

Propositions for a robust and inter-operable eXplicit Control Protocol on heterogeneous high speed networks

Dino Martín Lopez Pacheco

▶ **To cite this version:**

Dino Martín Lopez Pacheco. Propositions for a robust and inter-operable eXplicit Control Protocol on heterogeneous high speed networks. Networking and Internet Architecture [cs.NI]. Ecole normale supérieure de lyon - ENS LYON, 2008. English. NNT : . tel-00295072

HAL Id: tel-00295072

https://theses.hal.science/tel-00295072

Submitted on 11 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre: 459

N° attribué par la bibliothèque : 07ENSL0 459

THESE

en vue d'obtenir le grade de

Docteur de l'Université de Lyon - Ecole Normale Supérieure de Lyon

spécialité : Informatique

Laboratoire de l'Informatique du Parallélisme

Ecole doctorale de Mathématique et Informatique fondamentale

présentée et soutenu publiquement le 09/06/08

par Monsieur Dino-Martín LOPEZ-PACHECO

Titre :

Propositions for a robust and inter-operable eXplicit Control Protocol on heterogeneous high speed networks

---

Directeurs de thèse : *Monsieur LEFEVRE Laurent*
Monsieur PHAM Congduc

Après avis de : *Monsieur Jean-Jacques PANSIOT, Membre/Rapporteur*
Monsieur Nageswara RAO, Membre/Rapporteur

Devant la commission d'examen formée de :

Monsieur Chadi BARAKAT, Membre
Monsieur Frédéric DESPREZ, Membre
Monsieur Laurent LEFEVRE, Membre
Monsieur Jean-Jacques PANSIOT, Membre/Rapporteur
Monsieur Congduc PHAM, Membre
Monsieur Nageswara RAO, Membre/Rapporteur

Acknowledgments

First of all, I want to thank all my family, my girlfriend and Anya.

I would also like to thank Chadi BARAKAT, Frédéric DESPREZ, Laurent LEFEVRE, Jean-Jacques PANSIOT, Congduc PHAM and Nageswara RAO to be part of my jury. I especially want to thank Jean-Jacques PANSIOT and Nageswara Rao for the useful comments they provided to improve my thesis.

I want to say thank you also to my advisors Laurent LEFEVRE and Congduc PHAM. Their advises and support have been essential to the developpement of this thesis.

Thanks to Olivier MORNARD, JP, Romaric, Seb, Ludo, Dinil and the others PhD students from my team.

Thanks to the latin community.

Thanks to all my friends and anybody who always believed in a good ending of this thesis.

I am grateful to CONACyT for making this thesis possible by sponsoring the research presented here.

Thanks God.

Dino M.

Résumé

Les protocoles de contrôle de congestion ont pour but d'autoriser un partage équitable des ressources entre les utilisateurs et d'éviter la congestion dans les réseaux de communications. Dans cette thèse, nous avons commencé par comparer la performance des différents protocoles dans des modèles de réseaux qui intègrent la notion de variation de bande passante produit par l'agrégation/désagrégation des flux ainsi que la présence des mécanismes fournissant de la Qualité du Service.

Ainsi, nous avons montré que ce sont les protocoles basés sur l'assistance de routeurs fournissant aux émetteurs un taux d'émission explicite ("Explicit Rate Notification" (*ERN*)) partagent mieux les ressources du réseaux entre les utilisateurs et évitent mieux la congestion que les protocoles de bout-en-bout (comme les protocoles basés sur TCP). Cependant, l'absence d'inter-opérabilité des protocoles *ERN* avec les routeurs *non-ERN* (par exemple, les routeurs *DropTail*) et les protocoles de congestion de bout-en-bout (comme *TCP*), empêche leur mise en place dans les réseaux actuels.

Pour résoudre les problèmes d'inter-opérabilité des protocoles *ERN* avec des routeurs *non-ERN*, nous avons proposé des stratégies et des mécanismes capables de détecter les groupes des routeurs *non-ERN* entre deux routeurs *ERN*, d'estimer la bande passante minimale disponible à l'intérieur de ces groupes, et de créer des routeurs virtuels qui remplacent chaque groupe de routeurs *non-ERN* par un routeur *ERN*.

Nous avons également proposé un système d'équité inter-protocolaire entre les protocoles *ERN* et les protocoles de bout-en-bout. Avec notre solution, les routeurs *ERN* dans une première étape estiment les besoins en terme de bande passante des flux *ERN* et *non-ERN* puis, dans une deuxième étape, limitent le débit des flux *non-ERN* en rejetant leurs paquets avec une probabilité qui dépend des résultats de la première étape.

Le succès des protocoles *ERN* est basé sur l'information de rétroalimentation, calculée par les routeurs, et renvoyée par les récepteurs aux émetteurs. Nous avons montré que les protocoles *ERN* deviennent instables face aux pertes de l'information de rétroalimentation dans des environnements hétérogènes et à bande passante variable. Dans ce cadre là, nous avons proposé une nouvelle architecture qui améliore la robustesse des protocoles *ERN*, ainsi que la réactivité des émetteurs.

Toutes nos propositions, applicables sur d'autres protocoles *ERN*, ont été expérimentées et validées sur le protocole *eXplicit Control Protocol* (XCP). Ainsi, nous avons montré que nos solutions surmontent les défis concernant les problèmes d'inter-opérabilité des protocoles *ERN* dans un grand nombre des scénarios et de topologies.

De cette façon, nous avons développé les bases pour bénéficier d'un protocole *ERN* capable d'être déployé dans des réseaux hétérogènes à grand produit *bande passante - délai* où le transfert de grande quantité des données est nécessaire, tels que les réseaux de grilles (ex. GÉANT).

Abstract

The congestion control protocols aim to fairly share the network resources between users and avoid congestion. In this thesis, we have first compared the performance of different protocols in networks where the bandwidth varies over time (Variable Bandwidth Environments – VBE –) due to the aggregation/disaggregation of flows and/or the presence of Quality of Service (QoS) mechanisms.

Thus, we have shown that the routers-assisted protocols providing Explicit Rate Notification (ERN protocols) provide higher fairness level and avoid congestion better than End-to-End (E2E) protocols (e.g., TCP-based protocols). However, the absence of inter-operability of ERN protocols with the non-ERN routers (like the DropTail routers) and the E2E congestion control protocols (like TCP) prevent the deployed of ERN protocols in current networks.

In order to solve the problems of inter-operability of ERN protocols with non-ERN routers, we have proposed a solution which detects the set of non-ERN router between two ERN routers, estimates the available bandwidth in that set of non-ERN routers, and creates a virtual router that computes a feedback reflecting the estimated available bandwidth.

Regarding the inter-operability problems of ERN protocols with E2E protocols, we have proposed a solution which provides friendliness between both kind of protocols. The mechanisms implemented by our friendliness solution can be divided in two steps. First, our solution estimates the resources needed by the ERN and E2E protocols in terms of bandwidth. Later, our friendliness solution randomly drops packets belonging to the E2E flows, with a probability updated on base of the results found in the first step.

Since the success of ERN protocols depends on the information about the state of the network provided by the routers (the feedback), in this thesis we have studied the behavior of ERN protocols in scenarios where a percentage of the feedback has been lost. Such studies show that feedback losses in networks with VBE may lead to a chaotic behavior of ERN flows. Therefore, we have proposed a new architecture for ERN protocols which improves the responsiveness and the robustness of ERN flows in networks with VBE and feedback losses.

The set of mechanisms and solutions presented in this thesis, easy to apply to several ERN protocols, have been implemented and validated on the eXplicit Control Protocol (XCP). Thus, we have shown that our solutions surmount the inter-operability problems of ERN protocols in a wide range of network topologies and scenarios.

This thesis has provided the basis for creating an ERN protocol able to be gradually deployed in current heterogeneous large *bandwidth-delay product* networks, where users frequently need to move very large amount of data (e.g., Data Grid).

Contents

1 Introduction **1**
- 1.1 Motivation and Problem statement . . . 1
- 1.2 General Aim and Context of applications of our propositions . . . 3
- 1.3 Contributions . . . 4
- 1.4 Thesis Outline . . . 5

2 State of the Art **7**
- 2.1 Essential Components in Network Communication . . . 8
  - 2.1.1 End hosts . . . 8
  - 2.1.2 Forwarding nodes: Layer 2 and Layer 3 devices . . . 8
- 2.2 Transport and Congestion Control Protocols . . . 8
  - 2.2.1 Importance of Transport Protocols . . . 8
  - 2.2.2 Importance of Congestion Control Protocols . . . 11
- 2.3 Standard Congestion Control Protocols . . . 13
  - 2.3.1 From TCP Tahoe to TCP New Reno . . . 13
    - 2.3.1.1 TCP Tahoe . . . 13
    - 2.3.1.2 TCP Reno . . . 15
    - 2.3.1.3 TCP New Reno . . . 16
    - 2.3.1.4 Limitations of TCP Tahoe, Reno and New Reno . . . 17
  - 2.3.2 TCP Vegas: towards a more intelligent TCP version . . . 18
  - 2.3.3 Limitations of Standard Protocols in LDHP Networks . . . 20
- 2.4 Advanced Congestion Control Protocols for LDHP Networks . . . 20
  - 2.4.1 New challenges of Protocols for LDHP Networks . . . 21
  - 2.4.2 UDP-based protocols . . . 21
    - 2.4.2.1 UDT . . . 21
    - 2.4.2.2 RUNAT . . . 22
    - 2.4.2.3 Limitations of UDP-based protocols . . . 24
  - 2.4.3 Protocols only increasing the aggressiveness of TCP . . . 24
    - 2.4.3.1 High Speed TCP . . . 24
    - 2.4.3.2 Scalable TCP . . . 25
    - 2.4.3.3 Binary Increase TCP . . . 26
    - 2.4.3.4 Limitations of Protocols only increasing the aggressiveness of TCP . . . 29
  - 2.4.4 Protocols with DBECD mechanisms . . . 29
    - 2.4.4.1 FAST TCP . . . 29
    - 2.4.4.2 Compound TCP . . . 30

2.4.4.3 Others protocols . . . 31
2.4.4.4 Limitations of Protocols with DBECD mechanisms . . . 31
2.4.5 Protocols with Available Bandwidth Estimation strategies (Westwood TCP) . . . 31
2.5 Buffer management and notification for mitigating congestion . . . 32
2.5.1 Buffer management solutions without end hosts interactions . . . 33
2.5.1.1 Random Early Detection . . . 33
2.5.1.2 CHOKe . . . 33
2.5.1.3 BLUE . . . 34
2.5.1.4 Stabilized RED . . . 35
2.5.1.5 Limitations of AQM mechanisms . . . 36
2.5.2 Solutions providing ECN to end hosts . . . 36
2.5.2.1 Explicit Congestion Notification . . . 36
2.5.2.2 Variable-structure congestion Control Protocol . . . 37
2.5.2.3 Limitations of ECN solutions . . . 37
2.5.3 Solutions providing ERN to end hosts . . . 38
2.5.3.1 eXplicit Control Protocol . . . 38
2.5.3.2 JetMax . . . 42
2.5.3.3 TCP Quickstart . . . 44
2.5.3.4 Limitations of ERN Protocols in Heterogeneous Networks . . . 45
2.6 Conclusion . . . 45

3 E2E and ERN Protocols on Variable Bandwidth Environments 47
3.1 Sinusoidal-based bandwidth variation environments . . . 48
3.1.1 Definition . . . 48
3.1.2 Network topology for simulating sinusoidal-based models . . . 49
3.1.3 TCP on sinusoidal-based bandwidth variation environments . . . 49
3.1.3.1 Modeling TCP . . . 49
3.1.3.2 Simulation results for TCP . . . 52
3.1.4 HSTCP on sinusoidal-based bandwidth variation environments . . . 54
3.1.5 XCP on sinusoidal-based bandwidth variation environments . . . 55
3.2 Step-based bandwidth variation environments . . . 56
3.2.1 Network topology for simulating step-based models . . . 59
3.2.2 HSTCP on step-based bandwidth variation environments . . . 59
3.2.3 XCP on step-based bandwidth variation environments . . . 61
3.3 Conclusion and lessons learned . . . 63

4 Enabling inter-operability between ERN protocols and non-ERN routers 65
4.1 Defining the non-XCP router and non-XCP cloud terms . . . 66
4.2 Sensitivity of XCP to non-XCP routers . . . 66
4.3 Enhancing XCP for heterogeneous internetworking . . . 67
4.4 XCP-*i*: architecture and algorithm in routers . . . 69
4.4.1 Detecting non-XCP clouds . . . 69
4.4.2 Detecting the XCP-*i* edge routers . . . 71
4.4.3 Estimating the available resources into the non-XCP cloud . . . 72
4.4.4 The XCP-*i* virtual router . . . 74
4.5 XCP-*i* : architecture in end-hosts . . . 76

4.6 Simulation results . . . 76
4.6.1 Performance and Fairness in virtual XCP-i virtual routers . . . 77
4.6.1.1 Incremental deployment around non-XCP clouds . . . 77
4.6.1.2 Merge scenario: n non-XCP clouds share one XCP path . . . 77
4.6.1.3 Fork scenario: one non-XCP cloud serves n XCP paths . . . 81
4.6.2 Impact of the bandwidth estimation accuracy on XCP-i . . . 81
4.7 Open issues in our XCP-i approach . . . 84
4.8 Conclusion . . . 85

5 Improving robustness of ERN protocols against feedback losses 87
5.1 Impact of ACK losses on XCP . . . 88
5.1.1 Impact of ACK losses in fully controlled friendly networks . . . 88
5.1.2 Impact of ACK losses on VBE . . . 89
5.2 Simulating XCP on VBE and ACK losses . . . 90
5.2.1 Topology and variation model . . . 90
5.2.2 No ACK losses . . . 91
5.2.3 ACK losses . . . 93
5.3 XCP-r: Building a robust XCP version . . . 94
5.4 Simulating XCP-r on VBE and ACK losses . . . 96
5.4.1 Scenario of simulation . . . 96
5.4.2 Simulation results . . . 96
5.5 Other simulations of XCP & XCP-r under ACK losses . . . 98
5.6 Conclusion . . . 100

6 Enabling inter-operability between TCP and ERN protocols 103
6.1 Why XCP flows are unable to compete against TCP flows . . . 103
6.2 XCP vs TCP . . . 104
6.3 Propositions for an XCP-TCP friendliness mechanism . . . 107
6.3.1 Defining the XCP-TCP friendliness . . . 107
6.3.2 Estimating the resources needed by XCP . . . 108
6.3.3 Limiting the TCP throughput . . . 109
6.3.4 Special remark of our XCP-TCP friendliness mechanism . . . 110
6.4 Testing our XCP-TCP friendliness solution . . . 111
6.4.1 1 XCP flow & 2 TCP competing flows . . . 111
6.4.2 10 TCP flows & 3 XCP competing flows . . . 111
6.4.3 10 XCP flows & 3 TCP competing flows . . . 113
6.5 Limitations and optimizations . . . 116
6.6 Conclusion . . . 117

7 Conclusions 121
7.1 Context and problems studied in this thesis . . . 121
7.2 Weaknesses and strengths of ERN protocols . . . 121
7.2.1 Benefits of any ERN protocol in LDHP Networks . . . 121
7.2.2 Why are ERN protocols not used in current networks? . . . 122
7.3 Propositions for a robust and inter-operable ERN protocol . . . 122
7.3.1 Enabling the inter-operability with non-ERN routers . . . 122
7.3.2 Enabling the inter-operability with E2E protocols . . . 123

