship between η

Figure 4.4: The effect of the *late reaction* mechanism. **Left)** Without *late reaction*. **Right)** With *late reaction* ($\eta = 6$). *Late reaction* allows the queue controller to drain queue oscillations over time at the cost of an higher initial queue peak.

and τ is established by Eq. 4.10. The objective of this late reaction mechanism is to avoid spurious queue spikes or excessive oscillation caused by transient or temporary bandwidth fluctuations. For example, in some of the plots of Fig. 4.1 the queue oscillates indefinitely because the XCC controller reacts immediately whenever the queue is completely drained. In Fig. 4.4 we apply to one of those examples the *late reaction* mechanism. The oscillations due to premature identification of under-utilization are completely removed, only at the cost of slightly higher queue peaks. The usage of the *late reaction* mechanism opens up the possibility of using higher values of ρ (lower η) since *late reaction* helps reducing oscillation further. However, we suggest maintaining the recommended value of ρ to reinforce stability. This results in a conservative design choice, thereby systems requiring more responsive controllers have the freedom to increase ρ . We now explain the criteria used for choosing η . η has to be high enough to cover regular bandwidth fluctuations during the transient period but, at the same time, it should not prevent the system from detecting under-utilization timely. A reasonable assumption is to consider that typical bandwidth fluctuations oscillate at the system fundamental frequency $w_{XCP} = \frac{\beta}{\alpha \cdot d}$. The controller should wait for at least half of the period of such frequency, before considering the medium under-utilized. This

leads to:

$$\eta \geq \frac{1}{2} \cdot \frac{2\pi}{w_{XCP}} \cdot \frac{1}{d} \geq \pi \cdot \frac{\alpha}{\beta} \quad (4.18)$$

which, for the recommended values of α , β , results in $\eta \geq 5.54$. Using Eq. 4.10, with $\eta = 6$, and also the recommended value of $\rho = 0.22$, we finally get $\tau = 0.225$. The same reasoning may be applied if using other values of β, α, ρ .

4.1.2 The Error Suppression (ErrorS) Algorithm

We introduce another alternative algorithm, the Error Suppression (ErrorS) algorithm. Unlike Blind, ErrorS assumes that an estimation of the link capacity exists and, by analyzing queue accumulation over time, the controller is able to infer, and remove, the error present in the capacity estimation. With the knowledge of the approximate error present in the capacity estimation, the ErrorS controller compensates this error. The ErrorS algorithm may be particularly useful when the queue controller is able to estimate, even if not exactly, the medium capacity. It may also be useful when the medium capacity has a smaller degree of variation, while the arrival and departure of flows is still dynamic. ErrorS aims at being less responsive than Blind, allowing a smoother behavior of the queue.

In Section 3.1.2 it was shown that the error ϵ present in the capacity estimate is compensated by queue build-up, given the error satisfies $\epsilon < \frac{\beta}{\alpha} \cdot C_{real}$. Furthermore, we have established the relation between the error ϵ and the queue length q required to compensate this error (Eq. 3.8). For the reader's convenience, this relation is presented below:

$$q = \frac{\alpha}{\beta} \cdot \epsilon \cdot d \quad (4.19)$$

Solving in order to the estimation error ϵ we obtain:

$$\epsilon = \frac{\beta}{\alpha \cdot d} \cdot q \quad (4.20)$$

that tells us that queue length may be used to infer the error present in the capacity estimate. ErrorS takes advantage of this property of the system to measure and eliminate the error of the capacity estimate. It does so by introducing the evaluation of the capacity error in the router aggregate feedback formula:

$$F(t) = \alpha \cdot [C - y(t)] - \mu \cdot \xi(t) - \beta \cdot \frac{q(t)}{d} \quad (4.21)$$

where α, μ, β are system constants, $C = C_{real} + \epsilon$ is the router estimate of the medium capacity, and $\xi(t)$ represents an approximation of the capacity error ϵ , affected by the factor $\frac{\alpha}{\mu}$. We obtain $\xi(t)$ through the accumulation of the instantaneous estimation of the error:

$$\xi(t) = \frac{\beta}{\alpha} \int_0^t \frac{1}{d^2} \cdot q(t) dt \quad (4.22)$$

This equation tells us that, in each control interval, we sum up the instantaneous estimation of the error ($\epsilon = \frac{\beta}{\alpha \cdot d} \cdot q$) to the overall estimate of the capacity error $\xi(t)$. When $\xi(t) = \frac{\alpha}{\mu} \cdot \epsilon$ the error ϵ will be fully suppressed from the feedback function. The constant parameter μ is used to control the aggressiveness with which the error is suppressed from the system, and it should be dimensioned in conjunction with α, β to ensure system stability. In App. A, we present a stability analysis of the ErrorS system and provide recommended values of α, β, μ that enable system stability under general conditions.

A characteristic of the ErrorS algorithm is that, similarly to the Blind algorithm, it is usable only when queue build-up occurs, that is, when the medium is near full utilization. If the capacity estimation C is higher than the actual link capacity C_{real} , that is $\epsilon > 0$, the system will be able to reach full utilization; however, if for any reason C is below, or falls below the link capacity C_{real} , the system will not reach full utilization. For this reason, we adopt the same strategy used by the Blind algorithm: the inclusion of the variable parameter $\kappa(t)$ that represents the target length of the queue. $\kappa(t)$ will be as low as possible when the bottleneck link is fully utilized, otherwise it will increase up to a maximum value of Q_χ . This allows the

system to maximize bandwidth distribution in periods of under-utilization, while adding low queuing delay when utilization is high. The formula for updating $\kappa(t)$ is the same used in the Blind Algorithm (Eq. 4.9) which is reproduced here for the reader's convenience:

$$\kappa(t) = \begin{cases} \rho \cdot |q(t) - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } \bar{q} \geq \tau \cdot \kappa(t - d) \\ \rho \cdot |Q_\chi - \bar{q}(t)| + (1 - \rho) \cdot \kappa(t - d) & \text{if } \bar{q} < \tau \cdot \kappa(t - d) \end{cases} \quad (4.23)$$

where $\bar{q}(t)$ represents the exponential moving average of queue length, calculated as in Eq. 4.6, Q_χ is the maximum value of $\kappa(t)$, and τ is a parameter that controls the number of intervals during which the queue needs to be empty for the controller to consider the medium under-utilized. Note that the first branch of the equation of $\kappa(t)$ is, in fact, an approximation to the standard deviation of queue length. The inclusion of $\kappa(t)$ in the feedback function of ErrorS results in:

$$F(t) = \alpha \cdot (C - y(t)) - \mu \cdot \xi(t) - \beta \cdot \frac{q(t) - \kappa(t)}{d} \quad (4.24)$$

where $\xi(t)$ is now calculated as:

$$\xi(t) = \frac{\beta}{\alpha} \int_0^t \frac{1}{d^2} \cdot (q(t) - \kappa(t)) dt \quad (4.25)$$

4.1.2.1 Increasing Robustness and Stability

The mechanisms described in Section 4.1.1.1 used to increase the robustness and stability of the Blind algorithm are also applicable to ErrorS. These mechanisms include the usage of packet units rather than bytes, measuring the persistent queue length instead of the instantaneous queue length, and using the *delayed reaction* mechanism. Besides these three aspects, the ErrorS algorithm can benefit from an additional measure to improve its robustness. In ErrorS, the error present in the capacity estimate C is inferred through the integration of $q(t) - \kappa(t)$. In times of under-utilization $q(t) = 0$,

and $\kappa(t) \rightarrow Q_\chi$, therefore our accumulated error $\xi(t)$ will grow negatively to compensate the fact that the estimated value of capacity is below the actual link capacity. And that will subsist until the queue starts to build. The problem is that the system may be in under-utilization simply because there is no bandwidth demand from the sources and, in that case, the accumulated error $\xi(t)$ will grow (negatively) indefinitely. Whenever sources, in fact, demand bandwidth, the error estimation may already have grown negatively past any reasonable value. Since the router has now a highly negative error estimation it will tell sources to send much more data than the medium can transport causing a significant utilization overshoot and consequent packet loss. To avoid such situations, our error estimation $\xi(t)$ should have a limit. The criterion for setting this limit is to never allow the value of our estimated actual capacity $C - \frac{\mu}{\alpha} \cdot \xi(t)$ to exceed the amount of incoming bandwidth $y(t)$ by more than a certain *reasonable* value. We recommend to set this value to $\frac{Q_\chi}{d}$. This condition results in:

$$|\xi(t)| = \min \left(\left| \frac{\mu}{\alpha} \left(C - y(t) - \frac{Q_\chi}{d} \right) \right|, |\xi(t)| \right) \quad (4.26)$$

which, as will be shown in the next Section, does guarantee that capacity overshoot prevents the accumulated queue to exceed a certain maximum value Q_{max} .

4.1.2.2 Parameter Setting and Discussion

The ErrorS algorithm has six tunable parameters: $\alpha, \beta, \mu, \rho, Q_\chi$ and τ . These parameters, except for μ , have the same function and meaning as in the Blind algorithm although they take different values.

1. α, β , and μ are constants that regulate the weight given to each parcel in the calculation of the amount of bandwidth that is distributed by an XCC router in each control interval. α is the weight given to amount of the estimated spare bandwidth, including whatever error ϵ the estimation of capacity C may contain. β , on the other hand, controls the portion of outstanding queue bytes that are drained in

each control interval. Finally, μ controls the speed of the correction of the estimation error present in C ; the higher we set it, the faster the error will be corrected. Based on the model of the ErrorS algorithm and its stability analysis, presented in Appendix A, the values of $\alpha = 0.6, \beta = 0.32, \mu = 0.08$ are recommended. These values guarantee system stability for any value of capacity, delay or number of sources.

2. ρ has the same function as in the Blind algorithm. It controls the speed at which our target queue length $\kappa(t)$ moves. The same rationale used in the Blind algorithm (Section 4.1.1.2) may be applied to the dimensioning of ρ for the ErrorS algorithm. The difference is that the queue response of the ErrorS algorithm has a different fundamental frequency. The fundamental frequency w_{ErrorS} , as shown in Appendix A, is given by:

$$w_{ErrorS} = 1.75 \cdot \frac{\beta}{\alpha \cdot d} \quad (4.27)$$

Applying the same logic used for the dimensioning of ρ for the Blind algorithm, we now have to set the cut-off frequency of $\kappa(t)$, referred to as w_κ , to be n times lower than the fundamental frequency of oscillation of the ErrorS system:

$$w_\kappa = \frac{w_{ErrorS}}{n} \quad (4.28)$$

Using our MATLAB/Simulink model of the ErrorS algorithm, the queue response is analyzed with different values of n . The resulting plots of the queue behavior are shown in Fig. 4.5. Analyzing these plots it is possible to observe that for $n \geq 3$ the amplitude of the oscillations decreases considerably throughout time, even if not completely. For this reason our recommendation is that $n = 3$. Translating this setting into a concrete value of ρ , and using the same criteria as in the Blind algorithm design process, we can write:

Figure 4.5: The dynamics of the queue $q(t)$, its moving average $\bar{q}(t)$, and the queue target $\kappa(t)$, with different low-pass filters applied to $\kappa(t)$, and $\bar{q}(t)$ with the ErrorS algorithm. **Left to right, top to bottom:** $n = 0.1, n = 1, n = 2, n = 3, n = 4$, and $n = 6$.

$$\rho = \frac{d}{d + \frac{n}{w_{ErrorS}}} \quad (4.29)$$

$$\rho = \frac{1}{1 + \frac{3}{1.75} \cdot \frac{\alpha}{\beta}} \quad (4.30)$$

which, for the recommended values of $\alpha = 0.6\beta = 0.1817$ results in $\rho = 0.15$.

3. Q_χ controls the maximum value of κ . When the system is under-utilized, the router increases κ to maximize bandwidth distribution. The maximum amount of bandwidth F that is distributed within a

control interval is defined by the maximum value of κ , thus Q_χ :

$$F = \left(\frac{\mu}{\alpha} + 1 \right) \cdot \beta \cdot \frac{Q_\chi}{d} \quad (4.31)$$

which is obtained using Eq. 4.24 with the under-utilization conditions $C - y(t) = 0$, $q(t) = 0$, and with $\kappa(t) = Q_\chi$. The higher Q_χ is, the faster the system will be driven to full-utilization, but also the more severe will be the utilization overshoot. To dimension Q_χ we follow the same process used in the design process of the Blind algorithm. Using our MATLAB implementation [76] of the ErrorS algorithm we plot the worst-case scenario of capacity overshoot, and identify the queue peak. Fig. 4.6 shows the trajectory of the utilization error and queue length for a system using the recommended values $\alpha = 0.6$, $\beta = 0.1817$, $\mu = 0.1817$ and $\rho = 0.15$. It is possible to identify the queue peak at $2.25 \cdot Q_\chi$. Therefore, to prevent the queue from exceeding a certain limit Q_{max} , we derive $Q_\chi \leq \frac{Q_{max}}{2.25} = 0.444 \cdot Q_{max}$, for a system with the recommended values. Fig. 4.7 (Left) shows this same limit of Q_χ , but for a wider set of values of β , and ρ . Note that ρ does not impact the results much, at least not in the region of admissible values of ρ (the lower/leftmost division of the plot). β , on the other hand, causes Q_χ to decrease; however that does not mean that less bandwidth is distributed, on the contrary. The plot on the right of Fig. 4.7 shows the maximum amount of bandwidth distributed in each control interval as a function of β , using the recommended values of ρ ; this plot demonstrates that bandwidth distribution actually grows with the increase of β .

4. τ has the same function it has in the Blind algorithm - it defines the threshold length that the exponential moving average of the queue has to cross downward for the system to be considered under-utilized and implicitly, the number of control intervals η during which the queue needs to be completely drained before the medium is considered under-utilized. The idea behind this mechanism, which we refer to as *late*

Figure 4.6: Excess utilization versus queue length of a system with the recommended values $\alpha = 0.6$, $\beta = 0.1817$, $\mu = 0.1817$ and $\rho = 0.15$. The worst case scenario of capacity overshoot.

Figure 4.7: **Left)** The maximum value of Q_χ expressed as a fraction of Q_{max} , for various values of β , μ and ρ . **Right)** The maximum amount of bandwidth distributed in each control interval for different values of β , and using the recommended value of ρ .

reaction, is to avoid unnecessary oscillations due to temporary utilization fluctuation. In dimensioning η , and implicitly τ , we use the same reasoning applied in the Blind algorithm in Section 4.1.1.1. We assume η to cover half of the period of the fundamental system oscillation frequency, which is a reasonable estimate of the duration of typical response oscillations. As shown in Appendix A, the fundamental fre-

Figure 4.8: The effect of the *late reaction* mechanism. **Left)** Without *late reaction*. **Right)** With *late reaction* ($\eta = 6$).

quency w_{ErrorS} of an ErrorS system is given by:

$$w_{ErrorS} = 1.75 \cdot \frac{\beta}{\alpha \cdot d} \quad (4.32)$$

from which η can be derived:

$$\eta \geq 0.5714 \cdot \pi \cdot \frac{\alpha}{\beta} \quad (4.33)$$

which, for the recommended values of $\alpha = 0.6, \beta = 0.1817$ results in $\eta \geq 5.92$. Using Eq. 4.10, with $\eta = 6$, and also the recommended value of $\rho = 0.15$, we finally get $\tau = 0.37$. The same reasoning may be applied if using other values of β, α, ρ . Fig. 4.8 shows the actual effect of the *late reaction* mechanism. The magnitude of the oscillations decreases to null over time when the *late reaction* is used. On the downside, the initial queue peaks are slightly amplified, mainly because the controller does not react to queue build-up immediately, allowing extra queue build-up during the ramp-up phase. The usage of the *late reaction* mechanism opens up the possibility of using higher values of ρ since *late reaction* helps reducing oscillation further. However, it is suggested to maintain the recommended value of ρ to reinforce stability. This results in a conservative design choice; thereby systems requiring more responsive controllers have the freedom to increase ρ .

4.2 The Medium Access Control (MAC) Based Approach

The MAC-based approach consists in using Layer 2 information to infer the available link capacity. Based on measurements of the idle and busy periods of a station within each control interval, the XCC router is able to estimate the currently available bandwidth with some degree of accuracy. The MAC algorithm presented here is a modified version of the Wireless eXplicit Control Protocol (WXCP), identified as related work in the previous chapter. The modifications introduced, which are detailed in Section 4.2.2, intend to create a simpler, more practical and easier to implement algorithm, and remove some of the WXCP limitations such as high convergence time. However, we will not benchmark the MAC algorithm against WXCP because, at the time of writing of this thesis, there was no publicly available implementation, nor implementation level specification of WXCP.

4.2.1 The Medium Access Control (MAC) Algorithm

In the MAC algorithm, the router feedback function is modified as follows:

$$F(t) = \alpha \cdot S(t) - \beta \cdot \frac{q(t)}{d} \quad (4.34)$$

where $S(t)$ is the node estimate of the unused bandwidth, based on the MAC idle and busy periods during the last control interval; it is obtained as follows - during a control interval of duration d the transmission medium is perceived busy for d_{busy} seconds, while in the remainder of the time d_{idle} the medium is perceived as idle. The available bandwidth $S(t)$ is calculated by a node by multiplying the estimate of the medium bandwidth share $B_s(t)$ assigned to that node, by the fraction of the control interval d during which the transmission medium was idle:

$$S(t) = B_s(t) \cdot \frac{d_{idle}}{d} \quad (4.35)$$

where d_{idle} represents the time interval during which the medium was idle in the last control interval. The medium bandwidth share $B_s(t)$ currently being used by the node is given by:

$$B_s(t) = \frac{out(t)}{d_{busy}} \quad (4.36)$$

where $out(t)$ represents the amount of data, in bytes, that the station has transmitted in the last control interval d , and d_{busy} represents the time period during which the medium was perceived as being busy during that same control interval. $B_s(t)$ can be also interpreted as the total bandwidth the station would use if the medium was fully utilized.

To ensure that new stations have the opportunity to get its fair share of the medium, even when it is fully utilized ($d_{idle} = 0$), we introduce an additional bandwidth distribution parameter that is used when new stations start transmitting:

$$S(t) = B_s(t) \cdot \frac{d_{idle}}{d} + \vartheta(t) \quad (4.37)$$

with $\vartheta(t)$ given by:

$$\vartheta(t) = \begin{cases} 0 & \text{if } out(t) > 0 \\ \frac{Q_x}{d} & \text{if } out(t) \approx 0 \end{cases} \quad (4.38)$$

where Q_x has the same meaning and value as in the Blind algorithm. $\vartheta(t)$ allows nodes with new flows to use an initial fraction of the bandwidth which, otherwise would be impossible to use if the medium was fully utilized ($d_{idle} = 0 \rightarrow S(t) = 0$). After the initial phase, convergence will be ensured by the fairness properties of the medium access control mechanism, even if at a slow pace.

4.2.2 Differences to the Wireless eXplicit Control Protocol (WXCP)

There are two main differences between our simplified MAC-based approach and WXCP. The first is that, in WXCP, a node tracks the number of active

stations during a control interval, and the amount of bandwidth distributed by each station is only a portion $\frac{1}{N}$ of the total amount of available bandwidth in the transmission medium. The approach taken by WXCP avoids any utilization overshoot. However it may be too conservative as many of the active stations may not fully use their share of the medium capacity simply because they have no bandwidth demand from the application. This causes slow convergence to full utilization, which is the more severe the higher is the number of self-limited sources. Such situations may occur, for example, in ad-hoc wireless networks where typically each node is active in each interval sending routing control information to its neighbors, even if it has no data to transmit. In our simplified MAC-based approach, the number of active stations is not tracked in order to avoid this slow convergence problem; also, the computational complexity of the algorithm becomes smaller because the node does not check the uniqueness of the source address of each packet sent to the medium. The second difference between our simplified MAC-based algorithm and WXCP lies in the fact that WXCP tries to eliminate MAC-layer retransmissions, and although this seems a valid objective it is not clear that it results in increased utilization of the medium. It has not been proved that the maximum efficiency is obtained in the absence of MAC collisions, rather it is likely that the optimal operating point is achieved with a certain degree of collisions. Moreover, the optimal number of collisions should also be a function of the number of competing stations. Without this analysis, using the number of collisions to calculate the aggregate feedback may not yield any improvement, and worse, it may actually decrease the utilization of the medium.

4.3 Algorithm Comparison

The Blind, ErrorS, and MAC algorithms follow different approaches to how an XCC router may estimate the medium capacity. The Blind algorithm uses queue speed to measure the difference between the incoming bandwidth, which is known, and the medium capacity. ErrorS assumes that the router

has an (erroneous) estimate of the medium capacity and, through queue accumulation over the time, it measures and removes the error from that estimate. The MAC uses the idle time of the medium to infer how much bandwidth is available.

Blind and ErrorS rely only on the queue properties, while MAC relies both on MAC-layer information and on queue properties. By relying solely on queue information, Blind and ErrorS are able to abstract from the transmission technology details and their implementation results less complex and generally applicable to any transmission technology; however, they require some level of queue build-up to operate. This is, at the same time, a drawback and an advantage of these approaches. On the one hand queuing delay is not optimal but, on the other hand, maintaining some queue allows the algorithms to absorb bandwidth fluctuations without affecting utilization. Another characteristic of queue-based approaches is that, when the queue is empty, they have to probe the network for the available capacity which may often lead to queue spikes, and increase the level of jitter. In opposition, the MAC algorithm does not require any queue build-up to operate which enables near-zero queuing delay, but it may not yield maximum utilization of the network resources. Additionally, the MAC algorithm requires a technology-specific implementation since it must understand the MAC protocol of the underlying transmission technology, and it is computationally more complex because the transmission medium needs to be constantly monitored.

The Blind and ErrorS algorithms, while having the same queue-based approach, have distinctive rationales. The Blind approach makes no assumption about the underlying medium capacity, while ErrorS assumes that an estimate of the medium capacity exists, even if erroneous, and proceeds on correcting it over time - the error suppression mechanism. Blind reacts instantaneously to the queue variation, emphasizing responsiveness, while the error suppression mechanism of ErrorS takes into account the queue history to infer the estimation error. This fact makes ErrorS react more slowly to conditions changes, trading responsiveness for a more smooth response.

In conclusion, queue-based algorithms are more suitable for general deployment than the MAC-based algorithms due to their technology indepen-

dent approach. The Blind algorithm emphasizes responsiveness, thus it is the best choice for environments where the capacity is highly variable and abrupt changes occur. The ErrorS algorithm is a better match for networks where the bandwidth does not vary too much since, in this case, it will be able to produce less queue spikes than Blind. The MAC algorithm is useful in situations where queuing delay and jitter are aspects more important than utilization.

4.4 General Applicability to XCC Protocols

We have introduced two alternative router feedback functions that enable the operation of XCC protocols in networking environments with links and shared media with variable or unknown capacity. In the previous sections, the usage of these feedback functions in two of the most relevant mechanisms of this class, XCP and RCP, has been described. Now, we discuss the usage of concepts associated to our feedback functions in other XCC protocols. To illustrate that, the Coupled Logistic TCP (CLTCP), which takes an approach to XCC that slightly differs from XCP and RCP, is addressed. Our intent is merely to demonstrate how the concepts can be applied to other XCC mechanisms, rather than provide a detailed specification for CLTCP. For this reason it is not specified how different parameters should be tuned.

In CLTCP, routers calculate a certain congestion measure based on the link capacity C . This congestion measure is communicated to the sources through a header field placed in each packet similarly to XCP or RCP. Upon receiving the congestion measure, sources adapt their sending rate based on the congestion level of the network. According to our proposals the calculation of the congestion measure $r(t)$ at the router should be updated using:

$$\dot{r}(t) = \beta \cdot r(t) \cdot \left(1 - \frac{y(t) + \frac{q(t) - q_0}{d}}{C} \right) \quad (4.39)$$

where $y(t)$ represents incoming bandwidth, $q(t)$ is queue length over time, and q_0 is the target queue length of the controller. The router feedback

function still depends on the knowledge of the link capacity C . The Blind and the ErrorS algorithms could then be applied to CLTCP.

4.4.1 Blind Algorithm

The Blind algorithm infers the difference between link capacity C , and the incoming bandwidth $y(t)$ by measuring the queue speed, taking advantage of the relation $\dot{q}(t) = y(t) - C$. Using this relation in the router function we get:

$$\dot{r}(t) = \beta \cdot r(t) \cdot \left(1 - \frac{y(t) + \frac{q(t) - q_0}{d}}{y(t) - \dot{q}(t)} \right) \quad (4.40)$$

Adding a dynamic queue target $\kappa(t)$, as the Blind algorithm does in order to dynamically adapt to different levels of link utilization and capacity fluctuation, we finally get:

$$\dot{r}(t) = \beta \cdot r(t) \cdot \left(1 - \frac{y(t) + \frac{q(t) - \kappa(t)}{d}}{y(t) - (\dot{q}(t) - \dot{\kappa}(t))} \right) \quad (4.41)$$

that effectively removes the need for a CLTCP router to have knowledge about the link capacity.

4.4.2 ErrorS Algorithm

The ErrorS algorithm takes advantage of the router's ability to infer the error of its capacity estimate through queue build-up. In CLTCP, if the router capacity estimate contains an error such that $C = C_{real} + \epsilon$, we can infer that the queue will build-up in order to compensate the error, just as it occurs for XCP or RCP. The queue length required for a CLTCP router to compensate its capacity error estimate ϵ is:

$$q - q_0 = \epsilon \cdot d \quad (4.42)$$

which is obtained by analyzing the system in steady state, that is, when $y(t) = C$, and $\dot{q}(t) = 0$. Applying the ErrorS algorithm to the CLTCP

router function $\dot{r}(t)$ becomes:

$$\dot{r}(t) = \beta \cdot r(t) \cdot \left(1 - \frac{y(t) + \frac{q(t) - \kappa(t)}{d}}{C - \mu \cdot \xi(t)} \right) \quad (4.43)$$

where $\xi(t)$ is the accumulation of the instantaneous error estimation over time, which is controlled by the gain parameter μ . $\xi(t)$ is calculated using our knowledge of the relation between queue build-up and the capacity estimate error shown in Eq. 4.42. Noting that in ErrorS the queue target length is given by $\kappa(t)$, rather than by a fixed value q_0 , $\xi(t)$ can be written as:

$$\xi(t) = \int_0^t \frac{1}{d^2} \cdot (q(t) - \kappa(t)) dt \quad (4.44)$$

which is equivalent to the case of XCP and RCP, presented in Section 4.1.

4.5 Summary

We have introduced three alternative router algorithms that do not require an eXplicit Congestion Control (XCC) router to have knowledge of the instantaneous capacity of the transmission medium. These algorithms enable the utilization of XCC mechanisms over transmission media with variable or unknown capacity, thus enabling the advantages of XCC, such as low queuing delay, and stable and scalable throughput, to be enjoyed also in networks including time-varying capacity and shared-access links, for example, IEEE 802.11 links.

We have explored the entire design space for this problem by presenting solutions based both on queue dynamics and on MAC layer information. Two of the proposed algorithms, Blind and ErrorS, remove the requirement for the router to know the link capacity by using queue dynamic properties such as queue speed or queue accumulation to determine the available bandwidth. The third algorithm (MAC), which is based on previous work [17], relies on information obtained from the network interface card at the link layer such as the idle and busy periods of the link. It was shown how to tune the parameters used by each algorithm in order to achieve stable

operation regardless of the link capacity, path delay or number of sources. The solutions proposed considered the utilization of the algorithm in two of the most relevant XCC protocols to date - XCP and RCP - but it was also shown that they can be applied to other XCC protocols, giving the example of CLTCP. The three algorithms proposed represent the major contribution of this thesis and they allow the stable operation of XCC mechanisms in variable capacity networking environments.

The queue-based approach, taken by Blind and ErrorS, although common in AQM techniques is applied to XCC mechanisms for the first time [77]; our algorithms enable XCC to be operated without bandwidth configuration, and with self-tuning capabilities. By relying exclusively on queue information, they enable the router to abstract from the characteristics of the underlying medium, e.g. transmission overhead, resulting computationally simpler and generally applicable in future networks. The drawback of the queue-based approach is that queuing delay is not optimal; some degree of queue build-up is required in order to stabilize the system. Additionally, occasional queue peaks may occur when the router is probing the link capacity. The MAC-based approach, on the other hand, uses the link idle time to measure how far the link is from being fully utilized. By doing so, it allows the MAC algorithm to produce less queue spikes, and achieve optimal queuing delay. The drawback, in this case, is the computational complexity - the router is required to constantly monitor the transmission medium, what forces the router to have knowledge of the underlying technology details - and, as will be shown in the next Chapter, lower medium utilization - having low standing queues affects utilization.

Chapter 5

Validation and Performance Evaluation

In this chapter the proposed eXplicit Congestion Control (XCC) router algorithms are validated and evaluated through simulation and implementation. Simulations were performed for scenarios with IEEE 802.11 bottlenecks. IEEE 802.11 is an example of a multi-access transmission medium with variable rate, therefore it constitutes a valid option for validating our algorithms. Nonetheless, our queue-based algorithms should not be regarded as being IEEE 802.11-specific, as the only assumption that is made of the underlying medium is that it is able to provide fair access to the medium, that is, each station has the same probability of accessing the medium for transmission.

We performed extensive simulations of the algorithms, which we separate as follows. Section 5.1 presents the simulation results obtained using the ns-2 simulator [73] in single bottleneck scenarios, corresponding to the access-point case in real-world. Section 5.2 contains results for the case of multi-hop wireless access networks, which can be seen as the case of multiple congested 802.11 bottlenecks. Finally, in Section 5.3, the results obtained from measurements made over our test-bed, developed on Free-BSD, are presented.

The experiments were carried out with the purpose of assessing and quan-

tifying the following aspects:

1. Correctness of the protocol behavior regarding the main use cases;
2. Efficiency of the algorithms as a function of the network BDP;
3. Queuing delay associated to the algorithms;
4. Stability of the throughput;
5. Flow fairness;
6. Impact of the traffic dynamics considering, for instance, flows with different durations;
7. Gains obtained when comparing our algorithms with TCP Reno on each of the previous aspects.

5.1 Single Bottleneck Scenario

In this section the proposed algorithms are evaluated through the simulation of a single bottleneck scenario. In Section 5.1.1 the base simulation setup used for this set of experiments is described, and then the results obtained in individual experiments are presented and analyzed. In Section 5.1.2 the basic dynamics and the fundamental characteristics of each algorithm are analyzed. In Section 5.1.3 we test the algorithm's response in a networking environment with varying bandwidth. In Sections 5.1.4, 5.1.6, 5.1.5 and 5.1.7 we evaluate and compare the algorithms in terms of efficiency, queuing delay, fairness, and robustness to traffic characteristics.

5.1.1 Simulation Setup

The single bottleneck scenario on which we run our experiments is represented in Fig. 5.1. It is based on a dumb-bell topology and includes a single IEEE 802.11 bottleneck. Traffic consists of greedy flows (FTP-like) between the wireless nodes $W(i)$ and the wired nodes $N(i)$, in both directions. For

Figure 5.1: Single bottleneck simulation scenario.

the sake of simplicity we consider the flows traversing from nodes $N(i)$ to nodes $W(i)$ as downloads, while flows in the opposite direction are considered uploads. The wired links, with the exception of the link between the router R and the base station BS , have a capacity of 100 Mbit/s and a configurable latency (d_i ms). The IEEE 802.11 medium is configured with no hand-shake mechanisms (RTS/CTS), and stations use a data rate of 11 Mbit/s and a basic rate of 1 Mbit/s, unless stated otherwise. Let us remind that the data rate refers to the rate at which layer 3 data is transmitted in the medium and the basic rate refers to the rate at which layer 2 headers are transmitted. The propagation model used was the Two-Ray Ground model included in ns-2, and the default transmission and interference range parameters were used: 250 m and 500 m, respectively. The queue length of the wireless nodes, where the bottleneck occurs, is set to 60 packet if using XCP, while it is set to 80 packet if using RCP. The higher queue limit is set for RCP, because RCP causes higher queue peaks. The size of data packets is set to 1300 byte, while acknowledgment packets have 60 byte. The parameters of the algorithms are set to the recommended values, listed in Appendix B, and the value of the capacity in ErrorS, C in Eq. 4.21, is configured to 0 Mbit/s. Additionally,

Q_{max} is set to 50 packets, slightly below the actual limit of 60 packet of XCP and 80 packet of RCP, in order to reduce the probability of queue overflow. The simulations do not include packet loss due to corruption, therefore the only causes for packet loss are queue overflow and consecutive collisions or persistent interference at the medium access level.