7.3.3 Improving the robustness of ERN protocols 124
7.4 Generalization of our propositions 124
7.5 Perspectives 125

Bibliography **126**

Acronyms **135**

List of Figures

2.1 Forwarding network devices linking end hosts. 9
2.2 Congestion Collapse. 12
2.3 TCP Tahoe - The *cwnd* evolution. 15
2.4 TCP Reno - The *cwnd* evolution. 16
2.5 TCP Reno vs TCP New Reno. 17
2.6 AIMD vs AIAD vs MIMD. 18
2.7 $a(cwnd)$ and $b(cwnd)$ values for different congestion window size. 26
2.8 Scaling properties of Standard TCP and STCP (from [1]). 27
2.9 2 BIC flows (from [2]). 28
2.10 The XCP protocol. 39
2.11 4 XCP flows (from [3]). 42
2.12 4 JetMax flows (from [4]). 44

3.1 Sinusoidal-based bandwidth variation model. 49
3.2 Topology network for the sinusoidal-based bandwidth variation model. 50
3.3 Simple view of a TCP connection facing VBE. 51
3.4 More accurate TCP behavior with cases a, b and c. 52
3.5 TCP New Reno ($RTT \approx 100ms$) on a sinusoidal-based VBE. 53
3.6 TCP New Reno ($RTT \approx 200ms$) on a sinusoidal-based VBE. 54
3.7 HSTCP ($RTT \approx 100ms$) behavior on a sinusoidal-based VBE. 55
3.8 HSTCP ($RTT \approx 200ms$) on a sinusoidal-based VBE. 56
3.9 XCP ($RTT \approx 100ms$) on a sinusoidal-based VBE. 57
3.10 XCP ($RTT \approx 200ms$) on a sinusoidal-based VBE. 57
3.11 Step-based discrete bandwidth variation model. 58
3.12 Topology network for the step-based bandwidth variation model. 59
3.13 HSTCP ($RTT \approx 100ms$) on a step-based VBE. 60
3.14 Zooming on HSTCP behavior. 60
3.15 HSTCP ($RTT \approx 200ms$) on a step-based bandwidth VBE. 61
3.16 XCP ($RTT \approx 100ms$) on a step-based VBE. 62
3.17 Zooming on XCP behavior. 62
3.18 XCP ($RTT \approx 200ms$) on a step-based VBE. 63

4.1 (a) scenario for TCP, (b) and (c) scenarios for XCP. 66
4.2 Congestion window evolution and Throughput for scenarios a,b,c. 68
4.3 Discovering non-XCP clouds in the data path. 69
4.4 Discovering XCP-i edge routers. 72

4.5 An XCP-i router hosting 2 virtual routers. 75
4.6 Asymmetric deployment: optimized receiver side 76
4.7 Incremental deployment at peering point 77
4.8 XCP-i in the topology shown by Figure 4.7. 78
4.9 2 upstream non-XCP queues, $\Sigma input_capacity = output_capacity$ 79
4.10 Throughput of XCP-i on Topology of Fig. 4.9 79
4.11 2 upstream non-XCP queues competing with an XCP path, $\Sigma input_capacity > output_capacity$ 80
4.12 Throughput of XCP-i on Topology of Fig. 4.11 80
4.13 1 non-XCP queue shared by XCP-capable downstream nodes 81
4.14 Throughput of XCP on Topology of Fig. 4.13 82
4.15 Throughput of TCP on Topology of Fig. 4.9 82
4.16 XCP-i - Sender i - max error 20% - min error 5% 83
4.17 XCP-i - Sender j - max error 20% - min error 5% 83
4.18 1 bottleneck link shared by n XCP paths 85

5.1 Topology to test XCP on a VBE with ACK losses. 91
5.2 Variation model caused by ON-OFF UDP flows. 91
5.3 10 XCP flows on a step-based VBE without ACK losses. 92
5.4 10 XCP flows on a step-based VBE with 10% of ACK losses. 94
5.5 The XCP-r receiver algorithm. 95
5.6 10 XCP-r flows on a step-based VBE with 10% of ACK losses. 96
5.7 Average throughput of flows. 98
5.8 XCP - no ACK losses. 99
5.9 XCP - 30% ACK loss rate. 99
5.10 XCP-r - 30% ACK loss rate. 100

6.1 Topology: n XCP and m TCP flows sharing the bottleneck. 105
6.2 XCP flow dealing with TCP concurrent flows. 106
6.3 One XCP flow dealing with two concurrent TCP flows 112
6.4 3 XCP flows appear among 10 TCP flows 114
6.5 3 TCP flows appear among 10 XCP flows 115
6.6 3 TCP flows appear among 10 XCP flows - inspecting 50% of packets 118

Chapter 1

Introduction

1.1 Motivation and Problem statement

Traffic challenges in Long-Distance High Performance (LDHP) networks

Throughout the last two decades, Internet Protocol (IP) [5] networks have expanded and evolved in their physical structure (capacity and capability). The expansion of networks, produced by the interconnection of geographically large networks, have introduced large delays (the delay is defined as the time needed by a packet[1] to be transferred from the sender to the receiver). Thus, the delay between two computers can vary from only a few milliseconds to several tenths of seconds.

The evolution of the physical structure of networks, caused by the introduction of new technologies, like optical fiber or 10 Gigabit Ethernet over twisted pair cable, gave rise to networks with high capacity, able to transfer at the rates of several Gigabits per second (Gbps).

Thus, the expansion of networks and new technologies have produced networks with high capacity and large delay, called large *bandwidth-delay* product (BDP) networks, Long-Distance High Performance (LDHP) networks, or simply Long Fat Network (LFN). In LDHP networks the number of potential users can be incredibly high. Some examples of such LDHP networks are the interconnected Grid, like the one presented in the GEANT project [6].

At the same time, networks have become heterogeneous. This heterogeneity is caused mainly by (*i*) the large variety of applications, whose data flows[2] have different priorities and size, and (*ii*) the protocols enhanced to guarantee the good performance of such applications.

Thus, we can find in networks high priority flows which require Quality of Service (QoS) protocols able to guarantee a maximum of bandwidth, limited delay, etc. However, most of networks are used by low priority flows without guaranteed resources.

Since network resources are limited, such low priority flows must compete for them (specially bandwidth). As a result, the use of traffic control mechanisms to manage shared network resources is mandatory. In shared networks, the traffic control is assured by the congestion control mechanisms (congestion control protocols). Congestion control protocols have to (*i*) avoid congestion (overload of network resources) that causes losses of packets of data, while

[1] segment of informations with a fixed maximum length, transmitted as a discrete entity from one node on the network to another

[2] group of data packets circulating inside a network from one sender to one receiver

maximizing the utilization of available bandwidth, and (*ii*) achieve fair network-wide rate allocation (fairness) between users.

Furthermore, the available bandwidth could vary over time by the aggregation/disaggregation of high-priority flows, producing networks with variable available bandwidth for the lowest priority flows. Consequently, congestion control mechanisms must be robust and react quickly to the changes in the available resources (quickly grab or yield bandwidth), even in heterogeneous large BDP networks.

Limits in Congestion Control Solutions

Without congestion control mechanisms, the growth of the biggest networks, such as the already mentioned GEANT, or Internet, could not be possible. Without congestion control protocols, the aggregation of data flows can lead to serious congestion problems (many lost packets), that can result in inactivity periods of such flows.

Until now, many congestion control protocols, have been proposed. In a general way, congestion control protocols can be classified into (*i*) End-to-End (E2E) protocols [7] that implement congestion control mechanisms only in senders and receivers, (*ii*) Active Queue Management (AQM) mechanisms that implement congestion control only in the intermediate devices, and (*iii*) protocols that implement their congestion control mechanisms in each entity of the network: senders, receivers and the intermediate devices.

E2E protocols are mostly binary-based congestion control protocols, since they are based on the hypothesis that non-losses of data mean non-congestion (therefore, senders must increase their rate), and losses mean congestion (therefore, senders must decrease their rate). Transmission Control Protocol (TCP), in its New Reno version, is currently the most known E2E protocol. Due to the binary signal system used by E2E protocols, they produce congestion periodically (E2E protocols do not avoid congestion, but only decrease the impact of congestion). Other E2E protocols add new features to predict congestion, like delay variation monitoring systems, to the classical binary-based congestion control. That is why these E2E protocols are called multi-bit signal protocols.

However, both binary and multi-bit signal E2E congestion control protocols have proved unable to avoid congestion in networks, and lead to unfairness, when the senders sharing the bottleneck have different acceleration rate capacities or when flows start at different times. In addition, on most of E2E protocols, the increase in the delay multiplicatively increases the time needed to acquire the bandwidth in large BDP networks.

AQM mechanisms are based on the hypothesis that when the buffer occupancy level of intermediate devices (routers usually) exceeds a threshold, congestion could arrive at any time. Hence, after this threshold, AQM mechanisms punish senders by dropping their data packets with a given probability. The most known AQM mechanism is maybe Random Early Detection (RED). However, in some studies about AQM mechanisms, researchers have discovered that AQM protocols degrade the fairness level and do not avoid congestion, since some AQM mechanisms are not able to react quickly enough.

Usually, in protocols implementing their congestion control mechanisms in each entity of the network, the intermediate devices provide information which lets the senders adapt their rates more accurately, thus avoiding the losses of data. The information provided by intermediate devices arrives to the senders by mean of the receivers. Depending on the kind of information given by intermediate devices, routers-assisted protocols can be divided into:

- Explicit Congestion Notification (ECN) protocols: where intermediate devices provide

congestion binary signals (congestion/non-congestion) to the senders. ECN protocols are implemented on top of AQM mechanisms to detect "imminent" congestion. TCP-ECN [8] is the most known ECN protocol.

- Explicit Rate Notification (ERN) protocols: where intermediate devices compute and provide a value indicating the most adapted sending rate to the senders. eXplicit Control Protocol (XCP) [3] is the most known and most representative of ERN protocols.

Since ECN mechanisms are build on top of AQM mechanisms, ECN inherits most of the weakness of the AQM mechanisms, like fairness problems, and difficulty to really avoid congestion. On the other hand, ERN protocols seem to be promising approaches, since they get higher performance (faster adaptation to the network state) and fairness level, as shown in some simulations and experimental tests (see Chapter 2). In parallel, preliminary analysis of network devices manufacturers [9] shows that operations needed by ERN protocols could be implemented inside intermediate devices by using current technologies.

However, the problems of ERN protocols are based on the fact that this kind of protocols do not have any mechanism to interact with both E2E protocols and non-ERN-capable hardware devices, widely deployed in networks today. The absence of those mechanisms could degrade strongly the throughput of ERN protocols in heterogeneous networks, as showed in the next chapters. Thus, ERN protocols are not inter-operable in current heterogeneous networks. The gradual deployment of ERN protocols in current heterogeneous networks is therefore very difficult.

1.2 General Aim and Context of applications of our propositions

This thesis identifies the shortcomings in terms of inter-operability of the ERN protocols, and proposes solutions to mitigate such inter-operability problems. Our proposed solutions cover the mandatory steps to get an ERN protocol able to be deployed gradually in heterogeneous networks.

We want to remark that our solutions aimed at improving the cohabitation between non-ERN protocols (like E2E protocols) and non-ERN-capable hardware, with only one of the proposed ERN congestion control protocols.

However, our mechanisms could not be applied as such to any kind of networks. Our solutions concern mainly networks with the following characteristics:

- networks where long-life flows are necessary and frequent, due to the transfer of a large amount of data between geographically distributed sites,
- networks where it is not possible to guarantee a given amount of bandwidth to all flows, due to the high number of potential connections.
- networks with high capacity (1 Gbps or more),
- wired and/or optical networks.

Some networks with the characteristics mentioned above are Data Grid networks, used by some companies and scientific laboratories. In these networks, new applications have been developed and executed to solve complex problems. Frequently, these new applications need

to transfer a huge amount of data from one set of computer to another, in order to begin the processing and analysis of these data. For instance, in Health Sciences, medical images analysis applications need to transfer tens or hundreds of very high definition images (whose size can reach Gigabytes depending on the physical size and the resolution of the images), from databases of laboratories where the images are taken to databases of laboratories where they will be analyzed. The transfer of those image databases results frequently in the transfer of hundreds of Gigabytes. Other scientific areas, like Earth Science, Astronomy, etc., use also applications that need to transfer a large amount of data before processing it.

Thus, the approaches presented in this thesis are not intended for mainstream-oriented heterogeneous networks, like Internet, and could not be applied as such in wireless or satellite based networks.

In addition, our solutions concern mainly long-life flows without guaranteed resources (known also as best-effort traffic or best-effort flows), even if our solutions can also be used for long-life flows with QoS mechanisms. Therefore, the improvement of the throughput of small data flows, like those transferred by the HyperText Transfer Protocol (HTTP), the Secure SHell (SSH) application, etc., are also beyond the scope of this thesis.

1.3 Contributions

This thesis is focused mainly on (*i*) the identification and analysis of the inter-operability problems of the ERN protocols, and (*ii*) the development of solutions to the inter-operability problems, as well as the validation of such solutions. This way, we aimed at enabling the gradual deployment of ERN protocols in heterogeneous LDHP networks.

In a general way, we can group our contributions into the following 4 points:

1. Solutions to enable the inter-operability of ERN protocols with existing equipments. We present a set of propositions that enable ERN protocols to maintain their performance in presence of hardwares not implementing their required mechanisms. Those hardwares are generally classical IP hardware devices, like DropTail, RED, etc. It is mandatory to be able to use current non-ERN-capable hardwares in combination with hardwares specially designed for ERN protocols, since we cannot replace every non-ERN-capable hardware by an ERN-capable hardware instantaneously.

2. Solutions to enable the inter-operability of ERN protocols with existing E2E protocols. We present a set of propositions that enable ERN protocols to maintain their performance in presence of E2E congestion control protocols, as TCP. Currently, ERN protocols do not tolerate the presence of other congestion control protocols (inter-protocol fairness).

3. Models and mechanisms for improving the robustness of ERN protocols against the losses of the information provided by the routers. The execution of different mechanisms inside heterogeneous LDHP networks, as well as the burstiness nature of senders could make the environment highly aggressive against active senders, causing frequent loss of data packets. Thus, loss of information (about the senders rate) sent from the routers to the senders could happen, delaying the changes in the sending rate when such changes are necessary. Our propositions are aimed at armoring ERN protocols in presence of losses of information provided by the routers.

4. Bandwidth variation models for testing the capacity of adaptation of both E2E and ERN protocols, in response to the changes in the available bandwidth, introduced by the mechanisms running inside the networks. Bandwidth variations are caused by flow aggregation and/or QoS mechanisms implementing any resource reservation protocol. Hence, congestion control protocols must be able to quickly give back and grab bandwidth dynamically.

There is no ERN protocol considered yet as the "winner" or the "best". For this reason, all the propositions made during this thesis enabling the inter-operability of ERN protocols in current heterogeneous networks, have been designed to be implemented to any ERN protocol. But, at the time when our researches began, XCP was the best-known freely available ERN protocol (Internet Engineering Task Force (IETF) draft, modules for ns2 [10], Linux kernel modules, etc). Consequently, all our theoretical and experimental analyzes of the inter-operability problems concerning ERN protocols were focused on XCP.

1.4 Thesis Outline

This thesis is organized as follows: in the next chapter (Chapter 2), we introduce a set of concepts used in this thesis (end hosts, forwarding devices, congestion collapses, etc.), as well as a brief summary of the congestion control mechanisms. Firstly, we present the congestion control approaches proposed at the beginning of the congestion control protocols history in IP networks, when the bandwidth was limited to a few Megabits per second. Then, we present the protocols specially designed for networks with large bandwidth and large delay (LDHP networks). We explain why congestion control protocols for low speed networks cannot be used successfully in LDHP networks. Finally, we present the congestion control strategies that include algorithms in routers: (*i*) the AQM mechanisms, that only implement code in routers; (*ii*) protocols implementing binary signal congestion notification, from routers to the end users; and (*ii*) protocols providing ERN from routers to senders.