The algorithms were implemented on top of the XCP implementation present in ns-2 [73]. The RCP mechanism itself was implemented on top of the XCP implementation. The code is publicly available in [78].

5.1.2 Basic Dynamics

The first experiment has the purpose of analyzing the dynamics and fundamental characteristics of each algorithm. To this end, we run a simple experiment where flows incrementally enter and leave the network. A total of 10 flows is set up, composed of 5 pairs of flows. Each pair of flows consists of one download flow and one upload flow. A pair of flows enters the network each 5 seconds until time $t = 20$ s, and a pair of flows leaves the network each 5 seconds from time $t = 40$ s on. The experiment has the duration of 60 seconds. Additionally, the delay of the wired paths is set to 40 ms rendering the propagation RTT (d_0) to ≈ 80 ms, if we neglect the propagation delay within the wireless hop. The configuration of this experiment allows us to observe how the algorithms respond to utilization fluctuation caused by the dynamic arrival and departure of flows, and how they converge to their stable operating point.

The results of the experiment are plotted in Fig. 5.2 and Fig. 5.3. Fig. 5.2 represents the evolution of the congestion window of all 10 flows throughout time, while Fig. 5.3 shows the evolution of the instantaneous persistent queue, its moving average, and the parameter κ at the bottleneck (BS). The instantaneous persistent queue is the minimum queue length observed in a control interval, the queue moving average ($\bar{q}$) is obtained using Eq. 4.6, and κ is calculated using Eq. 4.9. The congestion window of the flows reflects the behavior of the BDP of the network, containing information about the individual throughput of each flow but also about the end-to-end delay (in-

Figure 5.2: The evolution of the congestion window throughout time for all algorithm combinations.

cluding queuing delay) of the path. Queue dynamics, and κ , on the other hand, help us to better understand the behavior of each algorithm combination. The first conclusion that can be drawn from the analysis of Fig. 5.2 and Fig. 5.3 is that all the algorithms converge to stable operating points and can adapt to the arrival and departure of flows. The MAC algorithm is the one showing less queue length throughout time and it leads to the lowest queue spikes; however this comes at the cost of lower network utilization, as can be observed in Table 5.1, in comparison with Blind and ErrorS. This difference

Figure 5.3: The evolution of the persistent queue length throughout time for all algorithm combinations.

is explained by the fact that Blind and ErrorS maintain some degree of occupancy of the queue buffer, allowing them to maintain the network saturated persistently. From the analysis of Fig. 5.2 we can also conclude that the modifications included in the MAC algorithm (in comparison with WXCP) effectively improve the convergence time of the algorithm. This experiment also shows the differences between Blind and ErrorS. Blind is more responsive to changes in network conditions, while ErrorS favors a more smooth response. This is observable in two periods of the simulation runs that used

Table 5.1: The average throughput in the interval $[20, 40]$ s.

Algorithm	Throughput
RCP-Blind	551.9 kByte/s
XCP-ErrorS	551.7 kByte/s
XCP-Blind	550.4 kByte/s
RCP-ErrorS	548.8 kByte/s
XCP-MAC	519.3 kByte/s
RCP-MAC	517.2 kByte/s

the algorithms applied to XCP. First, in the beginning of the simulation, it is noticeable that ErrorS takes longer time than Blind to reach full network utilization due to its slower, smoother nature. During this period, ErrorS has to infer the error contained in capacity estimation with which it has been configured (which was $C = 0$ bit/s in this case). Second, in the last 20 seconds of the simulation run, when flows start leaving the network, we can see frequent queue spikes in Blind as a result of the controller probing for the newly available bandwidth freed by the departing flows. In contrast, ErrorS is able to distribute the newly available bandwidth without producing queue spikes during this phase of the simulation. By this point ErrorS has already stabilized its estimation of the capacity error and that allows the controller to have an accurate estimate of the medium bandwidth. This allows the ErrorS controller to rapidly distribute bandwidth among the active flows before under-utilization is detected, and the probing mechanism is triggered. This results in one fundamental difference between Blind and ErrorS: while Blind reacts to changes in both the medium capacity and the level of utilization, ErrorS reacts mostly to capacity changes since it is able to cope better with fluctuations of utilization.

The dynamics of the parameter $\kappa(t)$ are a core aspect of the Blind and ErrorS algorithms. The role of $\kappa(t)$ is to define the target queue length, that is the length at which the controller will stabilize queue length. Let us analyze the plots of the queue dynamics from Fig. 5.3 to gain a better understanding of the behavior of $\kappa(t)$. During periods of under-utilization $\kappa(t)$ is increased by the controller in order to maximize bandwidth distribution, while during

periods of full, or over-utilization $\kappa(t)$ should be as low as possible so that queuing delay is reduced. This is observable in the plots from Fig. 5.3, and more clearly in the experiments where XCP was used. In the beginning of the simulation run we can see that $\kappa(t)$ increases in order to drive the system to full utilization and as soon as the queue starts to build, the value of $\kappa(t)$ decreases. During the periods when the medium is saturated, $\kappa(t)$ stays just high enough to cover the standard deviation of the queue length. Upon the departure of flows, from $t = 40$ s on, a portion of the medium bandwidth becomes available and, again, $\kappa(t)$ increases in order to maximize bandwidth distribution. This is more noticeable when using the Blind algorithm for the reasons mentioned in the last paragraph.

As a final note regarding this experiment, the differences between XCP and RCP are analyzed. Such comparison is not the objective of this thesis, however it is useful in order to fully understand the results obtained. RCP, as it was explained in Chapter 2, differs from XCP in the distribution of bandwidth among flows. With RCP every flow bottlenecked at a given router uses the same flow rate R instantly, whilst in XCP the flows converge gradually to the vicinity of a certain flow rate. The observation of the plots from Fig. 5.2 and Fig. 5.3 enable us to conclude that the result of the bandwidth allocation policy used by RCP is a more oscillatory, jittery and induce higher queuing delays. This is explicable by two factors: 1) the burstiness associated with the jump-start allowed by RCP, and 2) the simultaneous control of all flows. By jump-starting, the sources are allowed to send a large amount of packets back-to-back and, inevitably, more oscillation is visible in both congestion windows and queue length. This effect could be alleviated by introducing a pacing mechanism in the sources. The fact that RCP controls all flows simultaneously through the regulation of a common rate R also adds on to the system oscillation, as it magnifies any decision errors made by the controller. When the feedback information sent by the XCC controller to the sources contains an error, which it often does (due to errors in the calculation of the average RTT, packet size granularity, and clock imprecision), such information will have an impact on all sources magnifying the effect of this error. In XCP, the feedback information is distributed differently to the sources which

Figure 5.4: Throughput measured at the base station (BS). Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.

will reduce the effect of the error as different sources may see it with opposite values.

5.1.3 Response to Variable Bandwidth

In this section we extend the study by analyzing the algorithms' response in networking environments with varying bandwidth. Two aspects are of main importance here. The first is to understand whether the algorithms can adapt timely to abrupt and sudden changes of the medium capacity, such as when the data rate used by the stations varies synchronously (Section 5.1.3.1). The second is the algorithms' response when different stations use different data rates (Section 5.1.3.2). In the last scenario, the bandwidth variation is more progressive and smooth but only at the macroscopic level, since from frame to frame the data rate being used can be one order of magnitude different.

5.1.3.1 Abrupt Bandwidth Changes

In this experiment we induce sudden and abrupt changes of the medium capacity and analyze the algorithms' response to these changes. We set

Figure 5.5: The evolution of the congestion window throughout time for all algorithm combinations. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.

10 flows starting from 10 different stations that are active throughout the whole simulation. Abrupt changes of the capacity are caused by changing the data rate of all stations synchronously. Stations begin the simulation using a data rate of 54 Mbit/s and a basic rate of 1 Mbit/s. The data rate is the transmission rate used to transmit data whilst the basic rate is the transmission date used to transmit packet headers and MAC-layer control messages. At time $t = 40$ s the data rate of all stations is changed to 11

Figure 5.6: The evolution of κ and queue dynamics throughout time. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s.

Mbit/s, and at time $t = 80$ s the data rate falls down to 2 Mbit/s. At time $t = 120$ s the data rate of the stations returns to 11 Mbit/s, and to 54 Mbit/s at time $t = 160$ s. Note that the basic rate is kept unchanged at 1 Mbit/s throughout the simulation, thereby changes in the data rate are not linearly reflected in the overall medium capacity. Assuming a fixed basic rate of 1 Mbit/s, a data rate of 54 Mbit/s results in ≈ 7.5 Mbit/s of usable bandwidth, while for a data rate of 11 Mbit/s we get ≈ 4.5 Mbit/s, and with a data rate of 2 Mbit/s the usable bandwidth is only ≈ 1.5 Mbit/s as shown in Fig. 5.4.

The evolution of the congestion window and queue dynamics are analyzed in this experiment. The congestion windows of all 10 flows are plotted in Fig. 5.5, while queue variables are shown in Fig. 5.6. The results presented in the figures show that all algorithm combinations are able to adapt to the tested levels of instantaneous bandwidth variation, being that the Blind and the MAC algorithms yield the fastest adaption. In Blind and MAC, queue overflow caused by bandwidth reductions ($t = 40$ s, $t = 80$ s) is quickly drained, while when more bandwidth becomes available ($t = 120$ s, $t = 160$ s) it is quickly utilized. In ErrorS, not only newly available bandwidth is used slower, but also the queue build-up caused by bandwidth reduction is more severe and prolonged, as shown in Fig. 5.6. This is due to the fact that ErrorS is slower to adapt than Blind or MAC, requiring a larger number of control periods to converge. The slower behavior of ErrorS is a consequence of the weight given to the error suppression variable, $\xi(t)$ (plotted in Fig. 5.7), and the weight given to queue build-up being lower than those used in the Blind and MAC algorithms. The Blind and MAC algorithms use a weight of 0.4 to the current utilization error, and a weight of 0.226 to the amount of outstanding queue length, while the ErrorS algorithm uses only a weight of 0.1817, both for the error suppression and the outstanding queue. We could design an ErrorS controller with more aggressive values, however it would be at the cost of a higher degree of oscillation of the queue, thus these are our recommended values. The Blind and the MAC algorithms perform similarly in this experiment, the only noticeable difference being the slightly larger outstanding queue required by Blind. This is expected since the Blind algorithm requires the queue to stabilize at some positive value, so that it can measure queue speed around that same value.

5.1.3.2 Robustness to Rate Heterogeneity

In this experiment we test the robustness of the algorithms to the simultaneous utilization of different data rates by different stations. The objective is to perceive how robust the algorithms are to bandwidth fluctuations at the microscopic level, that is when the medium bandwidth varies from frame

(a) XCP-ErrorS

(b) RCP-ErrorS

Figure 5.7: The evolution of the error suppression $(\frac{\alpha}{\mu} \cdot \xi(t))$ throughout time in ErrorS. Bandwidth changes occur at times $t = 40, t = 80, t = 120, t = 160$ s. Note that the capacity estimate of the router is set to $C = 0$ Mbit/s.

to frame, even if at the macroscopic view, i.e. the average over one RTT, of the medium bandwidth is stable. 5 pairs of flows are considered; each pair consists of one download and one upload flow, resulting in a total of 10 simultaneous flows, 5 downloads and 5 uploads. Every 10 seconds a new pair

Figure 5.8: Throughput measured at the base station (BS). Every 10 seconds one pair of flows leaves the network and another pair of flows enters the network. These pairs of flows use different data rates.

of flows enters the network, while another pair of flows leaves the network, leaving a constant number of 10 active flows at all times. The data rate used by the stations is always either 2 Mbit/s or 54 Mbit/s, while the data rate of the base station is set fixed to 11 Mbit/s. We chose the data rates of 2 Mbit/s and 54 Mbit/s as they are extreme values of current IEEE 802.11 standards. The data rate of the stations starting new flows is always the opposite of that used by the stations with flows leaving the network; for instance, if the

Figure 5.9: The evolution of κ and queue dynamics throughout time. Every 10 seconds one pair of flows leaves the network and another pair of flows enters the network. These pairs of flows use different data rates.

flows leaving the network were using a data rate of 54 Mbit/s, then the new flows will use a data rate of 2 Mbit/s, and vice-versa. As a result, from the 10 flows that are always active, 6 will be using one data rate and the other 4 will be using another data rate. The data rate that is used by the majority of 6 flows changes every 10 seconds, and induces an oscillatory bandwidth. The medium bandwidth is plotted in Fig. 5.8, where bandwidth behavior is represented over periods of 10 seconds. The performance of the algorithms

in this experiment can be inferred from the queue dynamics, shown in Fig. 5.9. A good algorithm shall be able to filter the noise caused by the rate heterogeneity and the instantaneous bandwidth fluctuations, by maintaining the queue as stable as possible. The results show that the MAC algorithm is the most robust in this scenario, maintaining the most stable queue when applied to both XCP and RCP. The Blind algorithm also maintains a stable queue most of the time, but when applied to RCP a few queue spikes occur. The ErrorS algorithm, on the other hand, does not deal very well with this kind of bandwidth fluctuation producing frequent queue spikes, indicating that ErrorS, as in the previous experiment, does not react well to sudden bandwidth variations.

5.1.4 Efficiency and Scalability

We focus now on efficiency and how it scales with the bandwidth delay product. The objective of the set of experiments performed were twofold: 1) to characterize the algorithms in terms of network utilization, and 2) to verify whether their performance is independent of the BDP of the network, unlike what happens in TCP Reno. Network utilization depends strongly on the characteristics of the traffic used in the experiments. For that reason, the results presented below should be interpreted carefully and generalizations shall be avoided. Regarding scalability, the same reasoning should be applied when reading absolute values. Our main objective is to prove that XCC protocols maintain their properties when the BDP of networks grows, in opposition to Reno who sees its efficiency decrease.

A mixture of traffic composed of short and long flows, both downloads and uploads, is used. Long flows are active throughout the simulation, while short flows have a duration exponentially distributed around the mean duration of 10 seconds; a minimum duration of 1 second is defined for the short flows. 30 source nodes were set up, split evenly between the wireless and the wired parts of the network. A total of 6 long flows are initiated at the beginning of the simulation, while the short flows begin at different times of the simulation run. The interval between each short flow, at each source, is

Table 5.2: The relationship between the scale factor and the MAC layer parameters.

Scale factor	Data Rate	Basic Rate	SlotTime /SIFS
1	11 Mbit/s	1 Mbit/s	20/10 μ s
2	22 Mbit/s	2 Mbit/s	10/5 μ s
5	55 Mbit/s	5 Mbit/s	4/2 μ s
10	110 Mbit/s	10 Mbit/s	2/1 μ s
20	220 Mbit/s	20 Mbit/s	1/.5 μ s

exponentially distributed around an average interval such that the average number of simultaneously active short flows is also 6, thus we have 50% of short flows, and 50% of long flows and a total of 12 simultaneously active flows, in average. Additionally, the wired paths between each wired end-system $W(n)$ and the wired router R are configured with different latencies, between 20 ms and 120 ms, thereby the flows will have propagation RTTs within the interval $[40, 220]$ ms. Two options can be considered to vary the network BDP: the delay of the wired paths can be varied, or the bandwidth of the wireless medium. We choose to vary the bandwidth of the wireless medium since it seems a more plausible scenario, similar to next-generation wireless systems (IEEE 802.11n), and because increasing the delay of the wired paths would result in unrealistic propagation delays. The bandwidth of the wireless medium is varied by changing the value of the data and basic rates of IEEE 802.11 and other related interval variables, as shown in Table 5.2. Scale factors between 1 and 20 are tested. Each simulation run lasts 200 seconds, and the results shown refer to the period $[20, 180]$ s, in order to remove the effect of the convergence periods. Additionally, each individual experiment is repeated 3 times with different random seeds.

The results of this experiment are shown in Fig. 5.10. The plot shows how utilization scales with the increase of the capacity of the wireless medium, normalized to the data rate used in each experiment. The results support the argument that Reno, due to its fixed dynamics, is not able to follow the increase of bandwidth. Even with a constant set of background flows, Reno's utilization starts to decrease when the medium bandwidth exceeds a certain threshold. With our experiment setup this threshold was ≈ 0.44 Mbit/s

Figure 5.10: Efficiency as a function of medium capacity. TCP Reno utilization decreases with the increase of the medium bandwidth, while XCC mechanisms maintain their properties regardless of the bandwidth value.

($0.4 \cdot 110$ Mbit/s) but, of course, this value depends on the traffic pattern used. An important conclusion is the following: if the number of flows and traffic pattern is kept constant, there will be a bandwidth limit above which Reno will not be able to fully utilize network resources. Other conclusions to be drawn from this experiment are the following: 1) for lower bandwidths, Reno is able to achieve better utilization than any XCC mechanism, 2) MAC-based algorithms achieve lower utilization than Blind or ErrorS, and 3) RCP outperforms XCP with this kind of traffic pattern.

The higher utilization achieved by Reno at low bandwidths is explained by two factors: 1) Reno constantly maintains a significant amount of queue build-up so it is able to compensate the bandwidth wasted by the departure of flows, and 2) the ACK-loss effect, documented in detail in [77]. The ACK-loss effect consists in the loss of ACK packets which are automatically acknowledged by subsequent ACK packets; since these losses are not detected by the senders, no retransmissions nor slowdown occurs resulting in less ACK packets being transmitted in the wireless medium, leaving more bandwidth for data packets, thus helping Reno to achieve a higher throughput. This

mechanism also has implications in terms of fairness between upload and download flows, as we will see in Section 5.1.5. In XCC, the effect of ACK-loss is muted since they have very small or no loss at all.

The difference of throughput obtained by the MAC algorithm, in comparison with Blind and ErrorS, can be explained by the difference of average queue build-up required by each algorithm. As seen in previous experiments, MAC maintains very low, near-zero queue length at all times; this means that the departure of flows or bandwidth fluctuation results in wasted bandwidth. Blind and ErrorS, on the other hand, maintain small, non-zero queue build-up at all times, which allows them to compensate small bandwidth fluctuations and the departure of flows. These results show a trade-off between queue build-up and efficiency in the presence of dynamic traffic.

The fact that RCP outperforms XCP can be explained by XCP requiring a *slow start* period for new flows, while RCP allows new flows to *jump-start* to the rate R used by flows already in the network. This allows new flows to quickly compensate for the departure of other flows in RCP.

In summary, this experiment allowed us to confirm Reno's inability to scale with the BDP, and to observe the trade-off between queue build-up and efficiency when in the presence of dynamic arrival of flows.

5.1.5 Flow Fairness

In this section we evaluate the flow fairness achieved by each of the XCC algorithms and compare them to Reno. The objective of this experiment is to show that XCC algorithms are more fair than Reno, even removing the RTT-bias that Reno implies. Additionally, it is shown that in the specific case of having shared downstream and upstream links, as it is the case of shared media such as IEEE 802.11, Reno can be grossly unfair between upstream and downstream traffic.

In the first experiment the configuration of Section 5.1.4 was replicated. A total of 6 background flows were competing with an average of 6 short flows with a mean duration of 10 seconds. The propagation RTT of the flows is uniformly distributed within the interval $[40, 220]$ ms, and the simulation

Figure 5.11: The Jain's fairness index over time. XCC algorithms vs. Reno.

runs for 200 seconds. Flow fairness was measured using the Jain's index [26], given by:

$$J = \frac{(\sum_{i=1}^n \bar{x}_i)^2}{n \cdot \sum_{i=1}^n \bar{x}_i^2} \quad (5.1)$$

where $\bar{x}_i$ is the average throughput of source i and n is the number of active sources during the interval considered to calculate the index value. In our experiment, periods of 1 second were used to calculate the index, and flows were considered only if they are active during the full interval.

Table 5.3: Jain's fairness index ranges.

	Average	5 th to 95 th percentile range
RCP-ErrorS	0.974	[0.895,0.999]
RCP-Blind	0.964	[0.889,0.998]
RCP-MAC	0.961	[0.872,0.998]
XCP-MAC	0.958	[0.870,0.999]
XCP-Blind	0.955	[0.861,0.999]
XCP-ErrorS	0.949	[0.847,0.998]
TCP Reno (DL only)	0.734	[0.474,0.891]
TCP Reno	0.291	[0.191,0.423]

The results from this experiment are shown in Fig. 5.11 and complemented in Table 5.3. The plots support our claim that XCC algorithms produce much more accurate flow fairness than Reno. There exists no significant difference between Blind, ErrorS, or MAC; even between RCP and XCP the difference is slim being that RCP produces, in average, a higher fairness index value. This is understandable since RCP provides *instantaneous* fairness, allowing new flows to immediately use the rate being used by the flows already in the network. XCP flows, in opposition, go through a convergence period in the order of tens of control intervals. Even so the difference between XCP and RCP is only of 1 to 3 % in the index score. The difference to Reno is, on the other hand, significant. There exists a specific aspect of the behavior of TCP Reno that greatly impacts fairness on IEEE 802.11 media (and on shared access media in general): the ACK-loss effect, mentioned in Section 5.1.4 and described in [77], causes major unfairness between upload and download traffic. In a scenario using Access Points, queue build-up occurs at the output queue of the base station to the wireless medium, where data packets from download flows and ACK packets from upload flows pass. Losses of data packets are always detected by the sources, while the loss of ACK packets goes unnoticed most of the times since subsequent ACK packets automatically acknowledge previously lost ACKs. In this way, sources uploading data will detect packet loss less often than sources downloading data, causing unfair bandwidth distribution among them. This is clear from Fig. 5.12 where we show the fairness index of Reno using only

Figure 5.12: The Jain's fairness index over time. The difference between Reno with download traffic only and Reno with traffic in both ways.

Table 5.4: The amount of traffic transmitted in the download and upload directions by TCP Reno.

	Amount of Traffic	(%)
Download	12797200	9.5%
Upload	122189600	90.5%

download flows (TCP-DL-ONLY), which does not suffer from the ACK-loss effect, and the fairness index of Reno with flows on both directions (TCP), suffering from the ACK-loss effect. With download flows only, the fairness index consistently outperforms Reno with flows in both directions. In Table 5.12 it is shown the amount of traffic transmitted in both directions in this experiment and, once again, it is clear how upload traffic discriminates download traffic as it accounts for more than 90% of the bandwidth.

Next, the dependence of the flow fairness on the RTT is tested. The same simulation setup is used but now with only background flows, which are active throughout the simulation. Ten separate flows are created traversing different wired paths with different latencies; the propagation RTTs of the flows are uniformly distributed within the interval $[40, 220]$ ms. The results from this experiment are presented in Fig. 5.13 showing that the throughput

Figure 5.13: The amount of traffic transported by flows with different RTTs.

achieved is inversely proportional to the RTT with Reno. This is a well known characteristic of Reno, since its long-term throughput is a function of the inverse of the RTT [33]. XCC algorithms, on the other hand, suffer little impact from RTT heterogeneity. In our results it is somewhat visible a bias against flows with shorter RTTs, however we believe this is caused by the rounding error when using packet units. For example, if a flow has a fair congestion window size of 5.34 packets, its congestion window will be rounded to 5 packets, and this error is most noticeable the smaller the congestion window is. Since flows with smaller RTTs have a smaller congestion window they are the most affected. Apart from this aspect, XCC algorithms show a clear improvement over Reno regarding RTT-biased fairness.

5.1.6 Queuing Delay

The latency introduced by queue build-up adds to the overall latency perceived by end-systems which has an impact on the interactivity of applications. In this experiment we evaluate and compare the queuing delay introduced by XCC mechanisms and TCP Reno. A fixed propagation RTT of ≈ 80 ms is configured by setting the delay of all wired links between

(a) Static Flows

(b) Dynamic Arrival of Flows

Figure 5.14: The probability density function of the RTTs measured by the traffic sources. These measurements include the propagation RTT (≈ 80 ms) plus queuing delay.

nodes $W(n)$ and the wired router R to 40 ms, and test two traffic scenarios: one with only background flows, which are active throughout the whole simulation; another scenario which, in addition to the background flows, also includes the dynamic arrival of short flows. In both experiments the average

Table 5.5: The average value and the 95th percentile of the persistent queue length, presented in packets, and as a % of the queue buffer size. For easier comparison of the results we have considered the buffer size to be 60 packet in all experiments.

	Static Traffic Experiment		Dynamic Traffic Experiment	
	Average	95 th Percentile	Average	95 th Percentile
XCP-MAC	1.55 (2.6%)	8 (13%)	1.56 (2.6%)	6 (10%)
RCP-MAC	2.88 (4.8%)	11 (18%)	2.52 (4.2%)	10 (17%)
XCP-ErrorS	3.34 (5.6%)	11 (18%)	2.41 (4%)	8 (13%)
RCP-ErrorS	5.04 (8.4%)	13 (22%)	2.97 (5%)	12 (20%)
XCP-Blind	5.51 (9.2%)	11 (18%)	3.63 (6.1%)	10 (17%)
RCP-Blind	6.01 (10%)	14 (23%)	4.67 (7.8%)	12 (20%)
TCP Reno	38.2 (64%)	56 (93%)	33.6 (56%)	53 (88%)

number of active flows is set to 12, and the duration of the short flows is exponentially distributed with a mean value of 10 seconds.

The results are shown in Fig. 5.14 where the Probability Density Function (PDF) of the RTTs, as measured by the senders, are plotted. Fig. 5.14(a) relates to the simulation run having only background flows, while Fig. 5.14(b) corresponds to the simulation run with a mixture of background and short flows. Table 5.5 presents the statistics of the plots for better comparison. This experiment shows that XCC algorithms, by maintaining a small queue, introduce less queuing delay than Reno. The queuing delay introduced by Reno is essentially tied to the queue buffer size, which for this set of simulations was set to 60 packets. Within XCC algorithms, it is observable that the MAC algorithm, as expected, is the algorithm that introduces the smaller queuing delay. By stabilizing queue length at some small positive length, the Blind and ErrorS algorithms introduce always an additional latency in comparison to the MAC algorithm. Blind induces higher queuing delay than ErrorS, and also more queuing delay is introduced by RCP than by XCP mechanisms. In the presence of dynamic traffic (Fig. 5.14(b)), the average queuing delay is lower than in the experiment with background flows only. The fluctuations caused by the arrival and departure of flows provokes the queue to be drained more frequently, thus reducing the average queue length.

Figure 5.15: Network utilization as a function of the percentage of short flows.

Summarizing, the latency reduction obtained by XCC algorithms is significant in comparison with Reno, while the extra queuing delay introduced by Blind and ErrorS is quite small in comparison with MAC, even in the presence of dynamic traffic.

5.1.7 Robustness to Traffic Pattern

The characteristics of the traffic may impact the performance of a given algorithm. In this experiment XCC algorithms are evaluated with respect to the dynamics of the flows, namely their arrival and departure processes. Each algorithm was evaluated against a varying level of short flows (mice). The simulation setup of Section 5.1.4 was used, varying the amount of short flows, having these short flows durations exponentially distributed with a mean duration of 10 seconds. Simulations with 0%, 16.7%, 33%, 50%, 66.7%, 83.3%, and 100% of short flows were executed, having 12 simultaneous flows in average.

The results obtained are shown in Fig. 5.15, where we can see that efficiency remains unchanged with the traffic pattern, supporting the conclusion

that the tested mechanisms perform well even in the presence of highly dynamic traffic. These results confirm those obtained in Section 5.1.4, where the Blind and ErrorS did outperform the MAC algorithm in terms of network utilization, and that Reno achieves higher throughput than the XCC mechanisms in low BDP networking environments.

5.2 Multiple Bottleneck Scenario

The results shown in the previous section were obtained based on a single bottleneck scenario. In this section the validation is extended to the case of multiple bottlenecks. A multiple bottleneck scenario consists of several routers along a path having the same link bandwidth, thus causing concurrent control of the flows by the different routers. In this case, some overlap is expected between controlling routers along a path and this might have an impact on the accuracy of the system response. In addition, multi-hop wireless networking scenarios are a particularly challenging transmission environment; wireless interference, and the hidden, exposed, or capture node problems become significant in these networking scenarios, and are responsible for an increased rate of unsuccessful transmissions. In this section we study the basic dynamic characteristics of the proposed XCC mechanisms in these networking scenarios with the purpose of understanding if they are still able to operate, and to what extent, if any, their performance is affected.

5.2.1 Simulation Setup

In order to create a multiple bottleneck simulation scenario we extended the scenario used in the single bottleneck experiments, replacing the access point of Fig. 5.1 by a wireless mesh access network, shown in Fig. 5.16. The wireless mesh network consists of a grid of 5×3 nodes with IEEE 802.11 interfaces, with a spacing between the nodes slightly lower than the transmission range, that is 250 meter. Two wireless clients are connected at each edge of the mesh network, which establish connections to the end-systems on the wired network. The groups of wireless clients attached to the mesh edges

Figure 5.16: Multiple bottleneck simulation scenario.

are represented in Fig. 5.16 as $T1, T2, T3, R1, R2, R3, B1, B2, B3$. The communication between the access mesh network and the wired network is made through the gateway BS , and the routing protocol used in the wireless part of the network is DSDV [79]. The IEEE 802.11 interfaces are configured with parameter values different than those used in the single bottleneck experiment in order to achieve a similar overall throughput. As shown in [80], [81], [82], the capacity of multi-hop wireless networks decreases approximately with $O\left(\frac{1}{\sqrt{n}}\right)$ which, for our specific scenario, means applying a scale factor of 5 to the rate parameters of IEEE 802.11. The values of the IEEE 802.11 parameters used in this simulation are shown in Table. 5.6. The other simulation parameters have the same values used in the single hop scenario. The

Table 5.6: The relationship between the scale factor and the MAC layer parameters.

Scale factor	Data Rate	Basic Rate	SlotTime /SIFS
5	55 Mbit/s	5 Mbit/s	4/2 μ s

queue buffer is set to 60 packet when using XCP or Reno, while it is set to 80 packet if RCP is used. The Q_{max} parameter is set to 50 packet, while the remaining XCC parameters are set to their recommended values, presented in Appendix B. The size of data packets is set to 1300 byte as in the single-hop experiment. Additionally, no hand-shake mechanisms (RTS/CTS) were used. The only source of packet loss in this experiment was queue overflow, consecutive collisions at the MAC layer and wireless interference. The interference range was configured by default to 500 meter, twice the transmission range of 250 meter.

5.2.2 Basic Dynamics

Two experiments were made, similar to those executed in a single bottleneck scenario (Section 5.1.2). The first experiment consists in setting up flows over a simple chain of nodes in the mesh grid whilst, in the second experiment, the flows are set up across the whole mesh network grid. The simple chain setup is better suited to study the effect of multiple bottlenecks since all nodes of the chain have the same traffic load. The mesh access grid (second experiment) is probably a more realistic scenario of actual deployment, but less appropriate to study the effect of multiple bottlenecks. With the mesh access grid, the nodes across the grid will have different loads and the bottleneck will ultimately be located at the mesh access gateway BS . The grid scenario is, however, where MAC-layer problems such as the hidden-node [83] or interference [84] will be more severe, thus it enables the characterization of these problems in XCC algorithms. The two experiments are configured as follows:

- In the chain experiment flows traverse a chain of nodes from the wired nodes $N(i)$ to the wireless clients of group $T1$ only. The wireless chain

through which the data will flow is composed solely of the upper-left-most nodes of the mesh grid between the mesh gateway BS and the group of clients $T1$. 3 flows are initiated to each of the client nodes in $T1$ every 20 seconds resulting in 18 active flows after 40 seconds of simulation time. The simulation lasts 120 seconds and flows leave the network in groups of 6 flows at $t = 80, t = 100, t = 120$ s.