In Chapter 3, we test and compare some congestion control protocols over heterogeneous LDHP networks. The studied protocols are TCP, taken as the point of reference since this is currently the most used protocol; High Speed TCP (HSTCP), since at the time when our researches were made, this protocol was the most promising and the most representative of the congestion control protocols for LDHP networks; and XCP, which is the most widely known of the ERN protocols.

Chapter 3 aims to prove that XCP obtains better performance than TCP and HSTCP even under the most difficult conditions. Those difficult conditions were created by varying the network resources (bandwidth and routers buffer occupancy mainly) in the bottleneck of our topologies. Such variations simulate aggregation of flows in very active networks (e.g., as it should be in interconnected grids).

The research to identify the limitations of ERN protocols in terms of inter-operability begins in Chapter 4. In Chapter 4, we highlight the problem of XCP, when non-XCP capable routers are located between the sender and the receiver. Our proposed solution concerns the extension of the XCP routers capabilities, in order to be able to detect non-XCP routers and to take relevant decisions.

XCP can lose some of its advantages (reactivity, high resource usage...) when it coexists with other protocols. For instance, in networks totally controlled by the XCP protocol, the loss of data packets should be insignificant (almost 0% of losses). However, in networks not

controlled by XCP at 100%, the level of packet lost should be higher. For this reason, in Chapter 5, we analyze the impact of losses of Acknowledgments (ACKs) since they carry important information of the network state to the senders. In fact, the probability to lose packets increases when XCP does not totally control the networks. Chapter 5 presents a new architecture which improves the robustness of XCP protocol.

When XCP coexists with E2E congestion control protocols, XCP senders could see their performance seriously degraded. This phenomenon is caused by the fact that XCP routers (and most of the routers-assisted congestion control protocols providing ERN) monitor the input traffic rate and the output link capacity where they are placed, in order to make decisions about the rate of senders. However, XCP routers are unable to distinguish between XCP and non-XCP traffic, and to assign to every kind of protocol the right amount of resources. Our analysis and proposed solution are presented in Chapter 6.

Finally, in Chapter 7, we summarize the results obtained during this thesis, and we provide concluding remarks.

Chapter 2

State of the Art

This thesis presents our work on obtaining an inter-operable router-assisted congestion control protocol providing ERN, able to be implemented in heterogeneous LDHP networks. However, to understand the set of contributions given in this thesis, we need to clarify some important points, like:

- what are congestion control protocols?
- why congestion control protocols are important in networks?
- how have congestion control protocols evolved over time and what motivated such evolutions?
- what are congestion control protocols with assistance from routers?

Therefore, this chapter aims to answer these questions. However, before going to the questions related to congestion control protocols, first, we will begin by describing the required physical (end hosts, and forwarding network devices) and logical (transport and congestion control protocols) components to establish a communication. In our case, the communication will be basically the exchange of data between two computers, inside an IP network. We will introduce the reader to the transport and congestion control protocols area, where we will put special emphasis on the importance of congestion control in the networks.

Later on, we will focus our attention mainly on congestion control protocols, which is the main subject of studies of this thesis. We will then give a brief description and discussion of the current existing approaches aiming to control the congestion in IP networks. To do this, first of all we will concentrate on the congestion control approaches specially designed to work in the end hosts (E2E protocols). We will classify these approaches into two main groups: propositions made at the beginning of the history of congestion control protocols, when the bandwidth in network was limited to a few Megabits per second, and propositions made for networks with large bandwidth and large delay (large BDP networks). Such classification was motivated by the fact that large BDP networks increase the challenges to the congestion control protocols. Therefore, congestion control protocols for LDHP networks need to solve problems not present in low speed networks, thus making it impossible to evaluate protocols for low speed networks and protocols for LDHP networks under the same parameters.

Finally we will present the congestion control approaches specially designed to work in forwarding network nodes. These approaches are based on the idea that congestion occurs in

the forwarding network devices and therefore, E2E approaches will never be able to detect congestion successfully without the collaboration of these devices. Currently, many propositions for congestion control in forwarding network nodes exist. Thus, in this chapter, we will classify these propositions into (*i*) approaches that only try to avoid congestion without requiring the collaboration of end hosts; (*ii*) approaches that try to avoid congestion providing binary signals (congestion / non congestion) to the users; and finally (*iii*) approaches that avoid congestion by managing directly the rate of end hosts (ERN protocols).

2.1 Essential Components in Network Communication

The networks can be divided into Layer 2 [11] networks (circuit-oriented networks), like Asynchronous Transfer Mode (ATM) [12] or Frame Relay [13] networks, and Layer 3 [11] networks (connectionless networks), like IP networks. The main goal of networks is to support communications between devices. In our case, the communication will basically consist of exchange of data between two devices.

In Layer 2 or Layer 3 networks, we can divide the main physical components of a network, that are necessary to establish a data flow between two devices, into two main categories: end hosts and forwarding network nodes.

2.1.1 End hosts

In this thesis, we will define the exchange of data as the transfer of packets of data (that we will refer to as a data flow) from a sender device (also named sender host or simply the sender) to a receiver device (also named receiver host or simply the receiver). Both sender and receiver nodes are called also end hosts or end users.

2.1.2 Forwarding nodes: Layer 2 and Layer 3 devices

In networks, the sender and the receiver are not always directly connected, since they can be geographically apart. They are then interconnected by devices called forwarding network devices, forwarding network nodes or intermediate nodes.

The path followed by a data packet (from the sender to the receiver) is composed by n intermediate devices, where $n \geq 0$. These devices can be layer 2 devices (that include network interface cards, hubs, bridges and switches) or layer 3 devices (routers generally). Layer 2 devices can be used in both circuit and connectionless networks. Layer 3 devices are used only in connectionless networks.

Forwarding devices have to forward packets in order to let them reach the receiver node.

2.2 Transport and Congestion Control Protocols

2.2.1 Importance of Transport Protocols

In order to provide reliability and successfully exchange data between end users, the use of flow controls is mandatory in networking.

In layer 2 networks, like ATM or Frame Relay, the flow control (data flow control) is made by the application in end hosts. We will not go into the details of mechanisms used in layer 2 networks since this is not the subject of this thesis.

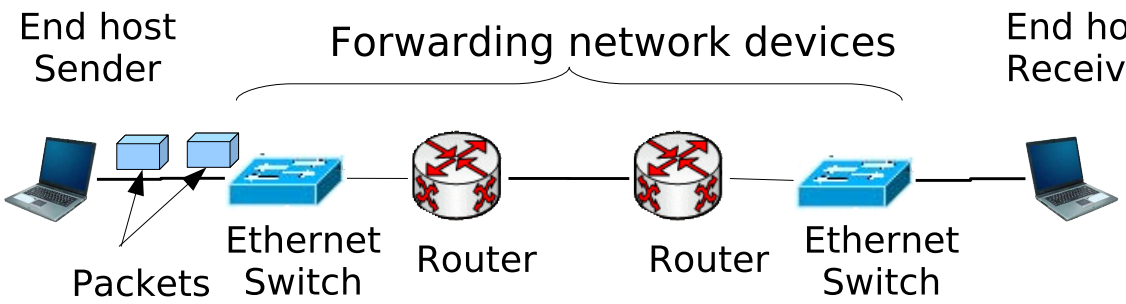


Figure 2.1: Forwarding network devices linking end hosts.

In layer 3, like IP networks, flow control is ensured by Transport protocols implemented in end hosts. Transport protocols provide flow control between senders and receivers, make multiplexing and splitting of connections, and can provide error detection and recovery. Two of the main transport protocols in IP networks: TCP [14] and User Datagram Protocol (UDP) [15].

TCP

TCP, defined originally in [14] with clarifications and bug fixes in [16], is a "*connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks*".

Since TCP is executed on top of a less reliable communication system, TCP requires facilities in the following areas:

1. Basic Data Transfer: TCP is able to transfer a continuous stream of data packets deciding when to block and forward packets at their own convenience (managed generally by congestion control protocols).

2. Reliability: To provide reliability, TCP assigns a sequence number to each transmitted segment, and requires an ACK from the receiver. When an ACK is not received by the sender host, the data is retransmitted (when and how a packet is retransmitted is dictated by the Congestion Control Protocols discussed later in Sections 2.3, 2.4, and 2.5). At the receiver side, the sequence numbers are used to order data packets that can be received out of order and to eliminate duplicated packets. However, since data packets can be corrupted in the path, TCP adds a checksum to each transmitted packet, which is used by the receiver to check the consistence of the data packet received. Thus, the receiver can discard damaged segments.

3. Flow Control: TCP provides mechanisms to control the rate of data entering the network by mean of a "window", which indicates an allowed number of bytes that the sender may transmit before receiving further permission. The flow control is referred also as congestion control. The goals of the Congestion Control mechanisms are shown later in Subsection 2.2.2.

4. Multiplexing: TCP provides a set of ports to allow many connections within a single host. The concatenation of the network, host and port addresses forms a socket. A pair of sockets uniquely identifies each connection. Thus, a socket may be simultaneously used in multiple connections.

5. Connections: In order to provide reliability and flow control, TCP maintains a certain status information for each data stream (for instance, ESTABLISHED, CLOSED, etc.). The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is identified by the concatenation of both the receiver and the sender sockets.

Due to the reliability provided by TCP, many applications where the delivery of messages must be ensured, like File Transfer Protocol (FTP), Telnet, SSH, between others, use TCP as the Transport Protocol.

User Datagram Protocol

UDP, defined in [15], is one of the core protocols of the IP suite. Using UDP, programs on networked computers can send short messages known as datagrams (using Datagram Sockets) to one another. UDP is also called the Universal Datagram Protocol or Unreliable Datagram Protocol.

UDP does not provide the reliability and ordering that TCP provides. Datagrams may arrive out of order, appear duplicated, or go missing without notice. Without the overhead of checking whether every packet actually arrived, UDP is faster and more efficient for many lightweight or time-sensitive purposes. Also, its stateless nature is useful for servers that answer small queries from huge numbers of clients. Compared to TCP, UDP is required for broadcast (send to all on local network) and multi-cast (send to all subscribers).

Common network applications that use UDP include streaming media applications such as IPTV, Voice over IP (VoIP), and on-line games. However, currently, these kinds of applications have replaced UDP by Real-Time Transport Protocol (RTP) [17]. RTP is a Transport Protocol for Real-Time Applications and is built on top of UDP.

Mixing TCP and UDP properties

Since TCP provides a strong reliability and congestion control, it introduces generally an important delay in the transfer of data. TCP is therefore recommended in networks where resources are not guaranteed and where the use of congestion control mechanisms are essential for avoiding collapses that could affect seriously the transfer of data (see Section 2.2.2).

On the other hand, UDP is faster. However, UDP is only recommended in the transfer of data generated by Real-Time Multimedia Applications, where the loss of data is not important. RTP built on top of UDP is the most used protocol in Real-Time Application.

However, there are some special cases, where TCP or UDP cannot provide the most adapted services. One example of such a case, is the transfer of files between storage devices connected by mean of dedicated channels (non-shared links), able to guarantee a certain quantity of bandwidth to the end users. In this example, TCP is not really adapted since the congestion control mechanisms are not necessary, and the delay introduced by these algorithms could be expensive in terms of time. UDP is not adapted either, since it does not provide reliability, or the reliability mechanism is not adapted for these kinds of situations.

The solution is therefore Transport Protocols including TCP reliability properties and the data sending manner of UDP (which is faster). There are many propositions with these characteristics. Some of these protocols mixing TCP and UDP properties are Reliable Blast UDP (RBUDP) [18], Tsunami [19], Hurricane [20], between others.

Even though most protocols mixing TCP and UDP properties do not implement fairness mechanisms (since they are designed to work in dedicated channels), there are some protocols able to provide also bandwidth fair share and that could hence be used in shared networks. We will describe in more details some protocols mixing TCP and UDP properties that could be used in shared networks in Section 2.4.

Finally, we can mention Reliable UDP (RUDP) [21, 22], which is used to provide reliability specifically adapted for telephony signaling.

2.2.2 Importance of Congestion Control Protocols

In order to understand why congestion control is necessary we will explain first what a congestion is and how it affects the performance of the connections.

A congestion occurs when active flows demand more resources than the network devices and links can provide. Specifically, congestion implies that the sender rate is exceeding the capacity of the bottleneck[1] in the path.

At first, the overload of data packets sent is stored in the buffer of the forwarding network node. However, if the sending rate does not decrease, the buffers will grow until packets are lost. Contrary to the general thought, increasing the capacity of networks does not solve the problem, since the number of active flows can grow very high and the total of sent data packets could easily exceed many Terabits per second. Thus, without a good congestion control mechanism, congestion problems can produce a phenomenon known as *congestion collapse* (see Figure 2.2), which consists in a drastic reduction of the throughput of the sender, caused by an important number of dropped packets in the bottleneck.

We can easily identify two important points before the congestion collapses (Figure 2.2). The first one is the *knee*. The *knee* is the point where the load begins to exceed the link capacity and the buffer of devices placed in the bottleneck stores the exceeding load. After the *knee*, the delay (defined as the time needed by a packet to be transferred from the sender to the receiver) grows exponentially. The second one is the *cliff*. After the *cliff*, the exceeding load cannot longer be stored in buffers and consequently, packets are dropped.

Hence, there are two main goals of congestion control mechanisms: (*i*) maximize the utilization of the links capacity while avoiding the network congestion (maintain flows below the *knee*) and (*ii*) recovering quickly from congestion to avoid congestion collapses (avoid imperatively the *cliff*).

But the work of congestion control is still more complex. Congestion controls need to guarantee also the fair share of the network resources to the end users. The fairness is not a less important goal, since a very aggressive flow could degrade strongly the performance of others flows. Congestion control mechanisms have to provide to the competing flows a *max-min fairness*. An intuitive description of the max-min fairness criterion is that flows restricted at the same bottleneck link are entitled to the same amount of resources.

[1] The bottleneck can be defined as the point in the network where a data flow meets the smallest available bandwidth.

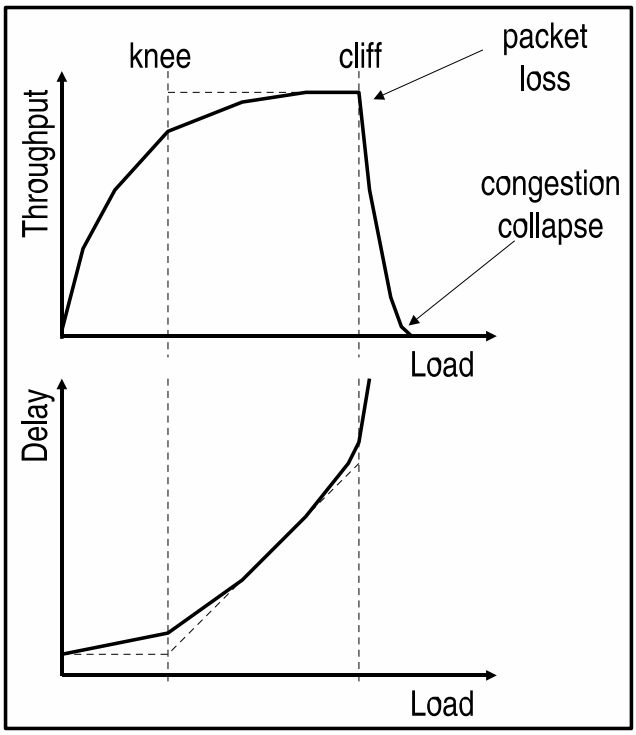


Figure 2.2: Congestion Collapse.