- In the grid experiment flows are set up from the wired nodes $N(i)$ to the wireless clients in all groups: $T1, T2, T3, R1, R2, R3, B1, B2, B3$. The entire grid of wireless nodes will be used. The simulation also runs for a total of 120 seconds, and each 20 seconds a set of 6 flows enters the network. At time $t = 0$ s 1 flow starts to each of the 2 clients in groups $T1, R1, B1$, at time $t = 20$ s another flow starts to each of the clients in groups $T2, R2, B2$ and finally a flow is initiated to each of the clients in groups $T3, R3, B3$ at time $t = 40$ s. This results that 6 flows start at $t = 0$ s, another 6 flows start at $t = 20$ s and, finally, a set of 6 flows starts at $t = 40$ s, similarly to the chain experiment. The flows also leave the network in blocks of 6 flows at $t = 80, t = 100, t = 120$ s.

In these experiments the reaction to losses of XCP and RCP has been turned off, that is, XCP and RCP use exclusively the explicit information they receive from routers to adjust the sending rate, ignoring any packet loss for this matter. The reason for turning this option off is the amount of failed transmissions at the MAC layer, which is significant; so we opted to isolate the algorithms from this effect in order to evaluate simply the operation due to explicit feedback.

The objective of these two experiments is to characterize quantitatively the impact of running the proposed XCC algorithms over paths with multiple bottlenecks, that is, how their response stability gets affected by the control overlap experienced in paths in such conditions. In order to quantify the response stability we recur to the measurement of the congestion window. As noted in previous sections, the congestion window metric is particularly useful as it reflects variations of both throughput and queuing delay (thus queue length itself). In order to characterize these variations we use the

Figure 5.17: The evolution of the congestion window throughout time in the chain scenario.

standard deviation of the congestion window, restricted to the interval $[45, 80]$ s - the period during which all flows are active. In Table 5.7 the results of the standard deviation of the congestion window of all flows is presented, while Fig. 5.17, and 5.18 present the actual evolution of the congestion window of each flow over time. The standard deviation values shown in Table 5.7 are expressed as a percentage of the mean congestion window (of all flows) in the interval in question. For the sake of simplicity, the calculations are performed per acknowledgment packet rather than periodically.

Figure 5.18: The evolution of the congestion window throughout time in the grid scenario.

There are three immediate conclusions to be drawn: 1) the existence of multiple bottlenecks and multiple 802.11 hops causes XCC algorithms to exhibit more oscillation; 2) RCP is more sensitive to this effect, and 3) the ErrorS algorithm holds its characteristic of being the most oscillatory algorithm, although it reflects a particular sensitivity to the multiple-hop/bottleneck effect when compared with the other algorithms. Let us now elaborate a bit more on these conclusions:

- Multiple bottlenecks cause high oscillations in the bandwidth used by

Table 5.7: The standard deviation (in %) of the congestion window of all flows during the interval $[45, 80]$ s.

Algorithm	Single-Hop	Wireless Chain	Wireless Grid
XCP-Blind	8.2%	14.2%	17.8%
XCP-ErrorS	10.5%	18.0%	23.2%
XCP-MAC	8.6%	15.7%	17.1%
RCP-Blind	11.7%	25.2%	26.2%
RCP-ErrorS	14.1%	24.1%	28.3%
RCP-MAC	11.2%	20.7%	23.2%
TCP-Reno	49.4%	70.2%	56.1%

the flows. We could observe that although the overall response was similar to the single-hop case, the standard deviation of the congestion window increases with the number of concurrent bottlenecks. This effect was more noticeable in the grid scenario than in the chain scenario, somehow contradicting our claim that the chain scenario would present the effect of multiple bottlenecks more clearly. The fact is that, in the grid scenario, other aspects of IEEE 802.11 come into play and actually their effect adds to the overall response oscillation. Such aspects include the hidden-node [83] and the interference [84] problems that greatly reduce the probability of a successful transmission in IEEE 802.11 and, additionally, tend to discriminate nodes in weaker positions in the grid.

- RCP is more sensitive to the existence of multiple bottlenecks. It was a clear conclusion from the standard deviation measurements of the congestion window. The degree of variation obtained when running the proposed algorithms applied to RCP was significantly higher than when applied to XCP. This is consistent with what was observed in the single bottleneck experiments where RCP has shown higher degree of oscillation in comparison with XCP. This behavior is a consequence of the burstier nature of RCP, and also of the simultaneous control of the flows performed by RCP. When the feedback given to sources contains an error, it will have a cumulative effect on each source. XCP

is the example of the opposite: each packet is given a different feedback value, and different decision errors will be spread among different traffic sources as white noise, reducing the effect of the sum of all errors.

- ErrorS produces the most oscillatory behavior. The measurement of the congestion window standard deviation denoted that ErrorS produces a more oscillatory response, which, again, is consistent with the results obtained in the single-hop scenario, and a result of the dynamic characteristics of ErrorS and of the control constants chosen for the algorithm.

The results from this experiment allowed us to conclude that the proposed XCC algorithms maintain a stable behavior in multiple bottleneck scenarios; however a higher degree of oscillation of the response is expected. It was also possible to confirm that RCP, as well as the ErrorS algorithm, were the most affected by the control overlap resulting from the existence of multiple bottlenecks. Finally, we could identify that not only multiple bottlenecks impact the performance of the algorithms, but also some properties of IEEE 802.11 such as the hidden-node problem or the interference phenomena have an impact on the increase of the response oscillation in wireless multi-hop scenarios.

5.3 Experimental Test-Bed

In our final set of experiments we aim at providing proof of the feasibility of the proposed algorithms in real-world systems. For that matter, the Blind and the ErrorS algorithms were implemented in the FreeBSD system. The experiments in no way intend to be exhaustive, but simply provide a proof-of-concept. Due to time constraints the Blind and ErrorS algorithms were implemented only in combination with XCP. The implementation was done on top of the XCP implementation from the Information Sciences Institute (ISI) [85], which is publicly available. There is however a limitation regarding this implementation that consists in the fact that the bottleneck has to be

Figure 5.19: The test-bed setup. Flows traverse from node R to the wireless clients $W(i)$.

artificially created. That is, a virtual queue needs to be created on top of the actual queue and it has to be configured with a lower capacity than that of the medium to which the queue will be attached. As a result we will not be able to perform tests with a saturated IEEE 802.11 medium. Still, the results from this experiment serve to validate the algorithm design but at the same time they should be interpreted conservatively.

5.3.1 Test-Bed Setup

We use a laboratory setup (Fig. 5.19) that resembles the scenario used in our single bottleneck evaluation simulation scenario. Due to physical constraints, the number of end-systems participating in the experiments was limited to 5 wireless clients $W(i)$, 1 base station BS , and 1 wired host R , from where all flows are generated. The base station BS is connected to the wired host R

directly using a 100 MBit/s Ethernet connection and an artificial delay of 40 ms was introduced in this path using the IP Firewall (IPFW) [86] of FreeBSD. The wireless clients $W(i)$ are connected to the base station through IEEE 802.11 Network Interface Cards (NICs). The base station itself used an IEEE 802.11 NIC configured in AP mode using the hostap driver [87]. The wireless nodes were configured to use a data rate of 24 Mbit/s. The virtual queues on the nodes are configured using the Packet Filter (PF) FreeBSD interface [88]. On the wireless nodes the virtual queues are configured with a capacity of 5 Mbit/s unless stated otherwise, whilst the virtual queue of the wired host R is setup with a capacity of 90 Mbit/s. We set the physical limit of the buffer of the virtual queues to 80 packet, and define a more conservative target maximum queue length of $Q_{max} = 50$ packet. The maximum value of $\kappa(t)$, that is Q_χ , is set according to the recommended values leading to $Q_\chi \approx 40$ kByte in both Blind and ErrorS. Additionally, the minimum value of $\kappa(t)$ is defined as 4 packet - or 6 kByte considering a packet size of 1500 kByte - which represents approximately 5% of the total size of the queue buffer. The values of the rest of the parameters of Blind and ErrorS are set to the recommended values described in Appendix B. The maximum send and receive socket buffers were set to 128 kByte, which for our experiments was high enough so it was never completely utilized.

5.3.2 Basic Dynamics

In this experiment we aim to analyze and evaluate the basic dynamics of the Blind and ErrorS algorithms applied to XCP. The experiment setup is similar to that of the simulation experiments presented in Section 5.1.2. A total of 5 flows are set up whereas one of the flows is initiated every 10 seconds, until $t = 40$ s. Then, from $t = 60$ s on, the flows start to leave the network, again one flow every 10 seconds. This flow pattern allows us to understand how the network redistributes bandwidth when new flows arrive and also how the bandwidth freed by the flows departure is allocated. The results are shown in Fig. 5.20, and Fig. 5.21. Fig. 5.20 refers to the congestion windows of all 5 flows, which reflect the BDP of the flows. The plots in Fig. 5.21

(a) XCP-Blind

(b) XCP-ErrorS

Figure 5.20: The evolution of the congestion window throughout time using our FreeBSD testbed.

present the evolution of queue length, its moving average, and the parameter $\kappa(t)$ over time. The results show that both Blind and ErrorS produce an accurate response, arguably even more accurate than that obtained through simulation. Such behavior is probably a consequence of not being able to

(a) XCP-Blind

(b) XCP-ErrorS

Figure 5.21: The evolution of the queue length, its moving average and the parameter $\kappa(t)$ using our FreeBSD testbed.

use the wireless medium near its saturation point where bandwidth fluctuations are more relevant. The results show also that ErrorS produces more oscillation than Blind during transient response, which is consistent with the results obtained through simulation. The analysis of the queue behavior

from Fig. 5.21 allows us to better characterize the differences between Blind and ErrorS. When using Blind, whenever a flow leaves the network freeing a portion of the bandwidth, the queue controller quickly increases $\kappa(t)$ to increase bandwidth distribution and a consequent queue spike occurs. Such spike is also noticeable in the congestion window as a result of the increased queuing delay. In contrast, ErrorS copes with the departure of flows without generating any queue spikes as we could already observe in our simulation-based experiments. This provides additional support to the argument that ErrorS performs better with the dynamic arrival and departure of flows than Blind. One aspect that was not visible in this experiment was the slower convergence of ErrorS. Contrary to our simulation-based results, the initial ramp-up period of ErrorS observable in Fig. 5.20 was rather short. This happens because the ErrorS controller was running for some time before the flows started, allowing the controller to build part of the error estimate before the arrival of the first flow. It would be impractical to perform this experiment and prevent this from happening, therefore we leave the analysis of ErrorS convergence time to the next section.

5.3.3 Response to Variable Bandwidth

In this experiment we analyze the queue controller response to abrupt bandwidth changes for both the Blind and ErrorS algorithms applied to XCP. The objective of the experiment is to understand and compare the responsiveness of the adaptation of Blind and ErrorS by analyzing how fast the queue gets drained upon bandwidth reduction, and how long it takes for the queue to build whenever the bandwidth increases.

The experiment setup is similar to that presented in Section 5.1.3. A total of 5 long-lived flows are initiated at $t = 0$ s and, at $t = 20, 40, 80, 100$ s the bottleneck bandwidth is changed. The bottleneck queue, which is configured using the FreeBSD PF interface, is initially configured with a capacity of 10 Mbit/s; afterwards it is reduced to 5 Mbit/s, and then further reduced to 1.5 Mbit/s. At $t = 60$ s the bottleneck capacity is increased to 5 Mbit/s and finally the capacity is set to 10 Mbit/s at $t = 100$ s. Traces of the congestion

(a) XCP-Blind

(b) XCP-ErrorS

Figure 5.22: The evolution of the congestion window throughout time using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.

windows of the 5 flows are shown in Fig. 5.22, and the queue length, its moving average and the variable $\kappa(t)$ at the bottleneck, are presented in Fig. 5.23. As a reference, we plot the total data throughput measured at the bottleneck queue in Fig. 5.24.

(a) XCP-Blind

(b) XCP-ErrorS

Figure 5.23: The evolution of the queue length, its moving average and the parameter $\kappa(t)$ using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.

From the results plotted in Fig. 5.22 it becomes evident that ErrorS has difficulties in addressing abrupt bandwidth changes, what is demonstrated by the large queue build-up caused by the bandwidth reduction from 5 Mbit/s

(a) XCP-Blind

(b) XCP-ErrorS

Figure 5.24: Data throughput throughout time using our FreeBSD testbed. Bandwidth varies abruptly at $t = 20, 40, 60, 80$ s.

to 1.5 Mbit/s at $t = 40$ s. For example, ErrorS takes about 5 s to clear the queue after the bandwidth contracts from 10 Mbit/s to 5 Mbit/s, at $t = 20$ s, and more than 10 s when the reduction is from 5 Mbit/s to 1.5 Mbit/s at $t = 40$ s. In comparison, Blind deals with bandwidth reduction effortlessly as

	Blind	ErrorS	MAC	Reno
Average Queue Length (buffer %)	5 - 10%	5 - 10%	0 - 5%	50 - 100%
Fairness (Jain Index)	0.85 - 1 (XCP) 0.9 - 1 (RCP)	0.85 - 1 (XCP) 0.9 - 1 (RCP)	0.85 - 1 (XCP) 0.9 - 1 (RCP)	0.2 - 0.4
Utilization ($\frac{throughput}{data_rate}$)	≈ 0.35 (XCP) ≈ 0.37 (RCP) (any BDP)	≈ 0.35 (XCP) ≈ 0.37 (RCP) (any BDP)	≈ 0.32 (XCP) ≈ 0.33 (RCP) (any BDP)	≈ 0.41 (low-BDP) < 0.3 (high-BDP)

Table 5.8: Comparison of the Blind, ErrorS and MAC algorithms and TCP Reno.

only short queue spikes are noticeable after bandwidth reduction at $t = 20, 40$ s, taking ≈ 10 RTTs to resume the queue behavior. Again the results are consistent with those obtained in our ns-2 simulations and also confirm the fact that ErrorS exhibits slower convergence properties, thus struggling to perform in the presence of abrupt bandwidth changes. Slow convergence is more problematic in the case of bandwidth reductions since a large amount of packets may be dropped and the recovery from queue build-up is harmed by the increased system delay that queue build-up causes; the system delay controls the pace of the queue controller, thus when the delay increases the pace of the controller slows down. In the case of bandwidth increase, the slower convergence of ErrorS causes temporary lower network utilization but, since the queuing delay does not increase in this situation, the pace of the queue controller does not slow down, and utilization and queue length return to their target values in less than 3-4 s.

5.4 Summary

We have evaluated extensively the Blind, ErrorS and MAC algorithms both through simulation and a FreeBSD-based implementation. The algorithms were tested in single bottleneck as well as multiple bottleneck scenarios based on the IEEE 802.11 technology, which is characterized by having variable

bandwidth and also shared medium access. The results have shown that Blind, ErrorS and the MAC algorithms are able to provide stable performance in these scenarios, when applied to both XCP and RCP. Furthermore, they were able to maintain low queuing delay, increased flow fairness, and high utilization regardless of the bandwidth-delay product of the network when compared with TCP Reno (see Table 5.8). The differences among Blind, ErrorS, and MAC have also been highlighted. The MAC algorithm is able to operate with less queuing delay and less queue spikes than Blind and ErrorS at the cost of utilization, which becomes about 90% of the utilization obtained by Blind or ErrorS. We have also shown that ErrorS has a convergence time slower than Blind when exposed to abrupt bandwidth variations; from these, bandwidth reductions have the most impact on the ErrorS performance. On the other hand, ErrorS deals better with the arrival and departure of flows when compared with Blind, which enables ErrorS to avoid queue spikes when flows leave the network. Another conclusion from our experiments is that RCP exhibits a higher degree of oscillation than XCP regardless of the algorithm applied, which we believe is in part caused by the burstiness allowed by RCP *jump-start* property. Certainly, the RCP response could be smoothed if the sources implement any sort of output pacing mechanism.