Congestion Control mechanisms in Layer 2 networks

Congestion controls are actually needed in all kinds of networks: Layer 2 and Layer 3 networks. In layer 2 networks, forwarding network devices are the main components in charge of providing congestion control mechanisms and fair share of the network resources. Therefore, forwarding network nodes have to forward the packets from the sender to the receiver, while providing congestion control and sharing networks resources.

Since this thesis is focusing on IP networks, we will not go into details of congestion control mechanisms used in layer 2 networks.

Congestion Control mechanisms in IP networks

In IP networks both congestion control and fair share of resources are provided mainly by Congestion Control Protocols implemented in end hosts, even though some approaches especially designed for routers have also been proposed, as we will see in Section 2.5.

Since this thesis is focused specifically on congestion control protocols in IP networks, we will highlight this topic and we will show in the next sections the evolution of the Congestion Control Protocols. We will present therefore a brief summary and discussion of congestion control protocols, from standard protocols to newest propositions. However, the evolution of congestion control protocols cannot be understood without taking into account the evolution of technologies in networking and the transition from low to LDHP networks, as both of them have lead to new challenges to the congestion control protocols. We will then classify Congestion Control Protocols into (i) standard protocols (Section 2.3), proposed at the beginning

when the available bandwidth was limited to a few Megabits per second and (*ii*) congestion control protocols proposed for LDHP networks (Section 2.4). Finally, in Section 2.5 we will present some propositions that move congestion control mechanisms from end hosts to the routers.

2.3 Standard Congestion Control Protocols

In this thesis, we will refer to as Standard Congestion Control Protocols, the most known of first TCP variants, which were proposed when the capacity of the networks was lower than 1Gbps: TCP Tahoe, TCP Reno, TCP New Reno and TCP Vegas.

2.3.1 From TCP Tahoe to TCP New Reno

The study of Congestion Control Protocols in IP networks began in October 1986, when the Internet suffered the first of a series of congestion collapses. At that time, the congestion collapses were originated by the increase in the number of active flows crossing the Internet.

In 1988, after an important mathematical analysis of the congestion collapses, V. Jacobson proposed the use of the first congestion avoidance and control protocol to solve the problem [23]. The well known TCP Tahoe protocol was born.

2.3.1.1 TCP Tahoe

The proposed TCP Tahoe was based mainly on the use of the following algorithms: Slow Start, Congestion Avoidance, Fast Retransmit and the Retransmission TimeOut (RTO) estimation.

Slow Start

The Slow Start algorithm is executed at startup of every connection. Slow Start always begins with a congestion window size (also called *cwnd*) equal to 1 Maximum Segment Size (MSS). The MSS is defined as the maximum size of a TCP packet.

The main ideas behind Slow Start are:

1. To avoid putting too much stress on the network when a connection starts.
2. To acquire quickly the available bandwidth.
3. To know the limit of the congestion window size growth.

The Slow Start algorithm relies on the increase of the congestion window size by 1 packet for every ACK received (see equation 2.1), as long as a certain value (the slow start threshold, *ssthresh*) is not reached. Note that using the Equation 2.1 during Slow Start, *cwnd* is doubled every Round Trip Time (RTT). The RTT is defined as the time elapsed from the sending of a packet until the reception of its ACK by the sender).

$$cwnd = cwnd + 1 \tag{2.1}$$

Congestion Avoidance

When the congestion window size is above the limit imposed by *ssthresh*, Congestion Avoidance is executed. The reason is that, after exceeding *ssthresh* a congestion could happen at any time.

During Congestion Avoidance, the congestion window is increased by only 1 packet (Additive Increase) at every RTT. This phase will end in case of loss of packets only.

The growth of the congestion window size at every ACK received by the sender is given by the Equation 2.2

$$cwnd = cwnd + \frac{1}{cwnd} \tag{2.2}$$

Fast Retransmit

The main goal of the Fast Retransmit phase is to resend a data packet after it was lost for any reason. This mechanism is based on the following assumptions:

When an i^{th} packet has already been sent,

a) at the reception of the first ACK asking for this packet, the sender supposes that the next ACK will ask for the $(i+1)^{th}$ packet.

b) at the reception of the first Duplicated ACK (DUPACK) asking again for the i^{th} packet, the sender can assume that the receiver has gotten the packets in wrong order.

c) at the reception of the second DUPACK, the sender assumes that the i^{th} packet may have been delayed in the path (sender is insecure about the loss of the i^{th} packet).

d) at the reception of the third DUPACK, the sender assumes that the i^{th} packet is lost and therefore, the sender must resend it (Fast Retransmit).

After resending a packet:

1. The threshold of the Slow Start phase is updated to the middle (Multiplicative Decrease) of the last congestion window size ($ssthresh = cwnd/2$).

2. The connection is restored, by setting the congestion window size to 1 packet ($cwnd = 1$).

RTO Estimation

As described earlier, when a data packet with $id = i$ is lost, for every received data packet with $id > i$, the receiver node will send an ACK asking for the i^{th} packet. After three DUPACKs, the sender executes Fast Retransmit in order to recover the lost packet.

However, in case where the congestion window size is not large enough and no more data packets are sent after the i^{th} packet, the sender will not receive three DUPACKs and Fast Retransmit will not be executed. In this case, after the expiration of the timer called RTO, the sender will resend the non acknowledged packet.

In his researches, Jacobson observed that the estimated RTO, as described in [14], produced a significant amount of retransmitted packets only delayed in the path, but not lost. The equation 2.3 shows the Jacobson's improvements to prevent this problem.

$$RTO = SRTT + K * RTTVAR \tag{2.3}$$

where

$$SRTT = (1 - \alpha) * SRTT + \alpha * R' \tag{2.4}$$

and

$$RTTVAR = (1 - \beta) * RTTVAR + \beta * |SRTT - R'| \tag{2.5}$$

In Equation 2.3, $K = 4$, $SRTT$ is the Smoothed RTT (calculated as shown in Equation 2.4) and the $RTTVAR$ is the RTT variation (calculated as shown in Equation 2.5). In the equations 2.5 and 2.4, $\beta = 1/4$ and $\alpha = 1/8$ respectively, and R' is the last measured RTT. With the improvements of Jacobson, the number of retransmission of packets only delayed in the path, but not lost, decreased significantly. To learn more about the problems in RTO estimation before the improvements of Jacobson, please refer to [16]. To learn more about the Jacobson's RTO algorithm, please refer to [24].

The set of algorithms incorporated into the TCP Tahoe protocol produces a typical saw-tooth behavior of the *cwnd*, as the one shown in Figure 2.3.

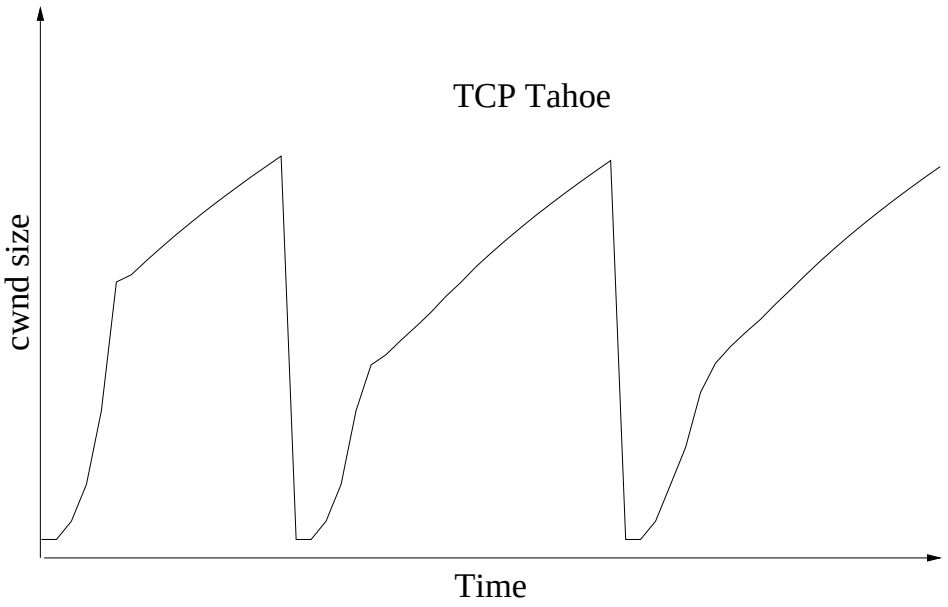


Figure 2.3: TCP Tahoe - The *cwnd* evolution.

2.3.1.2 TCP Reno

With the introduction of TCP Tahoe, the problem of collapses in Internet was (potentially) solved. However, some problems of underutilization of the available resources were noticed. Basically, these problems were caused by the TCP Tahoe algorithm that restores the connection after every retransmitted packet, updating the congestion window to 1 packet and executing the Slow Start phase, as observed in Figure 2.3.

The new researches to improve TCP Tahoe gave rise to a new TCP version called TCP Reno [25]. TCP Reno kept all the TCP Tahoe mechanisms, but it included a new mechanism

called Fast Recovery. Fast Recovery is always executed after Fast Retransmit and consists mainly in the execution of Congestion Avoidance. Since Slow Start is not executed, *cwnd* is not reinitialized, but halved (see equation 2.6).

$$cwnd = cwnd/2 \tag{2.6}$$

Slow Start is reexecuted only when a flow suffers a Timeout. Figure 2.4 shows the typical behavior of the TCP Reno *cwnd*. Figure 2.4 shows us also why TCP is called an Additive Increase Multiplicative Decrease (AIMD) protocol: in absence of losses the increment of *cwnd* is additive ($cwnd = cwnd + 1/cwnd$), but in case of losses the decrement is multiplicative ($cwnd = cwnd/2$). Thus, TCP Reno improved the average throughput of TCP Tahoe flows.

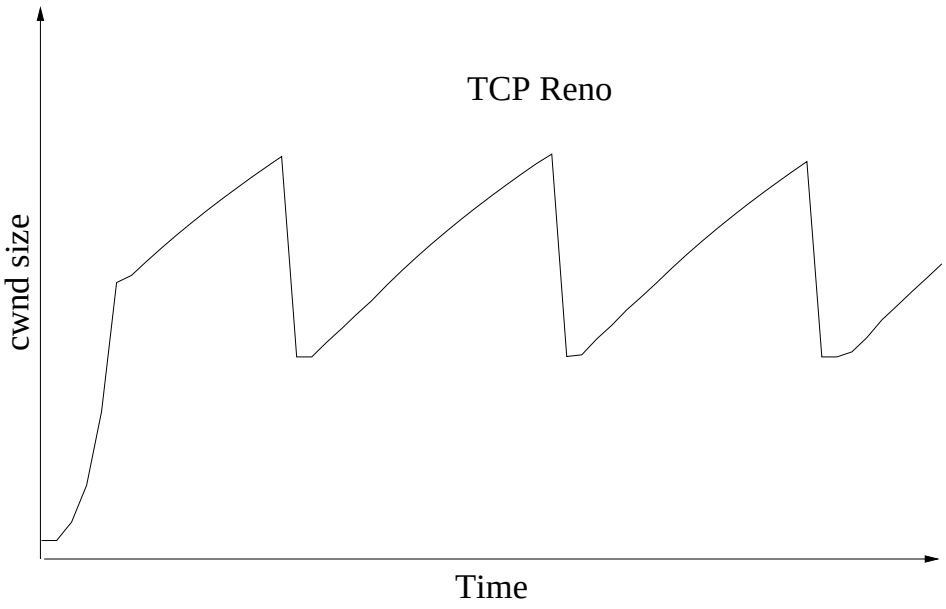


Figure 2.4: TCP Reno - The *cwnd* evolution.

2.3.1.3 TCP New Reno

TCP Reno succeeded in improving the average throughput of TCP Tahoe when a few packets are lost during the connection. However, a new problem appeared: when a connection loses a large amount of packets belonging to the same congestion window, TCP Reno executed Fast Retransmit several times, hence finishing with a very small window and, in addition, executing the slow Congestion Avoidance phase.

TCP New Reno [26] was proposed to overcome this problem, by introducing some important changes to the Fast Retransmit/Fast Recovery mechanism of TCP Reno. With TCP New Reno, when a connection comes into the Fast Retransmit phase, this phase does not finish as long as the sender knows that a packet from the same congestion window needs to be retransmitted. Figure 2.5 shows the congestion window of a connection using TCP Reno and a second one using TCP New Reno. The first time *cwnd* is halved (only one packet was lost) both TCP Reno and New Reno have similar behavior. However, the second time *cwnd* is reduced, when two packets were lost, the difference between TCP Reno and New Reno is

easily observed. While TCP Reno needs to execute twice Fast Retransmit/Fast Recovery, New Reno executes Fast Retransmit/Fast Recovery only one time to recover both lost packet.

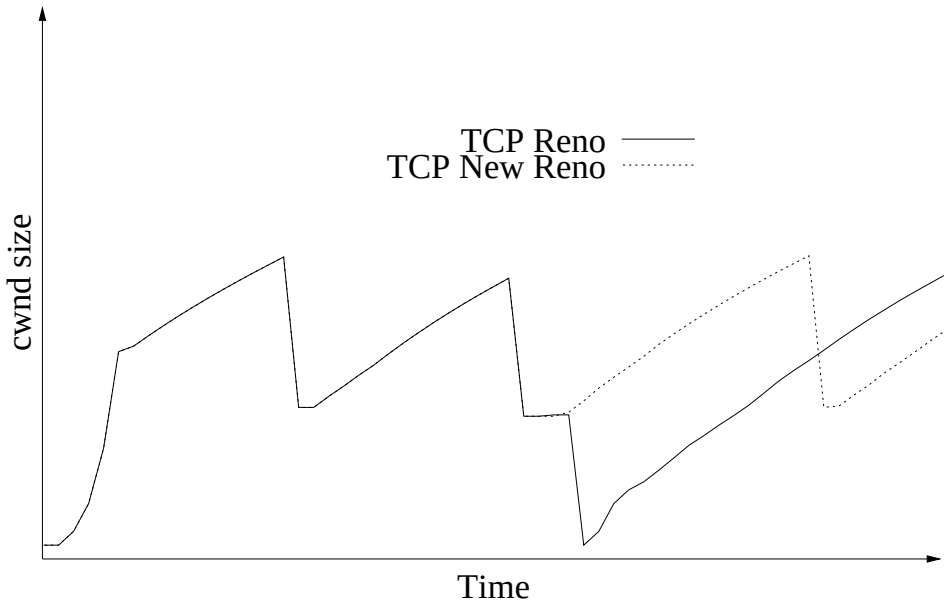


Figure 2.5: TCP Reno vs TCP New Reno.

These modifications produced a more robust TCP version, that improved the performance of flows in presence of multiple packet losses belonging to the same congestion window. For a more detailed information, please refer to [26].

2.3.1.4 Limitations of TCP Tahoe, Reno and New Reno

Fairness

TCP Tahoe, Reno and New Reno use the algorithm AIMD to achieve fairness between active flows. AIMD was accepted by the community as the best strategy for this purpose.

Others strategies for achieving fairness are Additive Increase Additive Decrease (AIAD), Multiplicative Increase Multiplicative Decrease (MIMD) between others. Comparative studies of such strategies can be found in the literature. AIMD always showed to be the more adapted to offer fair share to end users since it converges to the fairness line, while the others away from the fairness line (see Figure 2.6). The more important work in this field was presented in the well known article of R. Jain and D. Chiu [27].