Chapter 6

Deployment and Open Issues

The steps envisaged towards the incremental deployment of XCC mechanisms, as well as the open issues related to XCC operation are discussed in this chapter. The main disadvantage of XCC mechanisms is its deployment cost, since both end-systems and network elements require modifications. Moreover, the performance gains are only fully visible when every participating system is upgraded, and that reduces the incentive for early-adopters which, in turn, poses the whole deployment process in risk. Therefore, it is of great importance that an incremental deployment plan exists to enable XCC in the Internet.

In Section 6.1 we discuss how XCC mechanisms might co-exist with other congestion control mechanisms, namely TCP Reno. Next, we shed light on how incremental deployment of XCC can be achieved, discussing how non-XCC routers (Section 6.2) and non-XCC hosts (Section 6.3) can be integrated in an XCC system. In Section 6.4 we focus on how an XCC end-systems should react to packet loss. In Sections 6.5 and 6.6 we debate security implications of XCC and how IP-security and tunneling mechanisms (e.g. IPSec, IP tunnels, MPLS) affect the operation of XCC protocols. In Section 6.7 we discuss the utilization of pacing mechanisms at the sources to improve the overall stability of XCC systems. Finally, we summarize this chapter in Section 6.8.

6.1 Co-existence With Other Congestion Control Mechanisms

The integration of XCC mechanisms with other congestion control mechanisms is not trivial, and there are two major problems in this co-existence: 1) different mechanisms handle the same congestion signals differently; 2) each protocol has different levels of aggressiveness causing unfair bandwidth allocation. Let us take TCP Reno as example - congestion is only perceived by Reno when a packet gets dropped; therefore Reno allows queue build-up up to that point. If XCP (or RCP) run simultaneously with Reno, XCP will try to drain queue build-up by slowing down the traffic sources, while Reno will keep indefinitely pushing more data into the network until the queue overflows. This will cause Reno to monopolize the bottleneck bandwidth.

One approach to solve the problem of the co-existence of XCC controllers with traditional control mechanisms consists in **using separate queues** for XCC-enabled packets and for packets using other types of congestion control. This would allow both congestion control mechanisms to share the bottleneck link without affecting each other's behavior. The separate queues can be implemented as two virtual queues, one for XCC traffic and another for non-XCC traffic; but they can also be implemented using only one virtual queue, assigned to non-XCC traffic, and letting the XCC controller handle the physical queue directly. For the sake of simplicity, we assume that the separate queues are implemented as two distinct virtual queues. The difficulty of using separate queues for XCC and non-XCC traffic is to decide which portion of the bottleneck bandwidth to assign to each queue, in order to enable fairness (although a preferential treatment to XCC could be given in order to spur its adoption). In order to divide the bandwidth fairly between the two virtual queues, the router would have to know the percentage of flows using XCC. An XCC controller already estimates the number of flows traversing it (in the case of XCP) or, the number of flows that are bottlenecked by that controller (in the case of RCP). The hard part of the solution is to estimate the number of non-XCC flows. One solution would be

to maintain state about the active flows; however that would not be scalable. A more reasonable approach would be to quantify the number of non-XCC flows without keeping track of the active flows.

Let us exemplify how the number of non-XCC flows could be estimated when XCC flows share the bottleneck with TCP Reno flows. When in the congestion avoidance phase, each Reno flow source will average a throughput that can be approximated by [33]:

$$T = \frac{s}{RTT \cdot \sqrt{\frac{2p}{3}} + t_{RTO} \cdot \left(3\sqrt{\frac{3p}{8}} \cdot p \cdot (1 + 32p^2)\right)} \quad (6.1)$$

where s is the average packet size, p is the packet drop probability, RTT is the feedback delay, and t_{RTO} is the retransmission timeout. s and p can be directly measured by the router, while the RTT can be approximated by the average RTT of the XCC flows, which is known by the XCC controller. t_{RTO} is typically set to $4 \cdot RTT$, so we can also approximate it using the average delay of the XCC flows. Using Eq. 6.1 the XCC router is able to calculate the average throughput of the non-XCC flows and, with that, know the approximate number of non-XCC flows by dividing the aggregate bandwidth of all non-XCC flows by the average Reno flow rate T .

Having an estimate of the number of non-XCC flows, and an estimate of the number of XCC flows, the XCC controller is able to distribute the bandwidth dynamically and with approximate fairness among the XCC and the non-XCC queues, enabling the co-existence of XCC and traditional congestion control mechanisms.

6.2 Non-XCC Routers

The XCC mechanisms will not surely be widely deployed in a single step. On the contrary, they should be adopted first by closed networks which, throughout time, will start communicating between themselves. XCC-capable networks may then act as overlays in the Internet. For this reason, communication along a path is likely to traverse both XCC-capable and non-capable

routers. If the bottleneck router is not XCC-capable, then the system will not work, probably resulting in excessive packet loss. The following approaches might be followed to mitigate this problem:

- **Estimating conditions between XCC routers** is one possible approach. An XCC router will be peering with other XCC nodes through a set non-XCC of paths. The XCC router may estimate the conditions of the paths linking it to its peer XCC routers using a bandwidth measurement technique [89], [90], [91]. The XCC router will then be able to create one independent virtual queue for each of its peering XCC routers. A description of such an approach is presented in [92]. The tricky part of this scheme is how to keep track of the bandwidth of the non-XCC path, assuming it changes over time, without constantly filling the network with probing traffic. Probably the best way of monitoring this bandwidth would be to create a new feedback loop between neighboring XCC routers allowing them to exchange information periodically about the amount of data transmitted through the non-XCC path, possibly using an algorithm similar to Blind or ErrorS. This would allow an XCC router to determine the bandwidth of the non-XCC paths linking it other XCC routers without filling those paths with probe traffic.
- **Explicit signaling**, which is already available in an XCC system through the XCC header, may also be used to discover XCC capabilities along a path. For instance, each XCC router may increment a field of the header in the first packet of a connection. Upon receiving the packet, the receiver compares the counter of the field in the XCC header with, for example, the TTL field of the IP header. The difference between those two values determines the number of non-XCC routers along the path. This technique, however, does not take into account Layer 2 equipment such as switches, and nor does it allow the end-systems to identify if the non-XCC nodes are the bottlenecks of the path.

- **Packet loss** may also be used to infer that a bottleneck is not XCC-capable. If the the bottleneck is not XCC-capable, the sources will exceed the bottleneck bandwidth causing packet loss. The point here is in distinguishing what is *normal* packet loss (e.g. transmission errors, temporary bandwidth overflow), and what is non-XCC bottleneck induced packet loss. Defining a threshold level of packet loss up from which the source could determine that the bottleneck router of the communication path is not XCC-capable would allow the source to revert to a non-XCC mechanism. *Normal* packet loss rates can be obtained with some confidence from Internet monitoring reports; typical average values are in the order of a few percentage points. This approach is not fail-proof, but it would prevent XCC sources from causing large packet loss in the network when the bottleneck router is not XCC-enabled.

6.3 Non-XCC Hosts

As we do not expect that routers are upgraded in one-step, we also do not expect that all end-systems become XCC-capable at the same time. Still, if part of the network has that capability, it is possible to provide some degree of performance improvement to the end-systems by enabling routers to act as proxies. There is some early work in this field [14], where a Performance Enhancement Proxy (PEP) scheme is analyzed. In this scheme, XCC access routers split the end-to-end connection in two parts: one between the non-XCC end-systems and their access routers, and another between the two access XCC routers. The connection between the two XCC routers is controlled by XCC algorithms and semantics, while the end-systems use regular TCP. The access XCC routers control the pace at which the acknowledgments sent by the receiver reach the sender in order to control its rate. The router may even generate acknowledgment packets on behalf of the receiver in order to make the RTT of a path look smaller, thus reducing the BDP of the network; this approach requires the router to maintain the data packets for which it has generated an acknowledgment in cache until the actual ac-

knowledge sent by the receiver arrives at the router. Obviously, this is of limited applicability (e.g. some satellite applications) as it does not scale for many flows.

6.4 Reaction to Packet Loss

”How an XCC source should react to packet loss is still undefined”. This is what the XCP specification [93] states about loss but, as a temporary solution, it recommends that a source detecting packet loss should revert to traditional end-to-end congestion control. This approach provides a mechanism to automatically detect non-XCC bottlenecks or even malfunctioning XCC bottlenecks, but it would also remove the benefits of running XCC mechanisms on networks where packet corruption is frequent (e.g. wireless, high bandwidth optical networks). With RCP, reacting to packet loss in such a way would be even more problematic, since RCP allows frequent bandwidth overflow. Possible approaches to this issue are:

- **Revert** to the traditional congestion control mechanisms **only if abnormal levels of packet loss are detected**, where the *abnormal* level needs to be quantified. This approach would help not only in detecting non-XCC bottlenecks, thus facilitating XCC’s widespread deployment, but it would also provide a fall-back solution in case an XCC router exhibits an erroneous behavior.
- **Assume that a path’s XCC-capability can be determined** and, once all nodes within that path are known to be XCC-capable (or at least the end-systems and the bottleneck nodes) let sources **ignore packet loss**. This approach would show better performance in heavy corruption-based loss scenarios, but it would lack resistance to unpredictable errors such as implementations bugs or hardware failures, since it would leave all control to the router.
- Making sources react to packet loss (e.g. halving their rate upon loss detection), but **allowing them to resume to XCC after the loss**;

this would allow sources to react to packet loss retaining some robustness against malfunctioning XCC controllers. The solution however, is not optimal and it could lead to problems if the bottleneck controller is not XCC-capable, causing both queue length and utilization oscillation.

- Other techniques [94] currently under study, related to the **characterization of packet loss** causes, might also be applied.

Summing it all, we believe that ignoring the loss sources will compromise the robust behavior of the congestion control algorithm. The best approach, from our point of view, would be an XCC source to revert to other form of congestion control (e.g. TCP Reno) if significant and persistent packet loss occurs. Defining formally significant and persistent packet loss remains a topic for future work.

6.5 Attacks and Security

With the introduction of explicit communication between the network and the end-systems several attacks become easier to perform, but at the same time policing end-system behavior is also simplified. Below we list the possible attacks to an XCC system and possible counter-measures to each of them. Some of the listed issues are also discussed and analyzed in [95].

- **Receiver inflates feedback.** A receiver may inflate the feedback value in order to receive data faster. This attack will cause the receiver to have access to an unfair bandwidth share, and it has the potential to cause some degree of capacity overshooting. This is an important attack since there is a strong incentive to the receiver, i.e. receive data faster, and it is easier to perform than in traditional congestion control protocols [96].

One counter measure to this type of attack is to monitor the feedback loop near the receiver. A scalable manner of monitoring the loop would be monitored by maintaining a per-flow counter where all feedback values in the forward path would be added, and the feedback values in the

reverse path would decrement the counter. If the counter turns negative, then the receiver must be inflating the feedback values, and some punitive measure can be taken. Of course, this should be done at the network edge where the number of flows is manageable and the traffic is seen in both ways. To increase scalability, the counters could be maintained for aggregates of flows, and only when an anomaly is detected the counters of the anomalous aggregate should be decomposed into individual flow counters (or smaller flow aggregates counters). There are other feasible approaches, such as Re-ECN [97] however they require modifications in order to be applied to XCC protocols.

- **Sender reports erroneous RTT.** An XCC router relies on the RTT estimation reported by the senders to time the feedback control. By sending an erroneous RTT estimate, a sender may affect the performance of the control loop. In [95] it was shown that when a large number (more than half) of senders lie about their RTT, the utilization of the bottleneck link may decrease 50%. This attack is as a sort of partial Denial-of-Service (DoS) attack.

A possible way of controlling this attack is to perform the RTT estimation at the router, but this is a complex operation. A more feasible approach would be to limit the RTT estimate, both the overall average and the individual estimates reported by senders, within a reasonable interval. Doing so, would also limit the effect of the attack. Additionally, the truthfulness of the reported RTTs could be policed near the network edge, even if only for a small fraction of the flows.

- **Sender ignores feedback.** An XCC sender may ignore the feedback and use an arbitrary rate. This is the equivalent to a sender using UDP in today's Internet. Also, it is no different from the case of a TCP sender which ignores packet loss, increasing the congestion window indefinitely.

What is different from TCP is that XCC mechanisms provide a framework for the network to police input flows. For a TCP Reno flow, a

policer must track the RTT as well as the packet drop probability in order to calculate the flow throughput. When using XCC mechanisms, the throughput is carried explicitly in the acknowledgment packets, thus the policer's task is much simplified.

- **Sender reports lower throughput.** This is an attack that can only be performed when using XCP. In other XCC protocols the sender is not required to declare its throughput. By reporting a lower throughput a flow source will masquerade itself as the source of N flows, where $N = \frac{\text{throughput}_{\text{fair}}}{\text{throughput}_{\text{reported}}}$. This will cause the bottleneck router to allocate bandwidth to this flow as to an aggregate of N flows, thus unfairly increasing the flow throughput.

Detecting this behavior is straightforward, as well as it is to police the flow of a lying sender since each packet carries the value of the flow throughput. By applying this value in the configuration of the flow policer, e.g. token bucket filter (TBF), ensures that the flow uses only the bandwidth it declares, when there are such policers.

- **Man-in-the-middle attack.** Any node through which the traffic flows by can alter the XCC fields of the packet header, or simply drop the traffic.

We do not see any counter measure to this type of attacks. In fact, we do need the cooperation of intermediate nodes to forward the traffic to the destination. Without that cooperation, end-to-end communication is simply not possible.

6.6 Virtual Private Networks, Tunnels, MPLS, and IPsec

The traversal of Virtual Private Networks (VPN), IP-Tunnels, MPLS or any type of networks that encapsulates the original IP packet, requires special treatment of the XCC header. The XCC header, which is placed between

the network (IP) layer and the transport layer, should be processed in every network element, more specifically in every queue of the network. If the original packet is encapsulated, the XCC header should be copied in between the new network layer and the beginning of the encapsulated packet, so that the XCC header is clearly placed and can be processed by the network elements reading the encapsulated packet. When leaving the network segment that performs packet encapsulation, the XCC header needs to be copied back to its place, overwriting the previous values.

The case of IPSec is even more complicated. IPSec encrypts the packet payload above the IP layer, which obviously makes the XCC header unreadable by XCC routers. In order to use IPSec in conjunction with XCC mechanisms, IPSec needs to be modified. The XCC header needs to go unencrypted through the network, and for that purpose IPSec needs to recognize the XCC header. Alternatively, the XCC header could be used as an IPv4 option, or an IPv6 extension header, which are already ignored by IPSec. If IPSec is used in tunnel mode, the router needs to copy the inner XCC header to the outer packet encapsulation, so that nodes within the tunnel are able to read and write in the XCC header.

6.7 Pacing Mechanisms

We have observed a significant degree of oscillation of the proposed algorithms applied to RCP, particularly when there are only a small (up to 4) number of active flows. This comes as a result of the large number of back-to-back packets that the jump-start characteristic of RCP allows to send. The amount of packets that are sent back-to-back is the more significant the higher the congestion window of each flow; thereby the effect becomes more visible when there are only a few active flows.

By applying pacing mechanisms the sources are able to reduce their burstiness level, thus producing a smoother response. Implementing pacing at the source is a conceptually simple process, but that is limited by the hardware capabilities, specifically by the accuracy of the operating sys-

tem clock. For instance, it is straightforward for the sender to calculate an inter-packet interval given its congestion window, RTT, and the packet size. However, for the end-system to actually use that inter-packet interval its system clock must have an accuracy lower than that interval, something that often is not true. For this reason, the source may have to approximate this procedure by sending blocks of packets each time the timer expires. Alternatively, a source could use the arrival of ACK packets as the clock to send new packets, only limiting the number of packets that could be sent in each interval. This alternative would be a light pacing mechanism, with the potential to increase accuracy since ACK arrival would function as an external clock with unlimited precision (but also a will of its own). Regardless of how it is implemented, pacing is an important feature to maintain system stability when the congestion window grows.

6.8 Summary

In this chapter we have discussed how XCC mechanisms can be incrementally deployed in the Internet and the issues XCC will face when deployed.