However, it was proved as well that AIMD does not converge to the fairness equilibrium when flows with different RTT share the resources, as reported in [28]. New minor modifications have been proposed to AIMD therefore to improve the fairness ([29, 28] between others) and the research continues in this area. The problems concerning fairness are then open, since, even if TCP New Reno offers good stability and fairness in some scenarios, it could become very unstable in some others.

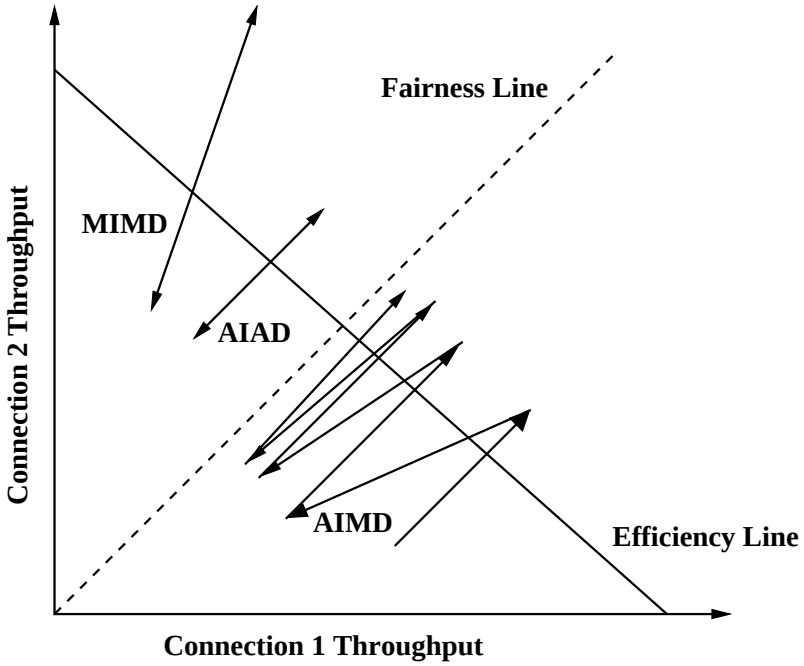


Figure 2.6: AIMD vs AIAD vs MIMD.

Performance

TCP, especially in its New Reno version, succeeded in getting a good performance while sharing successfully the resources, in networks with small delays and where the bandwidth is limited to a few Megabits per second. The advantages offered by the use of TCP New Reno resulted in a world-wide development of Internet.

However, the Congestion Avoidance mechanisms of TCP Tahoe, Reno or New Reno have the characteristic of always leading to congestion in the network, since these three protocols use a loss-based binary signal system for estimating the sending rates :

1. No losses: No congestion. The sender can increase its sending rate.
2. Losses: Congestion. The sender has to reduce its sending rate.

The result is a congested network, with periodical losses and under-utilization of the resources, every time losses are detected. This phenomenon marked the beginning of a continuous search for a congestion control protocol able to stay stable in the *knee* point and hence, able to improve the performance of TCP Tahoe, Reno and New Reno.

2.3.2 TCP Vegas: towards a more intelligent TCP version

TCP Vegas [30] is a TCP based protocol proposed by Lawrence S. Brakmo. TCP Vegas modifies the RTT calculation strategy proposed for TCP Tahoe. TCP Vegas proposes to calculate the average RTT by recording the time when a packet is sent and comparing this value with the time at which the respective ACK is received. This mechanism does provide

TCP Vegas with a more accurate RTO estimation. In addition, when a DUPACK is received, Vegas checks if the difference between the current time and the timestamp recorded for the relevant segment is greater than the timeout value. If the answer is positive, TCP Vegas retransmits the segment without waiting for three DUPACKs.

But the more important change made by TCP Vegas was the use of a "multi bit signal" system, that allows TCP Vegas to detect a congestion before the loss of a packet and adapt the congestion window of the sender to the new conditions. The algorithm that predicts the congestion events is based on the hypothesis that the increase of the RTT indicates a congestion in the bottleneck.

TCP Vegas (*i*) linearly increases or decreases; (*ii*) or leaves unchanged its congestion window, according to a $BaseRTT$ (that is actually the minimum of all measured RTTs), the current sending rate and a expected sending rate (Algorithm 1). Since TCP Vegas is able to leave unchanged its sending rate when the current rate is approximately equal to the expected rate (the optimal sender rate), TCP Vegas could remain stable in the intersection point of the fairness line and the efficiency line, hence eliminating the unstable behavior of TCP (Figure 2.5). The Vegas algorithm was strongly inspired from the *Tri-S* scheme [31] and the Congestion Avoidance using RTT Delay (CARD) protocol [32]. In case of losses TCP Vegas reacts as a standard TCP: entering in the fast recovery phase and halving its congestion window.

Data: $BaseRTT$: the smaller measured RTT
Data: α: 30KB/s
Data: β: 60KB/s
ExpectedRate = *CongestionWindow* / *BaseRTT*
Source calculates current sending rate (ActualRate) once per RTT
Diff = *ExpectedRate* - *ActualRate*
if Diff $< \alpha$ **then**
    Increase *cwnd* linearly
else
    if Diff $> \beta$ **then**
        Decrease *cwnd* linearly
    else
        cwnd unchanged.
    end
end

Algorithm 1: The TCP Vegas algorithm.

TCP Vegas was proposed to avoid the sawtooth behavior of TCP New Reno. However, it was soon realized that TCP Vegas behaves like Reno when the available bandwidth is not sufficiently large or the traffic in the network is very hard [33].

In addition, in [34] the authors show the incompatibility of TCP Vegas and Reno. It means that when TCP Vegas and Reno share the bandwidth, TCP Vegas could have a strongly degraded throughput compared to TCP Reno.

Finally, TCP Vegas uses a Delay-Based Early Congestion Detection (DBECD) mechanism, which has been carefully analyzed. Those analyzes show that the use of DBECD mechanism could not be taken as the key solution for detecting and avoiding congestion (see Subsection 2.4.4).

2.3.3 Limitations of Standard Protocols in LDHP Networks

The problems of TCP Vegas listed above prevented the deployment and utilization of this protocol. Instead, the TCP New Reno version was the main protocol in charge of providing congestion control and fairness in heterogeneous networks. However, the introduction of new technologies [35] increased enormously the capacity of the networks. This resulted in networks able to transfer many Gigabits per second [36, 37]. In parallel, the heterogeneity was increased as well, connecting wired networks with wireless networks or even satellite networks [38, 39]. All these factors gave rise to the large BDP networks.

In addition, a large variety of new scientific applications were developed to solve complex problems of research in different areas. These new applications can be classified as follows:

1. Applications that transfer a very large quantity of data (thousands or hundreds of Terabytes) from many sites distributed in different parts of a same country, or even, in different countries (e.g.. Astronomic applications, Earth Science applications, etc.).

2. Applications that, due to the incredible numbers of processes, cannot be executed over one computer only. These applications need to synchronize the processes by exchanging several small messages between CPUs, which can be physically far.

In both cases, TCP Tahoe, Reno or New Reno fail in getting a good performance. In the remaining of this thesis, to avoid enumerating the three TCP basic versions (Tahoe, Reno and New Reno) and to simplify the reading of the thesis, we will refer to these protocols as standard TCP or simply TCP.

In the first case, TCP is not able to get a good performance because the throughput depends upon the product of the transfer rate and the RTT. As shown in [40], "*for a Standard TCP connection with 1500-byte packets and a 100 ms round-trip time, achieving a steady-state throughput of 10 Gbps would require an average congestion window of 83,333 segments, and a packet drop rate of at most one congestion event every 5,000,000,000 packets (or equivalently, at most one congestion event every 1 2/3 hours). The average packet drop rate of at most* 2×10^{-10} *needed for full link utilization in this environment corresponds to a bit error rate of at most* 2×10^{-14}*, and this is an unrealistic requirement for current networks*".

In the second case, the problem of TCP is based on the use of the Slow-Start algorithm. As described in Subsection 2.3.1, at the beginning, TCP initializes *cwnd* to 1 packet. After that, *cwnd* increases in an exponential mode at every RTT. This is a good idea for bulk data transfers. However, when we need to send just a few packets, it could necessitate many RTTs in order to get a congestion window size large enough to send all the required data packets.

New congestion control protocols have therefore been proposed to replace standard TCP in large BDP networks. In the next subsection we will present first some considerations to take into account to evaluate such protocols. Then, we will list some congestion control protocols for LDHP networks.

2.4 Advanced Congestion Control Protocols for LDHP Networks

Congestion control protocols for LDHP networks (also known as high speed protocols) are generally based over standard TCP, modifying mainly the Congestion Avoidance algorithm.

However, there are also some UDP-based protocols especially designed to provide high performance in BDP networks, implementing congestion control mechanisms able to provide TCP friendliness. Thus, high speed protocols can be classified into (*i*) UDP-based protocols (*ii*) protocols that only increase the aggressiveness of TCP for getting a good performance in LDHP networks, (*iii*) protocols with DBECD mechanisms and (*iv*) protocols with available bandwidth probing strategies. Examples of protocols belonging to each family will be listed in the next subsection. However, before discussing high speed protocols, we will describe the new challenges facing such protocols.

2.4.1 New challenges of Protocols for LDHP Networks

Under the characteristics of the LDHP networks, new challenges for congestion control protocols appeared. Therefore, before accepting a high speed protocol as a viable new option, many criteria have to be considered [41]. Below are some points that we consider essential when evaluating high speed protocols.

1. Performance: the proposed protocol has to get a good performance in large BDP networks. For example, acquire 10Gbps with smaller or bigger RTT, whatever the network condition (e.g. Lossy Links, bandwidth fluctuations, etc.).
2. Congestion Control: the proposed protocol has to be able to avoid and control congestion problems.
3. TCP friendliness: in low speed networks, the proposed protocol has to be fair with TCP.
4. Intra Protocol Fairness: the senders using the proposed protocol have to fairly share the available resources.
5. Inter Protocol Fairness: the proposed protocol has to share the resources with others congestion control protocols fairly.
6. Deployment considerations: the mechanisms of the proposed protocol must be deployable by using current technologies.
7. Compatibility with current equipments: the performance of the proposed protocol should not be degraded by the use of current network elements.

2.4.2 UDP-based protocols

2.4.2.1 UDT

UDP-based Data Transfer (UDT) [42], proposed by Yunhong Gu, is a protocol which employs UDP-mechanisms for sending data and control information. UDT is considered as the successor of the Simple Available Bandwidth Utilization Library (SABUL) protocol [43].

Similar to TCP-based protocols, UDT provides reliability by mean of ACKs. However, in UDT, the ACKs are not sent for every n received packets (where $n > 0$). Instead, UDT uses a timer-based selective ACK, which generates an ACK at a fixed interval. Thus, UDT avoids flooding the network with ACKs in bulk-data transfer scenarios. However, when the receiver detects losses of data packets, UDT sends immediately a Negative ACK (NACK) in order for the sender to react as soon as possible to the congestion problem.

On the other hand, to provide congestion control, UDT uses a window control and a rate control both strongly influenced by a rate control interval (SYN[2]), which is equal to the ACK interval.

The window control, also known as flow control, implements a congestion window (W) which determines the number of packets that the sender can send before waiting for an ACK. Since the UDT sends ACKs at every ACK interval, W is calculated by the receiver as follows :

$$W = AS * (SYN + RTT) \tag{2.7}$$

where AS represents the packet arrival speed seen by the receiver.

The rate control estimates the packet sending period (P). P is updated by mean of a modified AIMD mechanism. When a NACK is received, P is increased as follows :

$$P = P' * 1.125 \tag{2.8}$$

where P' is the current sending packet period.

At every SYN time, if no packet losses are detected during the past SYN, P must decrease. In order to correctly decrease P, UDT computes the available bandwidth B every N packets ($N = 16$ is used in the experiences made by the authors of UDT). After N packets, the sender sends two consecutive packets without inter-packet delay to form a Receiver based Packet Pair (RBPP) [44] and estimates the link capacity. If L is the link capacity estimated by RBPP, C the current sending speed and d the decrease factor ($1/9$ on base of Equation 2.8), then B is calculated as :

$$B = minL * d, L - C \tag{2.9}$$

Once the estimated available bandwidth is obtained, the number of packets to be increased in the next SYN (inc) is calculated as follows :

$$inc = max(10^{\lceil log_{10} B \rceil - 9}, 1/1500) * 1500/MSS \tag{2.10}$$

and P is updated as follows :

$$P = \frac{SYN}{SYN/P' + inc} \tag{2.11}$$

To get a more detailed description of UDT, please refer to [42].

2.4.2.2 RUNAT

Reliable UDP-based Network Adaptive Transport (RUNAT) [45], proposed by Wu and Rao, employs UDP-mechanism for data and control information. RUNAT uses a floating window-based transport model and its congestion control is based on both Stochastic Approximation and ADaptive Increase ADaptive Decrease (ADIADD) mechanisms.

The window-based transport model of RUNAT works as follows: RUNAT keeps a congestion window W with a constant value ($W > 1$). Thus, the rate of a sender does not depend on the W value. The congestion window only determines the number of datagrams that a sender can send in a burst. After a burst, the sender stops sending datagrams. This time of

[2]SYN is also a synchronization packet used by TCP to initialize the connection. It sets the packet sequence number to a random value x.

inactivity is called the window-delay, T_{WD}, also referred to as *idle time* or *sleep time*. The value of T_{WD} is modified by the congestion control of RUNAT to increase/decrease the rate of a sender.

In order to update T_{WD} according to the network conditions, RUNAT identifies three levels of congestion or zones that a connexion can experience. The connexion is in the Zone I when it suffers from a loss rate $\hat{l}$ smaller than l_{low} ($l_{low} = 0.001$), which indicates an underutilization of the links. In this case, T_{WD} is updated as follows:

$$T_{WD}(t_{n+1}) = T_{WD}(t_n) + c_r \cdot T_{WD}(t_n)(\hat{l}(t_{n+1}) - l_{low}) \tag{2.12}$$

where c_r is the increase gain coefficient.

The connexion is in the Zone II when $\hat{l}$ is larger than l_{high} ($l_{high} = 0.05$). In this zone, the goodput reaches a peak value and $\hat{l}$ rises. In order to maximize the goodput, the Dynamic Kiefer-Wolfowitz Stochastic Approximation (DKWSA) method [46] is used to tune the T_{WD} value (see Equation 2.13).

$$T_{WD}(t_{n+1}) = \frac{W \cdot MDS}{r(t_{n+1})} \tag{2.13}$$

where the new rate, $r(t_{n+1})$, is computed as :

$$r(t_{n+1}) = r(t_n) + a_n \cdot \hat{G}_g \tag{2.14}$$

where a_n is a real value coefficient and $\hat{G}_g$ is an estimate of the goodput gradient, computed as :

$$\hat{G}_g = \frac{\hat{g}(t_{n+1}) - \hat{g}(t_n)}{\hat{r}(t_n) - \hat{r}(t_{n-1})} \tag{2.15}$$

In Equation 2.15, $\hat{g}(t_{n+1})$ and $\hat{g}(t_n)$ are noisy goodput observations in response to perturbed source rates $\hat{r}(t_n)$ and $\hat{r}(t_{n-1})$ respectively.

Finally, RUNAT detects and retransmits lost packet as described in Algorithm 2. In Algorithm 2, the outstanding time refers to the elapsed time since the datagram has been sent, and the RTO is calculated as show in Equation 2.3.

Data: *ackedDgSeqNo*: sequence number of the acknowledged datagram
Data: *snd_una*: pointer to the oldest standing datagram
Receive ACK and extract *ackedDgSeqNo*;
Remove the datagram of *ackedDgSeqNo* out of the sender buffer;
if ackedDgSeqNo == snd_una **then**
    advance *snd_una* until the first outstanding datagram is found;
else
    if ackedDgSeqNo > snd_una **then**
        for *each outstanding datagram odg from* snd_una *to (*ackedDgSeqNo*-1)* **do**
            if *odg is not in W && outstanding time of odg* > RTO_{odg} **then**
                // Packet loss detected
                load *odg* into W;
            end
        end
    end
end

Algorithm 2: RUNAT algorithm for detection and retransmission of lost packets.

Note that RUNAT requires that all the outstanding datagrams before the out-of-order ACK be examined and reloaded into W if they are not already in and their associated RTO expired.

For a more detailed description of RUNAT, please refer to [45].

2.4.2.3 Limitations of UDP-based protocols

UDP-based protocols have been proposed to provide high performance in BDP networks, especially in dedicated channels, since the aggressiveness of this kind of protocols can degrade the performance of TCP-like protocols, as we said in Subsection 2.2.1. However, there are some UDP-based protocols, like UDT and RUNAT, which incorporate mechanisms to provide friendliness between TCP and these protocols. Such TCP-friendly UDP-based protocols have then been designed to be implemented also in shared heterogeneous LDHP networks.

Moreover, some experiments show that UDT strongly degrades the throughput of standard TCP competing flows, as well as the performance of some high speed TCP versions, like Binary Increase TCP (BIC), as reported in [47]. Thus, it is not yet possible to consider the utilization of UDT in current heterogeneous BDP networks.

Concerning RUNAT, the experiments made by its authors show that RUNAT remains TCP friendly in low speed networks. However, some experiments concerning the friendliness between TCP and RUNAT in BDP networks are still missing. The intra-protocol fairness of RUNAT has not been studied yet. Thus, additional studies are needed to understand the behavior of RUNAT under different conditions.

Finally, it is important to remark that both UDT and RUNAT provide congestion control by mean of an inter-window-delay mechanism. This way, the sender increase (decrease) its sending rate by decreasing (increasing) the sending burst frequency, in the case of RUNAT, or the sending packet frequency, in the case of UDT. However, inter-window-delay mechanisms require high accuracy computations, which cannot always be provided in computers realizing multiples parallel tasks.

2.4.3 Protocols only increasing the aggressiveness of TCP

Some proposed high speed protocols only modify the response function of standard TCP in order to get a good performance in large BDP networks. In this section we will list and explain the protocols belonging to this subfamily that we consider as the most representatives.

2.4.3.1 High Speed TCP

HSTCP is a protocol proposed by Sally Floyd in 2003 [40]. HSTCP implements the Limited Slow Start mechanism for TCP with large congestion window [48].

As explained in the subsection 2.3.1, during Slow Start the congestion window doubles at every RTT. In LDHP networks, this behavior could produce serious problems. Let us imagine for instance a connection with a *cwnd* equal to 2000 MSS. After a RTT, the value of *cwnd* will be 4000 MSS. If the network capacity can only support a *cwnd* of 2500 MSS, then the number of dropped packets will be too high (approximately 1500 packets).

Limited Slow-Start solves the problem described earlier. Limited Slow-Start introduces a parameter, $max_ssthresh$, and modifies the slow-start mechanism for a *cwnd* larger than $max_ssthresh$. Limited Slow Start is executed when $max_ssthresh < cwnd \leqslant ssthresh$.

When $cwnd < max_ssthresh$, Slow Start is executed as always. Below, we present the algorithm for Normal and Limited Slow Start (Algorithm 3):

For each arriving ACK in slow-start:
if $cwnd <= max_ssthresh$ **then**
  $cwnd += MSS$;
else
  $K = int(cwnd/(0.5max_ssthresh))$;
  $cwnd += int(MSS/K)$;
end

Algorithm 3: Limited Slow-Start Algorithm.

With Limited Slow-Start, when the congestion window is greater than $max_ssthresh$, the window is increased by at most 1/2 MSS for each arriving ACK; when the congestion window is greater than 1.5 $max_ssthresh$, the window is increased by at most 1/3 MSS for each arriving ACK, and so on.

In [48], the authors recommend a value of 100 MSS for $max_ssthresh$. Limited Slow Start can also be used with standard TCP.

When a HSTCP flow is in the Congestion Avoidance phase, if the sender does not detect any loss, then the congestion window is increased as follow:

$$cwnd = cwnd + a(cwnd)/cwnd \tag{2.16}$$

where $a(cwnd)$ is a value that depends on the current congestion window size. The Figure 2.7 shows graphically the values for $a(cwnd)$ for every value of $cwnd$. From Figure 2.7 we can see that $a(cwnd)$ increases proportionally to the congestion window size.

In case of losses, HSTCP decreases the congestion window as follows:

$$cwnd = cwnd - b(cwnd) * cwnd \tag{2.17}$$

where $b(cwnd)$ is a value that depends on the current congestion window size. The figure 2.7 shows graphically the values for $b(cwnd)$ for every value of $cwnd$. From Figure 2.7 we can see that $b(cwnd)$ is inversely proportional to the congestion window size.

When the congestion window size of a HSTCP sender is inferior to 38 packets, $a(cwnd) = 1$ and $b(cwnd) = 1/2$ becoming fair with standard TCP.

2.4.3.2 Scalable TCP

Scalable TCP (STCP) [1], proposed by Tom Kelly, is a protocol very similar to HSTCP. The main difference is based on the fact that the amount to increase the congestion window size, for every received ACK, is always constant.

The equation 2.18 represents the way to update $cwnd$ during congestion Avoidance at every ACK received, in absence of losses.

$$cwnd = cwnd + 0.01 \tag{2.18}$$

The equation 2.19 represents the way to update $cwnd$ in case of losses.

$$cwnd = cwnd - 0.125 * cwnd \tag{2.19}$$

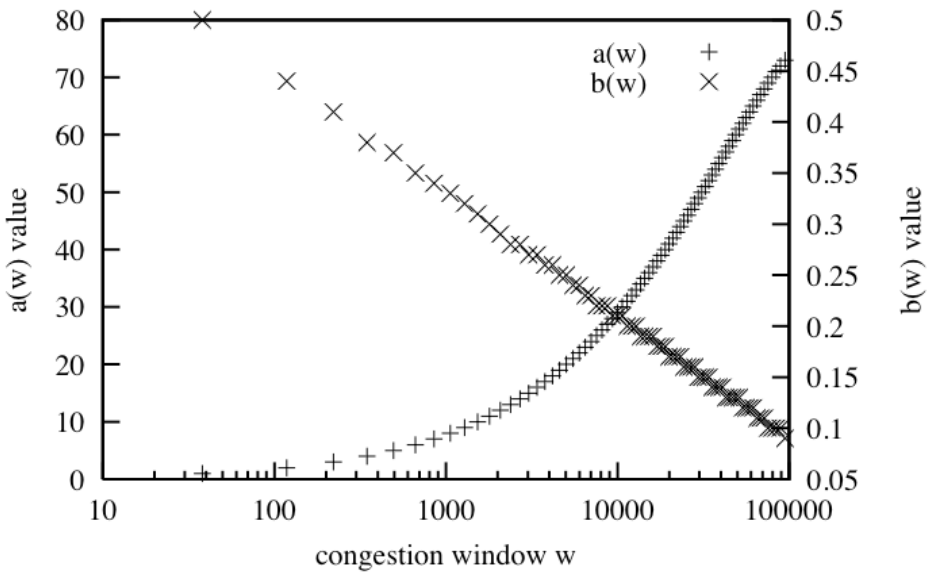


Figure 2.7: $a(cwnd)$ and $b(cwnd)$ values for different congestion window size.

STCP provides elastic speed convergence. It means that after a loss event, a STCP flow which has reduced its rate from X to Y Mbps ($X > Y$) will always need the same period of time to increase their rate from Y to X, independently of the value of X, if the RTT does not change. Standard TCP does not provide elastic speed convergence. A TCP flow will need different periods of time to reach X after losses, even if the RTT remains unchanged. The time needed by TCP will depend on the X value. Figure 2.8 from [1] illustrates the difference in the convergence speed between TCP and STCP.

2.4.3.3 Binary Increase TCP

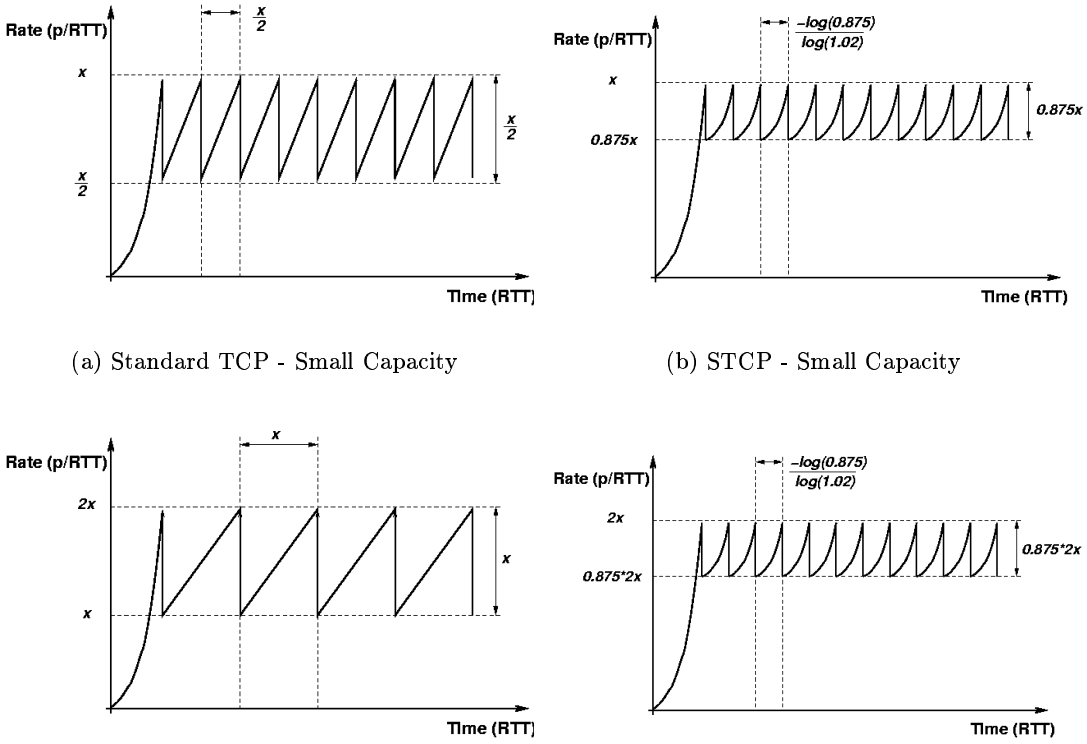
BIC [2] was proposed by Lisong Xu. Since BIC is used by default in current Linux kernel versions [49], the number of end-host using BIC is increasing.

BIC modifies strongly the behavior of the TCP Congestion Avoidance algorithm. First, BIC adds the next parameters in its congestion avoidance algorithm:

- *low_window*: if *cwnd* is larger than this threshold, BIC is activated.
- S_{max}: the maximum increment of *cwnd* by RTT.
- S_{min}: the minimum increment of *cwnd* by RTT.
- β: multiplicative window decrease factor (in case of losses).
- *default_max_win*: default maximum *cwnd* value (a large integer).

And the next variables:

- *max_win*: the maximum window size. Initially the default maximum.



(a) Standard TCP - Small Capacity

(b) STCP - Small Capacity

(c) Standard TCP - Large Capacity

(d) STCP - Large Capacity

Figure 2.8: Scaling properties of Standard TCP and STCP (from [1]).

- *min_win*: the minimum window size.
- *prev_max*: the maximum window size just before the current maximum is set.
- *target_win*: the midpoint between maximum and minimum.
- *cwnd*: the congestion window size.
- *is_BITCP_ss*: boolean indicating whether the protocol is in the BIC slow start algorithm.
- *ss_cwnd*: a variable to keep track of *cwnd* increase during the BIC slow start.
- *ss_target*: the value of *cwnd* after one RTT in BIC slow start.

In a general way, BIC [2] works as follows:

At the startup, and while *cwnd* is smaller than *low_window*, BIC executes the same Slow Start TCP algorithm (*cwnd* is doubled at every RTT). After *cwnd* becomes larger than *low_window*, BIC executes its own Slow Start algorithm. At this time *cwnd* will only increase as maximum S_{max} packets every RTT.

When the flow suffers its first dropped packet (and later at every dropped packet), *max_win* is set to the current *cwnd* value, *cwnd* is decreased by a factor β and *min_win* is set to the new *cwnd* value. Finally *target_win* is set to the middle between *max_win* and *min_win*. The next RTT, *cwnd* should increase by N packets in order to reach the *target_win* value. However, if N is larger than S_{max}, *cwnd* only will increase the maximum allowed increment S_{max}. The Figure 2.9 shows a sample run of two BI-TCP flows.

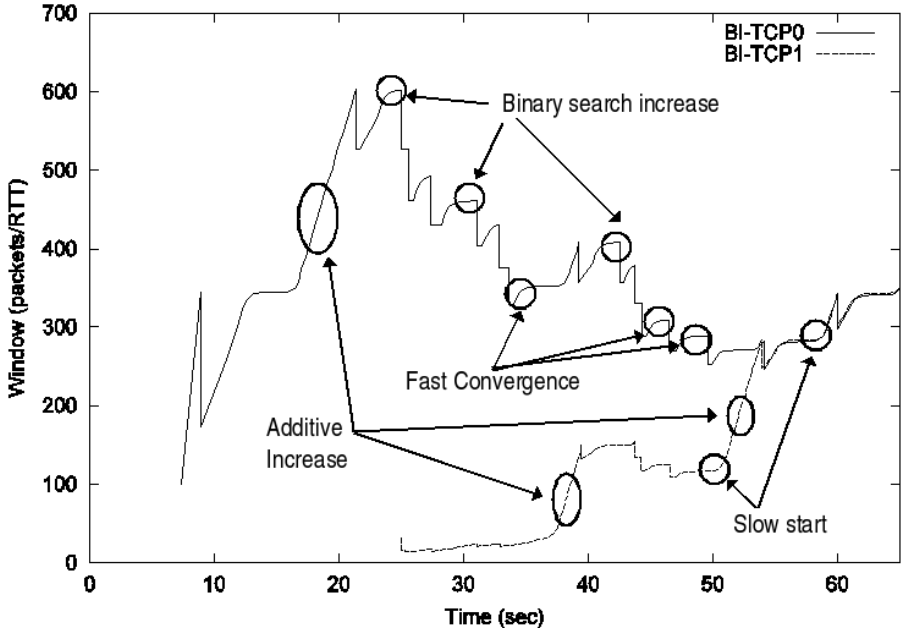


Figure 2.9: 2 BIC flows (from [2]).

However, two important drawbacks have been discovered: (*i*) BIC can be very aggressive for TCP in low speed networks and (*ii*) the intra protocol fairness of BIC can be low. CUBIC was then proposed to solve these problems [50].

CUBIC improves the functionality of BIC via a function, which simplifies window control and enhances TCP friendliness, while keeping the major strengths of BIC (particularly stability and scalability). As the name suggests, CUBIC has a new cubic growth function. In CUBIC, after a window reduction, *cwnd* grows quickly and when *cwnd* gets closer to W_{max} (a dynamically calculated maximum window) this growth slows down and the increment becomes zero at W_{max}. Then, *cwnd* grows in a reverse fashion, which means that it grows slowly at first and when it moves away from W_{max}, it grows quickly. The slow growth of *cwnd* around W_{max} enhances its stability and the utilization of network, and fast growth away from W_{max} makes it more scalable.