Regarding incremental deployment, XCC will have to co-exist with other congestion control approaches (e.g. TCP Reno). It is also likely that XCC will be deployed first in *networking islands*, which may constitute an overlay on top of the Internet. To co-exist with other forms of congestion control, XCC routers will have to maintain separate queues for XCC and non-XCC traffic. To enable XCC islands to form an overlay, XCC routers will have to treat the non-XCC paths between XCC routers as individual links, and to do so they have to monitor and probe those paths in an active manner. In any case, XCC sources have to determine if the bottleneck link of a path is XCC-capable or not. Monitoring packet loss might be a way to do so, being that if a high level of packet loss is detected there is a high probability the bottleneck is not XCC-enabled. Alternatively, an explicit detection mechanism may be used during connection setup to determine the XCC capability of the path.

Regarding the practical issues XCC faces when deployed, the security

implications of XCC are the most relevant. Allowing the end-systems to explicitly modify the rate of flows makes it easier for malicious end-systems to obtain an unfair bandwidth share. However, XCC also enables the network policing elements to control and identify these malicious flows; at least, XCC can detect them more easily than pure end-to-end congestion control mechanisms, such as TCP Reno.

Chapter 7

Conclusion

We conclude this thesis by reviewing the work carried out, summarizing its original contributions, characterizing their limitations, and recommending topics for future work.

7.1 Review of Work

We have addressed the problem of operating eXplicit Congestion Control (XCC) protocols on media with time-varying capacity. The thesis main contribution is a set of router algorithms that enables the extension of the benefits of XCC to networks composed by media with variable, unknown, or difficult to measure capacity, e.g. IEEE 802.11. It was shown that the algorithms proposed are able to improve the network performance in terms of throughput stability, latency, utilization, and flow fairness. The characterization of the problem presented in this thesis has also contributed to a better understanding of XCC systems and their feedback control loop, providing ground for further work in this area.

In Chapter 2 we provided an overview of the current state-of-the-art in congestion control for IP networks. We analyzed pure end-to-end mechanisms, active queue management techniques, and XCC approaches. We concluded that XCC produces the best results and it is the only approach that provides, at the same time, high utilization, low queuing delay, accurate

fairness, throughput stability, and rapid convergence.

In Chapter 3 we have analyzed the problem faced by XCC approaches when used over shared-access or other types of transmission media where the capacity is variable, or simply difficult to measure. By analyzing the XCC control loop, we showed that an XCC router is able to compensate an over estimation of the medium capacity up to a certain limit by building up the queue but, if the medium capacity is under estimated, the medium will result under utilized. In the best case, XCC performance will be sub-optimal in the presence of an erroneous estimate of the capacity but, if the error becomes significant, XCC mechanisms will exhibit poor performance.

In Chapter 4 we introduced three alternative XCC router algorithms, Blind, ErrorS and MAC, that enable the operation of XCC mechanisms in time-varying capacity media. Blind achieves this by using queue speed to infer the available medium bandwidth, while ErrorS relies on queue accumulation over time to remove the error from the capacity estimation. The MAC algorithm, which is based on [17], estimates the medium capacity by assessing Medium Access Control (MAC) layer indicators such as the idle and busy periods. The algorithms proposed were shown to be stable for any BDP network or number of sources.

In Chapter 5 we presented results from extensive testing of the Blind, ErrorS and MAC algorithms, both through simulation and a FreeBSD implementation. The results show that the algorithms perform well in both single and multi-hop IEEE 802.11 scenarios, and outperform TCP Reno in terms of utilization for high BDP, throughput stability, flow fairness, and queuing delay.

In Chapter 6 we discussed the incremental deployment of XCC in large-scale networks, i.e. in the Internet. Since XCC is unlikely to be deployed in one step, we highlighted how XCC may co-exist with other forms of congestion control, and which mechanisms might allow XCC clouds to form an overlay over the Internet. The security implications of deploying XCC have been analyzed, concluding that XCC makes it easier for malicious end-systems to grab an unfair share of bandwidth and conduct other types of attacks; but it also makes it easier to implement network policing mechanisms.

7.2 Contributions

The emerging class of congestion control mechanisms based on explicit communication between routers and end-systems promises significant performance improvements over traditional congestion control mechanisms, in terms of network utilization, queuing latency, stability, and flow fairness. In this thesis we focus on the operation of such mechanisms over media with variable capacity, largely ignored up to now. The original contributions of this work are the following:

- **Enabling eXplicit Congestion Control in Variable Rate Media.**

We have developed a set of algorithms (Blind, ErrorS and MAC) that enable the operation of XCC mechanisms in media with time-varying capacity and, with that, networks using these media are now able to enjoy the benefits of XCC. This is particularly useful as we are facing a new generation of IEEE wireless technologies, such as 802.11n, which aim to increase the offered bitrate up to hundreds of Mbit/s. This represents an increase of the BDP of these networks by one order of magnitude and, as it is common knowledge, traditional congestion control mechanisms see their performance degrade as the BDP increases. The algorithms developed within the context of this thesis can be subdivided into two groups: those based solely on queue dynamics (Blind and ErrorS), and the MAC-based algorithm, which is an altered version of a previously proposed algorithm [17], and relies on information obtained from the MAC layer.

With the proposed algorithms we fill one important gap in current XCC proposals which, typically, do not consider time-varying capacity media. In order to enable the deployment of the XCC architecture every router in the Internet is required to have XCC capabilities; by explicitly addressing media with unknown or varying capacity, this thesis also represents one important step towards the widespread adoption of XCC.

- **Improved Understanding of the eXplicit Congestion Control**

Loop. In the description of our Problem Statement, Section 3, we have analyzed in detail the XCC control loop in the presence of a perturbation, which we considered to be a constant error in the router capacity estimate. It was shown that the control loop is able to compensate such error by building up the queue, as long as it does not exceed some threshold. This result contributes to a better understanding of the control loop, and it may be applied to the study of other sources of perturbation. This result, we believe, may assist in the design of alternative algorithms.

- **The Queue-Based Approach in XCC.** Although queue controllers based solely on queue dynamics have been proposed in the past, to the best of our knowledge this thesis addresses for the first time the use of such controllers in XCC. The early approach [17] to the problem of time-varying capacity relied on measurements at the MAC layer. The queue-based approach has some benefits in comparison with the approach followed in [17]. An algorithm based solely on queue dynamics abstracts from the underlying details of the physical and medium access layers, making no assumptions other than considering that those layers can provide fair medium access within a service class. For this reason, changes in the underlying technology, e.g. NIC firmware update following some Standard revision, are transparent to the queue controller, thus enhancing the system flexibility. Additionally, using QoS-enabled L2 technologies (e.g. IEEE 802.11e) becomes simpler using a queue-based approach.

7.3 Limitations

The limitations of our work are the following:

- **Dependence on the Buffer Size.** When using a queue-based algorithm the controller lacks a reference value for the capacity. For this reason the controller has to distribute an arbitrary amount of bandwidth until the queue starts to build and the available capacity can be

estimated. The amount of bandwidth that is distributed in each control interval will determine how fast the system will reach full utilization, thus its efficiency. It will also determine the maximum utilization overshoot when reaching full utilization. This capacity overshoot causes a queue spike so, in order to reduce the amount of packet loss experienced, the arbitrary amount of bandwidth distributed in each control interval must be a function of the queue buffer size. As a consequence, the system efficiency becomes tightly coupled to the queue buffer size, which - for achieving a fast response - should be at least comparable with the BDP of the network.

- **Model Simplifications.** The stability analysis of the ErrorS algorithm, presented in Appendix A, was done assuming a set of simplifications which limit its results. It did not incorporate the effect of the $\kappa(t)$ parameter, queuing delay, nor it considered queues to be bounded. A more thorough analysis which includes the mentioned system characteristics could lead to a more optimized choice of parameter values.
- **FreeBSD Implementation** Our FreeBSD implementation served as a proof-of-concept of the feasibility of the proposed algorithms in real systems; however it contained one major limitation - the bottleneck queue had to be artificially created, and the controller runs on top of this virtual queue. This means that we could not test the system over IEEE 802.11 media near its saturation point, where an higher level of bandwidth variance is expected to occur.

7.4 Topics for Future Work

Topics for future work are both new research ideas building upon the work presented in this thesis, and working issues to help improving the limitations of our work.

- **Exploration of New QoS Concepts.** It would be interesting to explore bandwidth allocation schemes other than max-min flow fair-

ness, the only that we have considered within the scope of this thesis. Namely, the recently revisited [24], [20] topic of utility fairness [98] and its variations seem particularly promising. Bridging the gap between what the network can offer and what the user application requires is a long-standing problem of IP networks. An interesting idea from this point of view is to explore the queue that exists between the application and the transport layer, that is the socket buffer. Another interesting topic for further research is the interaction of these mechanisms with QoS-enabled Layer 2 technologies such as IEEE 802.11e, which interfere with bandwidth allocation.

- **Multipath.** Multipath was left out of this thesis, and we assumed that a flow would traverse a single path between source and destination. How congestion control can be performed across multiple paths simultaneously, and particularly when using XCC protocols, still is an open issue. This gains even more relevance as new communication paradigms rise, e.g. network coding, which heavily rely on distributed routing techniques.
- **General Applicability to eXplicit Congestion Control Protocols.** In this thesis we have analyzed in-depth the applicability of the proposed algorithms to the particular case of two XCC protocols: XCP and RCP. It was described how the main concepts behind the algorithms could be applied to other protocols, giving the example of CL-TCP; however no extensive evaluation was done. How the algorithms developed in this thesis can be applied to any XCC protocol is regarded as a topic for further work.
- **Complete Model Analysis.** The simplifications considered in our stability analysis of the ErrorS algorithm have already been addressed. Aspects such as the inclusion of the κ parameter, explicitly model the queuing delay, or the consideration of bounded queues would contribute to a better understanding and dimensioning of the algorithm parameters.

- **Improving the FreeBSD Implementation.** As mentioned above, the FreeBSD implementation relied on the creation of a virtual queue which limited its ability to actually run the algorithm over a saturated IEEE 802.11 medium. Future work on the implementation could focus on linking the queue controller with the actual NIC queue, so that more realistic tests can be performed.

Appendix A

Stability Analysis of the ErrorS Algorithm

We use a fluid model in order to study the stability of the ErrorS algorithm as a function of the values of α, β and μ . Abstracting from packet granularity, neglecting queuing delay and its effect, and assuming a continuous control loop, as described in Section 2.3, an ErrorS system is characterized by the following equations:

$$\dot{y}(t) = \frac{F(t-d)}{d} \quad (\text{A.1})$$

$$F(t) = \alpha \cdot (C - y(t)) - \frac{\mu}{d} \cdot \int_0^t \frac{\beta}{\alpha \cdot d} \cdot q(t) dt - \frac{\beta}{d} \cdot q(t) \quad (\text{A.2})$$

$$\dot{q}(t) = y(t) - C_{real} \quad (\text{A.3})$$

$$C = C_{real} + \epsilon \quad (\text{A.4})$$

where $y(t)$ represents the aggregate incoming bandwidth, C is the capacity value configured at the router, C_{real} is the actual bottleneck link capacity, ϵ is the router capacity estimate error, $q(t)$ represents queue length, and d is the system (average) RTT. α, β and μ are the tunable constants of the algorithm. Now consider the following substitutions:

$$K_1 = \frac{\alpha}{d} \quad (\text{A.5})$$

$$K_2 = \frac{\beta}{d^2} \quad (\text{A.6})$$

$$K_3 = \frac{\mu \cdot \beta}{\alpha \cdot d^3} \quad (\text{A.7})$$

which allows us to rewrite Eq. A.1 and Eq. A.2 as:

$$\dot{y}(t) = K_1 \cdot (C - y(t - d)) - K_3 \cdot \int_0^t q(t - d) dt - K_2 \cdot q(t - d) \quad (\text{A.8})$$

Applying the Laplace transformation we finally get:

$$s \cdot Y(s) = e^{-sd} \left(K_1 \cdot [C - Y(s)] - K_3 \cdot \frac{Q(s)}{s} - K_2 \cdot Q(s) \right) \quad (\text{A.9})$$

which is equivalent to the negative feedback control loop represented in Fig. A.1. The open-loop transfer function $G(s)$ of the the system is given by:

$$G(s) = \frac{Q(s)}{Y(s)} = e^{-sd} \cdot \frac{K_1 \cdot s^2 + K_2 \cdot s + K_3}{s^3} \quad (\text{A.10})$$

To ensure system stability, we have to guarantee that the phase (or gain) margin of the open-loop transfer function is positive. To simplify the analysis, let us consider the following condition:

$$\mu = \beta \quad (\text{A.11})$$

which implies:

$$K_3 = \frac{K_2^2}{K_1} \quad (\text{A.12})$$

Using this condition, the zeros w_z of $G(s)$ are:

$$w_z = \frac{K_2}{2 \cdot K_1} \cdot (1 \pm i\sqrt{3}) \quad (\text{A.13})$$

Now we have to choose the cut-off frequency w_{ErrorS} of the ErrorS system. For the sake of tractability, we consider the cut-off frequency as a multiple of the real part of the frequency of the zeros of the open-loop response:

Figure A.1: The ErrorS control loop.

$$w_{ErrorS} = n \cdot \Re\{w_z\} = n \cdot \frac{K_2}{2 \cdot K_1} \quad (\text{A.14})$$

Applying the inherent condition of the cut-off frequency (i.e. unitary gain):

$$|G(jw_{ErrorS})| = 1 \quad (\text{A.15})$$

we get:

$$\beta = \frac{\sqrt{1 - \left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^4}}{\left(\frac{n}{2}\right)^3} \cdot \alpha^2 \quad (\text{A.16})$$

To ensure system stability we must guarantee that the open-loop response phase does not exceed $-\pi$ at the cut-off frequency:

$$\angle G(jw_{ErrorS}) > -\pi \quad (\text{A.17})$$

which results in:

$$\frac{\beta}{\alpha} \leq \frac{2}{n} \cdot \left[\arctan \left(\frac{n}{2 - \left(\frac{n}{\sqrt{2}}\right)^2} \right) - \frac{\pi}{2} \right] \quad (\text{A.18})$$

The final step of the design analysis consists in choosing the actual cut-off frequency of the open-loop response by choosing the value of n . We choose n so that the suppression of the error is as fast as possible, and the importance given to the erroneous estimation is as low as possible. Considering that the speed of the error suppression is controlled by the parameter μ ($\mu = \beta$), and that α is the weight given to the erroneous capacity estimation, we achieve our design objective by choosing the value of n that maximizes the ratio $\frac{\beta}{\alpha}$. In Fig. A.2 we plot this ratio as a function of n , where it is observable that a maximum is obtained for $n \approx 3.5$. Note that using $n = 3.5$ implies a system cut-off frequency w_{ErrorS} :

$$w_{ErrorS} = \frac{3.5}{2} \cdot \frac{K_2}{K_1} = 1.75 \cdot \frac{\beta}{\alpha \cdot d} \quad (\text{A.19})$$

Applying $n = 3.5$ in Eq. A.16, Eq. A.18 we get our final set of stability restrictions for the parameters α, β and μ :

$$\beta = 0.5047 \cdot \alpha^2; \quad \frac{\beta}{\alpha} \leq 0.4955; \quad \mu = \beta \quad (\text{A.20})$$

which, simplifying, come as:

$$\beta = 0.5047 \cdot \alpha^2; \quad \alpha \leq 0.9818; \quad \mu = \beta \quad (\text{A.21})$$

In order to reduce the oscillation amplitude and to increase the system robustness to fluctuations we should add a certain phase margin. Based on simulation results we recommend the utilization of the following values:

$$\beta = 0.1817; \quad \alpha = 0.6; \quad \mu = 0.1817 \quad (\text{A.22})$$

Figure A.2: The maximum value of β as a function of n .

for which the ErrorS algorithm is stable independently of link capacity, delay or number of sources.