2.4.3.4 Limitations of Protocols only increasing the aggressiveness of TCP

The clear advantage of using HSTCP or STCP is that we can grab link with high capacity and large delays.

However, both HSTCP and STCP are too aggressive, and this aggressiveness produce unfairness between flows with different RTT. Simple analysis based on synchronized loss model [2], shows that for any loss-based congestion control protocol with a response function of c/p^d where p is the packet loss rate, and c and d are constant, RTT unfairness is proportional to

$$\left(\frac{R_1}{R_2}\right)^{\frac{1}{1-d}} \tag{2.20}$$

These d values for HSTCP and STCP are 0.82 and 1, respectively. Thus, the RTT unfairnesses of HSTCP and STCP are 5.55 and infinite, respectively.

Concerning CUBIC (we will not discuss about BIC since CUBIC is the improved version of BIC), in [51], the authors show that the problems of fairness persist in CUBIC. According to the authors, the unfairness problems are generated by a slow convergence to the fairness lines, between flows with different RTTs. This time of convergence is very dependent on the start time of the flows and the reasons are not yet clear.

2.4.4 Protocols with DBECD mechanisms

Only increasing the aggressiveness of TCP creates many congestion problems in the network. Thus, some congestion control protocols for LDHP networks implement new mechanisms to detect congestion before it occurs. Generally, these mechanisms are based on the idea that an increment in the delay of the path is synonym of congestion. That is why such strategies are called DBECD mechanisms. Below, we describe some of the most representatives protocols using DBECD mechanisms.

2.4.4.1 FAST TCP

FAST TCP was born with the improvement of the stability of TCP Vegas in large BDP networks [52]. FAST TCP, presented in [53] by Steven Low, is considered the "High Speed version" of TCP Vegas.

FAST TCP reacts to both queueing delay and packet loss. Under normal network conditions, FAST TCP periodically (20ms in the designers' prototype) updates the congestion window, based on the average RTT and average queueing delay, by using the following equation:

$$cwnd = min\left\{2 * cwnd, (1-\gamma) * cwnd + \gamma\left(\frac{baseRTT}{RTT}cwnd + \alpha\right)\right\} \tag{2.21}$$

where $\gamma \in (0, 1]$, $baseRTT$ is the minimum RTT observed so far, and α is a positive protocol parameter that determines the total number of packets queued in routers along the flow path.

2.4.4.2 Compound TCP

Compound TCP (CTCP) [54] is a high speed protocol proposed by Kun Tan, from the Microsoft Research Asia team, and is currently used in some Microsoft products, like Windows Vista [55].

CTCP combines a loss-based and a delay-based component. For this reason, CTCP keeps two window state variables: $cwnd$ (the congestion window), which controls the loss-based component, and $dwnd$ (the delay window), which controls the delay-based component of CTCP. Thus, the CTCP sending window is controlled by both $cwnd$ and $dwnd$, according to the following equation:

$$win = min(cwnd + dwnd, awnd) \tag{2.22}$$

where $awnd$ is the advertised window from the receiver. $cwnd$ is halved upon a packet loss event, $cwnd = cwnd/2$, as always, but the increase is slightly different: upon an ACK reception $cwnd = cwnd + 1/win$.

The way to update $dwnd$ is more complex. First, it is necessary to compute the difference, $Diff$, between the expected ($Expected$) and the actual ($Actual$) sending rate, as in TCP Vegas [30]:

$$Expected = win/baseRTT \tag{2.23}$$

$$Actual = win/RTT \tag{2.24}$$

$$Diff = (Expected - Actual)/baseRTT \tag{2.25}$$

In Equations 2.23 and 2.25, $baseRTT$ is the lower computed RTT. After computing $Diff$, $dwnd$ can be updated as follow:

$$dwnd(t+1) = \begin{cases} dwnd(t) + (\alpha.win(t)^k - 1) & Diff < \gamma \\ dwnd(t) - \zeta.Diff & Diff \geq \gamma \\ win(t).(1-\beta) - cwnd/2 & \text{if loss is detected} \end{cases}$$

where $\alpha = 1/8$, $k = 0.75$, $\zeta = 1$ and $\beta = 1/2$. γ could be a fixed value (30 in the experiments made by the authors), or a dynamic value given by the next equation:

$$\gamma = max\left(\gamma_{min}, W_{low}.\frac{\kappa}{1+\kappa}\right) \tag{2.26}$$

where the value of W_{low} is not given by the authors, and

$$\kappa = \frac{RTT_{max} - RTT_{min}}{RTT_{min}} \tag{2.27}$$

2.4.4.3 Others protocols

There are others protocols with DBECD, like TCP-Illinois [56] or Hamilton TCP [57]. However, exploring all those protocols would be beyond the scope of this thesis. In addition, we believe FAST TCP and CTCP represent very well the group of Congestion Control protocols with DBECD. These last two protocols can clearly show the advantages and disadvantages of this kind of protocols.

2.4.4.4 Limitations of Protocols with DBECD mechanisms

Currently, many propositions add DBECD strategies, in order to improve TCP in LDHP networks, and they have begun to be implemented in some Operating Systems (OSs), like CTCP in new Windows releases. DBECD has been taken as the solution to the TCP problems to avoid congestion and losses of packets.

The first question is therefore: Are RTT variations actually caused by congestion in the network? In [58] the authors try to answer this question mathematically and by mean of simulations. The answer in both cases is that the probability that the RTT variations are caused by congestion is too weak. In addition, they proved that the RTT measured by TCP is imprecise and bears a high random component independent of the actions of the sender, resulting in a sending rate not adapted to the network conditions.

Using simulations, in [59] the authors show that RTT variations are caused mainly by the bursty nature of aggregate TCP traffic arriving at high-speed switches and the coarseness of a TCP constrained RTT probe mechanism. They have shown also that it is rare that DBECD strategies detect the queue buildup that precedes packet loss and react in time to avoid the loss (only 7% – 18% of the time on average). The reason is that protocols using DBECD strategies need one RTT to react, and some others to get the good congestion window size.

In addition, in [60] the authors identify and explain clearly some open issues regarding DBECD algorithms, that have not yet been addressed or resolved in some manner. The conclusion is that it would be premature to consider replacing the existing TCP congestion control with much less understood DBECD schemes.

2.4.5 Protocols with Available Bandwidth Estimation strategies (Westwood TCP)

One other alternative used by congestion control protocols to increase the performance in LDHP networks is to detect the available bandwidth between end hosts, as in the TCP Westwood protocol [61].

TCP Westwood is a high speed protocol proposed by Saverio Mascolo. TCP Westwood is based on TCP. However, it adds a new important feature to standard TCP: a more intelligent way to update the slow start threshold (*ssthresh*).

In order to set a more accurate *ssthresh* value, Westwood implements 2 new algorithms to standard TCP:

1. Eligible Rate Estimated (ERE). ERE is basically the estimation of the available bandwidth between the sender and the receiver. The ERE is computed when 3 DUPACKs are received and it is used to update the *ssthresh* value before executing Congestion Avoidance.

2. Agile Probing. Agile Probing is a new algorithm added to the well known Slow Start algorithm. It consists in calculating ERE and updating the *ssthresh* variable while Slow Start is executed.

After the birth of TCP Westwood, a problem of available bandwidth estimation in the presence of reverse traffic due to ACK compression was soon discovered. TCP Westwood+ [62] fixed this problem.

Later, it was discovered that when Westwood+ suffers some losses and Congestion Avoidance is executed, the time to get the available resources over LDHP networks could be really long (similar to standard TCP). To solve this problem, TCP Westwood+ implemented a like-STCP Congestion Avoidance algorithm [63]. TCP Westwood+ then inherits the same problems as STCP (intra and inter protocol unfairness) described earlier.

2.5 Buffer management and notification for mitigating congestion

All the protocols for providing congestion control and fair share of resources shown so far are E2E protocols, since they are only implemented in the end host, and do not require the collaboration of others network elements. However, such protocols have proved unable to satisfy the requirements in performance and fairness, as we have already seen.

One of the weakness of E2E approaches is based on the fact that they have imperatively to predict the state of the networks, since they are unable to communicate with forwarding network devices, where the congestion occurs. For this reason, some existing approaches specially conceived for routers, implement queue management to avoid congestion with/without notification to the end hosts, in order to control the congestion and the fairness more accurately. Such approaches aimed at replacing current DropTail routers (widely deployed in IP networks), that cannot provide any information about congestion level to the end hosts or to make any decision for avoiding congestion.

We will describe in the next subsections the most important existing approaches for routers. We will classify them into three large groups: (*i*) approaches focused only on queue management, without providing any information to the end hosts, (*ii*) approaches focused on queue management, providing binary signal to end hosts (congestion or non-congestion signals) and (*iii*) much more complex approaches able to provide the most adapted sending rate to the end hosts.

However, before listing the congestion control mechanisms embedded in routers, and only as an additional information, we want to remark that in IP networks, in the past, there were already some attempts to use the forwarding network node in collaboration with end hosts in order to improve the performance of applications. For instance, in Active Networks in 1996 [64], the routers were used like active nodes, able to process data packets before forwarding them, if the senders required such treatments. Some examples of projects in Active Networks are ANTS [65] (Active Node Transfer System), Tamanoir [66] and FAIN (Future Active IP Networks) [67].

2.5.1 Buffer management solutions without end hosts interactions

Buffer management solution without end hosts interaction are also known as AQM strategies. They are based on the hypothesis that when the occupancy buffer exceeds a certain threshold, a congestion could occur at any time. Therefore, AQM strategies execute some mechanisms in order to avoid congestion when this threshold is exceeded. Since AQM mechanisms are independent of the senders, they can be used without considering which congestion control protocol is used in end hosts. Below, we present the strategies that we consider as the more important ones.

2.5.1.1 Random Early Detection

RED, proposed in [68], calculates the average queue size of the router, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a minimum threshold and a maximum threshold. When the average queue size is less than the minimum threshold, no packets are discarded. When the average queue size is greater than the maximum threshold, every arriving packet is discarded.

When the average queue size is between the minimum and the maximum threshold, each arriving packet is discarded with probability p_a, where p_a is a function of the average queue size avg (it means that the probability to drop a packet increases as the buffer occupancy increases). This way, the probability that a packet is discarded from a particular flow is roughly proportional to the available bandwidth used by this flow. The algorithm of RED is presented in Algorithm 4.

```
for each packet arrival do
    calculate the average queue size avg;
    if min_th < avg < max_th then
        calculate probability p_a;
        with probability p_a drop the arriving packet;
    else
        if max_th < avg then
            drop the arriving packet;
        end
    end
end
```

Algorithm 4: RED Algorithm

2.5.1.2 CHOKe

CHOKe [69] is an AQM mechanism that improves RED in the sense that CHOKe penalizes the more aggressive flows. Since RED penalizes indiscriminately both aggressive and non-aggressive flows during congestion, non-aggressive flows could have performance excessively degraded.

In a general way, CHOKe works as RED: if the average queue size of a router is less than a certain threshold min_{th}, every arriving packet is queued into the buffer. If the average queue size is greater than a maximum threshold max_{th}, every arriving packet is dropped. This moves the queue occupancy back to below max_{th}. When the average queue size is bigger than min_{th} and smaller than max_{th}, each arriving packet is compared with a randomly selected packet,

called drop candidate packet, from the buffer. If they have the same flow ID, they are both dropped. Otherwise, the randomly chosen packet is kept in the buffer (in the same position as before) and the arriving packet is dropped with a probability that depends on the average queue size. The drop probability is computed exactly as in RED. In particular, this means that packets are dropped with probability 1 if they arrive when the average queue size exceeds max_{th}. The CHOKe algorithm is given in Algorithm 5.

```
for each packet arrival do
    calculate the average queue size avg;
    if min_th < avg < max_th then
        draw a packet at random from queue;
        if both packets belongs to the same flow then
            drop both packets;
        else
            calculate probability p;
            with probability p drop the arriving packet;
        end
    else
        if max_th < avg then
            drop the arriving packet;
        end
    end
end
```

Algorithm 5: CHOKe Algorithm

2.5.1.3 BLUE

BLUE [70] is another AQM algorithm that aims at improving RED. When a large number of TCP sources are active, the aggregate traffic generated is extremely bursty. Bursty traffic often defeats the AQM techniques used by RED since queue lengths grow and shrink rapidly, well before RED can react.

To remedy the shortcomings of RED, BLUE performs queue management based directly on packet loss and link utilization rather than on the instantaneous or average queue lengths. BLUE maintains a single probability, p_m, which it uses to drop packets when they are enqueued. If the queue is continually dropping packets due to buffer overflow, BLUE increments p_m, thus increasing the rate at which it sends back congestion notification. Conversely, if the queue becomes empty or if the link is idle, BLUE decreases its dropping probability.

Besides the dropping probability, BLUE uses two other parameters which control how quickly the dropping probability changes over time. The first is $freeze_time$. This parameter determines the minimum time interval between two successive updates of p_m. The other parameters used, δ_1 and δ_2, determine the amount by which p_m is incremented when the queue overflows or is decremented when the link is idle. δ_1 is set significantly larger than δ_2. This is because link underutilization can occur when congestion management is either too conservative or too aggressive, but packet loss occurs only when congestion management is too conservative. The Algorithm 6 presents the BLUE algorithm to update p_m.

Data: now: current time.
Upon packet loss event:
if $((now - last_update) < freeze_time)$ **then**
  $p_m = p_m + \delta_1$;
  $last_update = now$;
end
Upon link idle event:
if $((now - last_update) < freeze_time)$ **then**
  $p_m = p_m + \delta_2$;
  $last_update = now$;
end

Algorithm 6: BLUE Algorithm

2.5.1.4 Stabilized RED

Stabilized RED (SRED), proposed in [71], improves RED, by adding a new feature: an estimation of active flows, without keeping per-flow states in the routers.

SRED has two main modules that, once combined, avoid the congestion and improve the fair share of the network resources. These modules are (i) estimation of the active flows and (ii) a RED-like mechanism.

The algorithm for estimating the active flows in a network works as follows:

SRED keeps a table called the *zombie table* able to keep up to 1000 ID flows (the ID flow could be src_addr:src_port::dest_addr:dest_port). The zombie table is filled at the beginning with the ID flows belonging to the first 1000 incoming packets.

When the zombie table has been filled in, every arriving packet to the router is examined. First, the ID flow of the packet is taken and compared with a ID flow taken randomly from the zombie table. If the ID are the same, then an event *hit* is declared. In the other case, an event *mis* is declared. When a *mis* is detected, with a probability of 25%, the old stored ID in the zombie table will be replaced by the new one. After a *hit* or *mis*, a equation that computes the probability to make a *hit* is updated (See equation 2.28).

$$P(t) = (1 - \alpha)P(t-1) + \alpha.Hit(t) \tag{2.28}$$

Where α = probability to replace old ID/zombie table size ($\alpha = 0.00025$) and

$$Hit(t) = \begin{cases} 0 & \text{if no hit} \\ 1 & \text{if hit} \end{cases}$$

Now, if the total of active flows in a router is N, then the probability to get a *hit* is $1/N$, which is calculated in $P(t)$. Therefore, $1/P(t)$ represent the estimation of the number of active flows seen by a router at time t.