Appendix B

Recommended Parameter Values for the Blind and the ErrorS Algorithms

	Blind	ErrorS
α	0.4	0.6
β	0.226	0.1817
μ	-	0.1817
ρ	0.22	0.15
Q_χ	$0.541 \cdot Q_{max}$	$0.444 \cdot Q_{max}$
τ	0.225	0.37

Table B.1: Recommended parameter values for the Blind and the ErrorS algorithms

Appendix C

Pseudo-code of the Blind Algorithm

Algorithm 1 Blind Algorithm Pseudo-code

inititalization:

$last_queue \leftarrow 0$

$\kappa \leftarrow 0$

$smooth_queue \leftarrow 0$

on queue timeout:

$smooth_queue \leftarrow \rho \cdot queue + (1 - \rho) \cdot smooth_queue$

if $smooth_queue \leq \tau \cdot \kappa$ **then**

$Q \leftarrow Q_\chi$

else

$Q \leftarrow queue$

end if

$\kappa \leftarrow \rho \cdot |Q - smooth_queue| + (1 - \rho) \cdot \kappa$

$spare_bw \leftarrow (last_queue - queue)/control_intvl$

$Feedback = \alpha \cdot spare_bw - \beta \cdot (queue - \kappa)/control_intvl$

$last_queue \leftarrow queue$

Appendix D

Pseudo-code of the ErrorS Algorithm

Algorithm 2 ErrorS Algorithm Pseudo-code

inititalization:

$last_error \leftarrow 0$

$\kappa \leftarrow 0$

$smooth_queue \leftarrow 0$

on queue timeout:

$input_bw = input_traffic_bytes / control_intvl$

$smooth_queue \leftarrow \rho \cdot queue + (1 - \rho) \cdot smooth_queue$

if $smooth_queue \leq \tau \cdot \kappa$ **then**

$Q \leftarrow Q_x$

else

$Q \leftarrow queue$

end if

$\kappa \leftarrow \rho \cdot |Q - smooth_queue| + (1 - \rho) \cdot \kappa$

$error \leftarrow \frac{\beta}{\alpha} \cdot \frac{(queue - \kappa)}{control_intvl}$

$acc_error \leftarrow \min \left(\frac{\mu}{\alpha} \cdot \left(C - input_bw - \frac{Q_x}{control_intvl} \right), acc_error + (error + error_last) / 2 \right)$

$Feedback \leftarrow \alpha \cdot (C - input_bw) - \mu \cdot acc_error - \beta \cdot (queue - \kappa) / control_intvl$

$last_error \leftarrow error$

$input_traffic_bytes \leftarrow 0$

Bibliography

- [1] J. Postel. Transmission Control Protocol. RFC 793, September 1981.
- [2] Van Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM Computer Communication Review*, 1988.
- [3] J. Nagle. Congestion control in IP/TCP internetworks. RFC 896, January 1984.
- [4] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proc. of ACM SIGCOMM*, pages 89–102, New York, NY, USA, 2002. ACM.
- [5] Nandita Dukkkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. Processor Sharing Flows in the Internet. In *Proc. of IEEE IWQoS*, 2005.
- [6] B. Wydrowski and M. Zukerman. MaxNet: a new network congestion control architecture for max-min fairness. In *Proc. of IEEE ICC*, 2003.
- [7] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, January 1997.
- [8] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.
- [9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996.

- [10] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. In *Proc. of ACM SIGCOMM*, pages 24–35, New York, NY, USA, 1994. ACM.
- [11] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proc. of ACM Mobicom*, pages 287–297, New York, NY, USA, 2001. ACM.
- [12] Xiaomeng Huang, Chuang Lin, Fengyuan Ren, Guangwen Yang, P.D. Ungsunan, and Yuanzhuo Wang. Improving the Convergence and Stability of Congestion Control Algorithm. In *Proc. of IEEE ICNP*, 2007.
- [13] Y. Zhang and T.R. Henderson. An implementation and experimental study of the explicit control protocol (XCP). In *Proc. of IEEE INFOCOM*, volume 2, pages 1037–1048 vol. 2, 13-17 March 2005.
- [14] A. Kapoor, A. Falk, T. Faber, and Y. Pryadkin. Achieving faster access to satellite link bandwidth. In *Proc. of INFOCOM 2005*, volume 4, pages 2870–2875 vol. 4, 13-17 March 2005.
- [15] G. Nychis, G. Sardesa, and S. Seshan. Analysis of XCP in a Wireless Environment. Carnegie Mellon University, 2007.
- [16] ANSI/IEEE. 802.11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 2000.
- [17] Yang Su and Thomas R. Gross. WXCP: Explicit Congestion Control for Wireless Multi-hop Networks. In *Proc. of IEEE IWQoS*, pages 313–326, 2005.
- [18] Saad Biaz and Nitin H. Vaidya. Is the round-trip time correlated with the number of packets in flight? In *Proc. of ACM IMC*, pages 273–278, New York, NY, USA, 2003. ACM.
- [19] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 2481, September 2001.

- [20] Konstantin Miller and Tobias Harks. Utility Max-Min Fair Congestion Control With Time-Varying Delays. In *Proc. of IEEE INFOCOM*, 2008.
- [21] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166, March 2008.
- [22] Hideyuki Shimonishi, Medy Sanadidi, and Tutomu Murase. Assessing Interactions among Legacy and High-Speed TCP Protocols. In *Proc. of PFLDNet*, 2007.
- [23] S. Floyd and L. Andrew. Common TCP Evaluation Suite. Internet Draft (draft-irtf-tmrg-tests-00) (work in progress), July 2008.
- [24] Bob Briscoe. Flow rate fairness: dismantling a religion. *SIGCOMM Comput. Commun. Rev.*, 37(2):63–74, 2007.
- [25] Sally Floyd and Mark Allman. Comments on the Usefulness of Simple Best-Effort Traffic. RFC 5290, July 2008.
- [26] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling*. John Wiley and Sons, Inc., 1991.
- [27] G.A. Korn and T.M. Korn. *Mathematical Handbook for Scientists and Engineers*. McGraw-Hill, 1967.
- [28] H. Balakrishnan, N. Dukkupati, N. McKeown, and C.J. Tomlin. Stability Analysis of Explicit Congestion Control Protocols. *Communications Letters, IEEE*, 11:823–825, 2007.
- [29] Injong Rhee and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variants. In *Proc. of PFLDnet*, 2005.
- [30] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *Proc. of IEEE INFOCOM*, 2006.

- [31] Van Jacobson. Modified TCP Congestion Avoidance Algorithm. end2end-interest mailing list, April 30 1990.
- [32] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-based congestion control for unicast applications. In *Proc. of the ACM SIGCOMM*, pages 43–56, New York, NY, USA, 2000. ACM.
- [33] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: A simple model and its empirical validation. *Proc. of ACM SIGCOMM*, pages 303–314, 1998.
- [34] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448, January 2003.
- [35] Filipe Abrantes and Manuel Ricardo. On Congestion Control for Interactive Applications in Dynamic Heterogeneous 4G Networks. In *Proc. of PIMRC*, 2005.
- [36] Andrei Gurtov and Jouni Korhonen. Effect of vertical handovers on performance of TCP-friendly rate control. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(3):73–87, 2004.
- [37] Injong Rhee and Lisong Xu. Limitations of equation-based congestion control. In *Proc. of ACM SIGCOMM*, pages 49–60, New York, NY, USA, 2005. ACM.
- [38] Kai Chen and Klara Nahrstedt. Limitations of Equation-Based Congestion Control in Mobile Ad Hoc Networks. In *Proc. of IEEE ICDCS*, pages 756–761, Washington, DC, USA, 2004. IEEE Computer Society.
- [39] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.*, 14(6):1246–1259, 2007.
- [40] J. Andren, M. Hilding, and D. Veitch. Understanding end-to-end Internet traffic dynamics. In *Proc. of IEEE GLOBECOM*, 1998.

- [41] Jim Martin, Arne Nilsson, and Injong Rhee. Delay-based congestion avoidance for TCP. *IEEE/ACM Trans. Netw.*, 11(3):356–369, 2003.
- [42] Hadrien Bulot, R. Les Cottrell, and Richard Hughes-Jones. Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks. *Grid Computing, Springer Journal of*, 1:345–359, 2004.
- [43] S. Mascolo and F. Vacirca. The Effect of Reverse Traffic on the Performance of New TCP Congestion Control Algorithms. In *Proc. of PLFD-net*, 2006.
- [44] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control for Fast, Long Distance Networks. In *Proc. of IEEE INFOCOM*, 2004.
- [45] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, 1993.
- [46] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Trans. Netw.*, 2(2):122–136, 1994.
- [47] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proc. of IEEE IWQoS*, 1999.
- [48] S.S. Kunniyur and R. Srikant. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE/ACM Trans. Netw.*, 12(2):286–299, 2004.
- [49] Srisankar S. Kunniyur and R. Srikant. An Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. In *Proc. of ACM SIGCOMM*, 2001.
- [50] Raj Jain, Shiv Kalyanaraman, and Ram Viswanathan. The OSU scheme for congestion avoidance in ATM networks: lessons learnt and extensions. *Perform. Eval.*, 31(1-2):67–88, 1997.

- [51] Shivkumar Kalyanaraman, Raj Jain, Sonia Fahmy, Rohit Goyal, and Bobby Vandalore. The ERICA switch algorithm for ABR traffic management in ATM networks. *IEEE/ACM Trans. Netw.*, 8(1):87–98, 2000.
- [52] A. Charny. An Algorithm for Rate Allocation in a Packet-Switching Network With Feedback. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1994.
- [53] Yehuda Afek, Yishay Mansour, and Zvi Ostfeld. Phantom: a simple and effective flow control scheme. *SIGCOMM Comput. Commun. Rev.*, 26(4):169–182, 1996.
- [54] S. Keshav. *An engineering approach to computer networking: ATM networks, the Internet, and the telephone network*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
- [55] Ion Stoica, Scott Shenker, and Hui Zhang. Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks. In *Proc. of ACM SIGCOMM*, 1998.
- [56] Dina Katabi. Decoupling Congestion Control and Bandwidth Allocation Policy With Application to High Bandwidth-Delay Product Networks. Ph.D. Dissertation, Massachusetts Institute of Technology, March 2003.
- [57] J. Postel. Internet Protocol. RFC 791, September 1981.
- [58] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.
- [59] S. Low, L. Andrew, and B. Wydrowski. Understanding XCP: equilibrium and fairness. In *Proc. of IEEE INFOCOM*, 2005.
- [60] P. Wang and D.L. Mills. Simple Analysis of XCP Equilibrium Performance. In *Proc. of IEEE CISS*, 2006.
- [61] L. L. H. Andrew, B. P. Wydrowski, and S. H. Low. An Example of Instability in XCP.

- [62] Nandita Dukkkipati and Nick McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, 2006.
- [63] Nandita Dukkkipati, Yashar Ganjali, and Rui Zhang-Shen. Typical versus Worst Case Design in Networking. In *ACM HotNets-IV*, 2005.
- [64] Filipe Abrantes, Joao Taveira Araujo, and Manuel Ricardo. Flash Crowd Effect in RCP. In *Proc. of PFLDnet*, 2008.
- [65] Lachlan L. H. Andrew, Krister Jacobsson, Steven H. Low, Martin Suchara, Ryan Witt, and Bartek P. Wydrowski. Maxnet: Theory and implementation. CalTech Technical Report Available: [http://netlab.caltech.edu/maxnet/MaxNet Implementation TechReport.pdf](http://netlab.caltech.edu/maxnet/MaxNet%20Implementation%20TechReport.pdf), 2006.
- [66] FastSoft. <http://www.fastsoft.com>.
- [67] S. E. Kingsland. *Modeling nature*. 1995.
- [68] Yong Xia, Lakshminarayanan Subramanian, Ion Stoica, and Shivkumar Kalyanaraman. One more bit is enough. In *SIGCOMM Comput. Commun. Rev.*, volume 35, pages 37–48, New York, NY, USA, 2005. ACM.
- [69] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-Start for TCP and IP. RFC 4782, January 2007.
- [70] Pasi Sarolahti, Mark Allman, and Sally Floyd. Determining an appropriate sending rate over an underutilized network path. *Comput. Netw.*, 51(7):1815–1832, 2007.
- [71] Yee-Ting Li, Douglas Leith, and Robert N. Shorten. Experimental evaluation of TCP protocols for high-speed networks. *IEEE/ACM Trans. Netw.*, 15(5):1109–1122, 2007.
- [72] Y. Zhang and M. Ahmed. A control theoretic analysis of XCP. In *Proc. of IEEE INFOCOM*, volume 4, pages 2831–2835 vol. 4, 13-17 March 2005.

- [73] The network simulator - ns-2. <http://www.isi.edu/nsnam/ns/>.
- [74] Mathwork's MATLAB/Simulink. <http://www.mathworks.com/>.
- [75] A.V. Oppenheim and R.W. Schaffer. *Discrete-Time Signal Processing*. Prentice-Hall, 1999.
- [76] Filipe Abrantes. MATLAB Scripts for "Explicit Congestion Control Algorithms for VariableCapacity Media". <http://pong.inescporto.pt/~fabrantes/root/code>.
- [77] Filipe Abrantes and Manuel Ricardo. XCP for shared-access multi-rate media. *SIGCOMM Comput. Commun. Rev.*, 36(3):27–38, 2006.
- [78] Filipe Abrantes. Simulator code and scripts for "Explicit Congestion Control Algorithms for Variable Capacity Media". <http://pong.inescporto.pt/~fabrantes/root/code>.
- [79] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, 1994.
- [80] Jinyang Li, Charles Blake, Douglas S.J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of Ad Hoc wireless networks. In *Proc. of ACM MobiCom*, pages 61–69, New York, NY, USA, 2001. ACM.
- [81] P. Gupta and P.R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, Mar 2000.
- [82] Benyuan Liu, Patrick Thiran, and Don Towsley. Capacity of a wireless ad hoc network with infrastructure. In *Proc. of ACM MobiHoc*, pages 239–246, New York, NY, USA, 2007. ACM.
- [83] D. Allen. Hidden terminal problems in Wireless LAN's. IEEE 802.11 Working Group Papers, 1993.
- [84] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? *Communications Magazine, IEEE*, 39(6):130–137, Jun 2001.

- [85] Information Sciences Institute (ISI)'s XCP Implementation.
<http://www.isi.edu/isi-xcp/#software>.
- [86] FreeBSD IP Firewall. <http://www.freebsd.org/doc/en/books/handbook/firewalls-ipfw.html>.
- [87] Jouni Malinen. Host AP driver for Intersil Prism 2/2.5/3.
<http://hostap.epitest.fi/>.
- [88] Pyun YongHyeon and Max Laier. OpenBSD Packet Filter for FreeBSD.
<http://pf4freebsd.love2party.net/>.
- [89] Vinay Ribeiro, Rudolf Riedi, Richard Baraniuk, Jiri Navratil, and Les Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Proc. of PAM*, 2003.
- [90] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Proc. of PAM*, 2002.
- [91] NLANR/DAST's Iperf. <http://sourceforge.net/projects/iperf>.
- [92] D. M. Lopez-Pacheco, C. Pham, and L. Lefevre. XCP-i : eXplicit Control Protocol for Heterogeneous Inter-Networking of High-Speed Networks. In *Proc. of IEEE GLOBECOM*, pages 1–6, 2006.
- [93] Aaron Falk, Dina Katabi, and Yuri Pryadkin. Specification for the Explicit Control Protocol (XCP). draft-falk-xcp-03.txt (work in progress), July 2007.
- [94] Wesley M. Eddy, Shawn Ostermann, and Mark Allman. New techniques for making transport protocols robust to corruption-based loss. *SIGCOMM Comput. Commun. Rev.*, 34(5):75–88, 2004.
- [95] Dina Katabi. XCP's Performance in the Presence of Malicious Flows. In *Proc. of PFLDnet*, 2004.
- [96] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *SIGCOMM Comput. Commun. Rev.*, 29(5):71–78, 1999.

- [97] Bob Briscoe, Arnaud Jacquet, Carla Di Cairano-Gilfedder, Alessandro Salvatori, Andrea Soppera, and Martin Koyabe. Policing congestion response in an internetwork using re-feedback. *SIGCOMM Comput. Commun. Rev.*, 35(4):277–288, 2005.
- [98] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.