The second part of SRED, the RED-like mechanism to avoid congestion and improve the fair share between flows, works as follows:

SRED drops packets with a probability p_{zap} computed as

$$p_{zap} = p_{sred}.min\left(1, \frac{1}{(256.P(t))^2}\right)$$

Where p_{sred} is a value that depends on the amount used, q, of the total buffer capacity, B, in a router, at a given time t and a predefined constant $p_{max} = 0.15$:

$$p_{sred} = \begin{cases} p_{max} & \text{if } \frac{1}{3}B \leq q \leq B \\ \frac{1}{4}p_{max} & \text{if } \frac{1}{6}B \leq q < \frac{1}{3}B \\ 0 & \text{if } 0 \leq q < \frac{1}{6}B \end{cases}$$

Finally, note that SRED can discard packet in both cases, i.e. *hit* or *mis*.

2.5.1.5 Limitations of AQM mechanisms

In order to prove the effectiveness of AQM mechanisms and the impact of these algorithms in networks, some studies dedicated to the comparison and analysis of AQM mechanisms have been made ([72, 73, 74] between others).

Such studies show that each AQM mechanism offers different advantages and disadvantages, depending on the network conditions. Hence, it is very difficult to declare a "winner" and support the deployment of one of many proposed AQM mechanisms. For instance, in some scenarios, the AQM mechanisms presented before could improve the fairness, while in others it could degrade it seriously. In addition, in [73], we can see that both the variable capacity of queue and variable input load negatively impact the performance of AQM schemes. AQM algorithms prove unable to keep the queue length and drop rate low and stable simultaneously in dynamic networks.

2.5.2 Solutions providing ECN to end hosts

2.5.2.1 Explicit Congestion Notification

In Frame Relay networks, in order to provide congestion control and notification from forwarding network devices to the senders, the use of the Forward Explicit Congestion Notification (FECN) bit, the Backward Explicit Congestion Notification (BECN) bit and the Discard Eligibility (DE) bit were proposed. FECN is turned on when a congestion is present in the forwarding path and BECN is turned on when a congestion is present in the reverse path. Thus, when a host receives packets with the FECN or BECN bit turned on, the host must decrease its rate. On the other hand, the DE bit is turned on inside a frame, when a sender decides that this frame could be discarded in case of congestion. The idea behind FECN for Frame Relay networks was later reused in IP networks, in order to create the ECN protocol.

ECN, described in [8], was proposed to complement AQM mechanisms. However ECN is not an AQM mechanism in itself. ECN should be used in combination with (*i*) any AQM algorithm, like RED, SRED, BLUE, etc, and (*ii*) any of the congestion control protocols listed earlier. ECN breaks the paradigm of E2E protocols, since it implements a mechanism of communication between the end hosts and layer 3 devices to avoid congestion. Evidently, both end hosts and routers must be ECN-capable.

Routers implementing any AQM mechanism and ECN, provide a signal indicating a congestion in the network to the end users, instead of dropping its packets. For instance, a RED router that exceeds its minimum threshold could drop a packet to prevent congestion. However, if this router implements ECN (e.g. RED-ECN), it will not drop the packet, but will only mark the packet of ECN-capable senders (e.g. TCP New Reno-ECN) in order to indicate that a congestion could arrive and that it is necessary to decrease their sending rates.

More specifically, ECN works as follows:

Two bits are used by ECN in order to coordinate the actions between routers and ECN-capable senders. When a sender does not support ECN both bits are turned off. When the sender supports ECN, one of the bits is turned on to notify to the ECN routers that it is an ECN-capable sender. This way, when a ECN router detects incipient congestion (detected by mean of a certain buffer occupancy level) and if the sender supports ECN, the router will turn both bits on. The congestion notification inserted in the data packets will be sent back to the sender by mean of the ACK packets. Finally, when a sender receives a congestion notification from an ECN router, the sender should reduce its rate to avoid the congestion and losses of packets.

2.5.2.2 Variable-structure congestion Control Protocol

Variable-structure congestion Control Protocol (VCP) [75] is an improved version of ECN. However, VCP is not based on any of the currently known AQM mechanism. Instead, VCP routers monitor the input traffic rate and compare it with the output link capacity to provide information to the senders about the state of the network. This new protocol classifies the input traffic rate into :

1. Low: VCP concludes that the input traffic rate is low when it does not exceed 80% of the output link capacity. When the input traffic rate is low, VCP indicates to the VCP-capable senders to use a Multiplicative Increase algorithm.

2. High: VCP concludes that the input traffic rate is high when it is higher than 80% of the output link capacity, but does not exceed 100% of the link capacity. When the input traffic rate is high, VCP indicates to the VCP-capable senders to use an Additive Increase algorithm.

3. Over: When the input traffic rate exceeds 100% of the output link capacity, VCP indicates to the VCP-capable senders to apply a Multiplicative Decrease algorithm.

2.5.2.3 Limitations of ECN solutions

The ECN protocol proved unable to solve the problems in congestion and fairness satisfactorily. The problems of ECN are the following:

- since ECN punishes only ECN-capable senders, it could degrade strongly the performance of these senders, while non-ECN senders will take the resources, resulting in a high level of unfairness.

- ECN inherits the problems of AQM mechanisms since it is implemented on top of these algorithms.

- ECN only provides a binary signal (congestion or non-congestion) to end users, that is equivalent to the binary signals used by congestion control protocols, and that is the origin of the instability in the throughput of the senders, which oscillates always between the *knee* and the *cliff* (Figure 2.2) (i.e. between the point of non-congestion and congestion).

On the other hand, VCP inherits one of the weakness of ECN: since VCP punishes only VCP-capable senders, it could degrade the performance of these senders, while non-VCP

senders will take the resources, resulting in a high level of unfairness. In addition, in the simulations of VCP provided by its authors [75], we can see that the convergence time[3], which increases as the rate of the senders increases, can be too large when flows are inserted asynchronously into the network (up to 100s in order to get fairness between a sender with an RTT of 40ms that has already taken 45Mbps, and a new flow with an RTT of 50ms).

2.5.3 Solutions providing ERN to end hosts

New research in congestion control protocols originated new solutions totally different from E2E protocols and AQM mechanisms. While in E2E protocols, the congestion control mechanisms resided in the end hosts, these solutions aimed at migrating all the congestion control and fairness mechanisms from the end hosts to the routers. This way, end hosts, that only follow the indication provided by routers, do not cause congestion problems inside the network, stay stable in the *knee* and get surprising fairness level. These new solutions extend the ECN mechanism by providing ERN to the end hosts. This is why they are known as ERN protocols.

One of the first proposed ERN protocol was the Explicit Rate Indication for Congestion Avoidance (ERICA) protocol [76] (patented in September 1998 by Jain et al.). ERICA is an ERN protocol specially designed to work in forwarding network devices of ATM networks. ERICA computes an optimal rate for the senders, taking into account the output link capacity of the ATM switches and the number of active flows during a control interval. Thus, ERICA could be considered as the predecessor of the ERN protocols for IP networks. However, due to the problems of scalability of ATM networks, ERICA and most of the research concerning ATM mechanisms were stopped.

We can currently find different propositions of ERN protocols for IP networks (such as [3, 77, 4, 78, 79]). However, in this section, we will focus on XCP [3], Quickstart [79] and JetMax [4] since they are the most known and most discussed protocols. We believe also that the mechanisms of other ERN protocols are very similar to the mechanisms inside XCP, JetMax and TCP Quickstart.

2.5.3.1 eXplicit Control Protocol

Description of XCP

XCP [3] is a protocol proposed by Dina Katabi. This protocol is based on the use of XCP routers specially designed to compute the more adapted congestion window size of the sender. XCP has the benefits of scalability since it does not need to keep per flow states in order to correctly share the available resources.

In a general way, XCP works as follows: the XCP sender sends data packets containing a desired rate and some values that will let the XCP routers execute their algorithms for computing the available resources. After computing the available bandwidth, if the XCP router estimates that the desired rate is larger than its available resources, then it will change the desired rate by the allowed maximum sending rate. The next XCP routers will change the last allowed sending rate only if they have less available resources. Finally, the information written by the routers into the data packets will be sent back to the sender in the ACK packets sent by the receiver. Figure 2.10 illustrates the XCP protocol.

[3]The elapsed time before flows get fairness

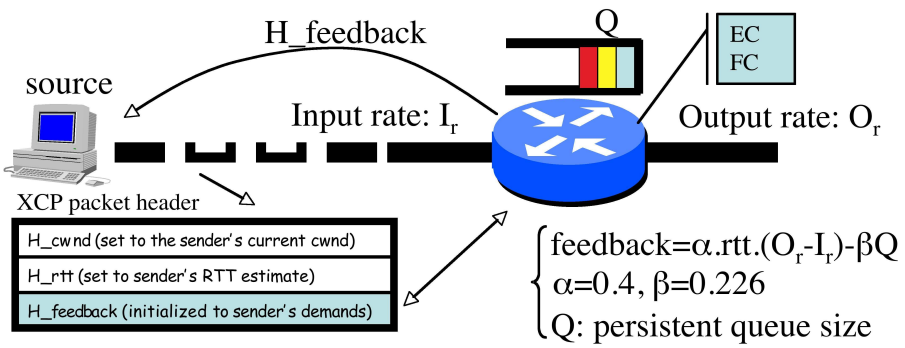


Figure 2.10: The XCP protocol.

To understand how XCP works in detail, we will describe the work accomplished by every XCP component: the packets, the senders, the receivers and the XCP routers.

The XCP packets

The XCP protocol adds three new fields to the TCP header, to get a XCP packet. They are:

1. *H_cwnd*: it indicates the current congestion window size to all routers in the data path.

2. *H_rtt*: it indicates the average RTT computed by the sender to all routers in the data path.

3. *H_feedback*: this field is initialized by the sender. *H_feedback* indicates to the XCP routers the sender desired rate. Later, this field is modified by the routers according to the network conditions.

The information contained in *H_cwnd* and *H_rtt* allows to the XCP routers to execute the appropriate mechanisms to maximize the utilization of the bandwidth between all the active flows and share fairly the resources. *H_feedback* will contain the way to increase (or decrease if this value is negative) the current throughput of the sender.

XCP efficiency relies on the XCP routers: they dictate the sending rate of the senders. We will explain in more details how a XCP router works, but before we will describe the task of the end XCP hosts.

The XCP sender

Since in XCP the routers compute a value that indicates the increase or decrease of the sender congestion window, the sender only has to add this value to the congestion window size at every ACK received:

$$cwnd = cwnd + H_feedback \tag{2.29}$$

The XCP sender does not estimate the network capacity as a sender using an E2E protocol does, since XCP senders only follow the "instructions" given by the XCP routers.

The XCP receiver

When a data packet reaches the XCP receiver node, the packet will arrive with a *H_feedback* field modified by the XCP routers. Therefore, the XCP receiver has to copy the feedback in the respective ACK packet that will be sent back to the sender.

The XCP router

As we said before, the XCP router is the heart of the XCP protocol. In order to compute a value that will indicate the congestion window size to the sender, a XCP router has to execute two main algorithms for every incoming data packet: the Efficiency Controller (EC)) and the Fairness Controller (FC).

Both the EC and the FC, in order to correctly fulfil their goals, need to know the XCP parameters of every incoming data packet: the *H_feedback*, the *H_cwnd* and the *H_rtt*, explained earlier. However, both the EC and the FC need also to be aware about specific characteristics of every incoming packet, contained in the IP header of the packets. For every incoming packet, a XCP router indeed extracts the following informations from the IP header:

- the packet size, that will let the XCP routers compute the input traffic rate, necessary for the feedback computation.
- the packet type, that will let the XCP routers distinguish between ACK packets (whose rate must not be controlled) from data packet (whose rate must be controlled).
- the congestion control protocol used, that will let the XCP routers identify the XCP capable senders from the XCP non-capable senders.

The value obtained at the end of the execution of the EC and the FC is called the *feedback per packet*. If the computed feedback per packet is smaller than the value of the *H_feedback* field from the XCP header, then the value of this field will be replaced by the new feedback per packet. Now, we will see the goals and the mechanisms used in both EC and FC.

- The EC is the mechanism that maximizes the network resources utilization, while avoiding packet drops.
- The FC is the mechanism that fairly assigns the resources between the actives flows.

The EC computes a desired increase or decrease in the aggregate traffic rate. This aggregate feedback, ϕ, is computed at each control interval (approximately equal to the average *H_rtt* of all incoming data packets):

$$\phi = \alpha.rtt.(O - I) - \beta.Q \tag{2.30}$$

where α and β are constant parameters, whose values are set, based on stability analysis, to 0.4 and 0.226, respectively. rtt is the average RTT, O is the output link capacity and I the input traffic rate. Note that $(O-I)$ can be negative. Finally, Q is the persistent queue size (i.e.,

the queue that does not drain in a round trip propagation delay), as opposed to a transient queue that results from the bursty nature of all window-based protocols. Q is computed by taking the minimum queue seen by an arriving packet during the last propagation delay, which is estimated by subtracting the local queuing delay from the average RTT. Equation 2.30 makes the feedback proportional to the available bandwidth because, when $(O - I) \geqslant 0$ the link is underutilized and the feedback to send must be positive, while when $(O - I) < 0$, the link is congested and the feedback to send must be negative.

Finally, ϕ must be proportional to the persistent queue, in order to drain the queue when the input traffic matches the capacity.

EC computes an aggregate feedback. EC does not care which packets get the feedback and by how much each individual flow changes its rate. ϕ must be translated therefore in a per-packet feedback to control reasonably efficiency and fair share of every flow. This work is made by FC.

The FC relies on the same principle TCP uses to converge to fairness, namely AIMD. FC computes the per-packet feedback according to the policy:

If $\phi > 0$, allocate it equally to all flows.

If $\phi < 0$, allocate it to flows proportionally to their current throughputs.

This ensures continuous convergence to fairness as long as the aggregate feedback is not zero. To prevent convergence stalling when efficiency is around optimal ($\phi \approx 0$), the concept of bandwidth shuffling is introduced. This is the simultaneous allocation and deallocation of bandwidth such that the total traffic rate (and consequently the efficiency) does not change, yet the throughput of each individual flow changes gradually to approach the flow's fair share. The shuffled traffic, h, is computed as follows:

$$h = \max(0, \gamma.(I - \phi)) \tag{2.31}$$

where I is the input traffic rate and γ is a constant set to 0.1. This equation ensures that at least 10% of the traffic is redistributed according to AIMD at every interval control.

Since FC uses AIMD to ensure the fairness, XCP divides the per-packet feedback in a positive (Additive Increase) and a negative (Multiplicative Decrease) feedback

$$H_feedback = p_i - n_i \tag{2.32}$$

When $\phi > 0$, FC has to increase the sending rate of each flow to the same level. The next equation gives the increase to be assigned to every incoming byte seen by the router during a control interval:

$$\xi_p = \frac{h + \max(0, \phi)}{\sum \frac{s_i}{r_i}} \tag{2.33}$$

where s_i and r_i are the size and the input traffic rate of every incoming packet to the router. After computing ξ_p, the next equation estimates the positive feedback for every incoming packet seen by the router during a control interval:

$$p_i = \xi_p \frac{s_i}{r_i} \tag{2.34}$$

Concerning the negative feedback, n_i, the authors discovered that n_i should be proportional to the packet size. The next equation gives the decreasing value for every incoming byte seen by the router during a control interval:

$$\xi_n = \frac{h + \max(0, -\phi)}{\sum s_i} \tag{2.35}$$

Therefore, the negative feedback should be computed as :

$$n_i = \xi_n . s_i \tag{2.36}$$

Finally, if the computed feedback per packet is smaller than the value of the *H_feedback* field from the XCP header, then the value of this field will be replaced by the new feedback per packet.

XCP in action

The Figure 2.11 shows the results of a simulation, made by the XCP authors, of 4 XCP flows sharing the bottleneck. In Figure 2.11, we can see the high performance and fairness of XCP.

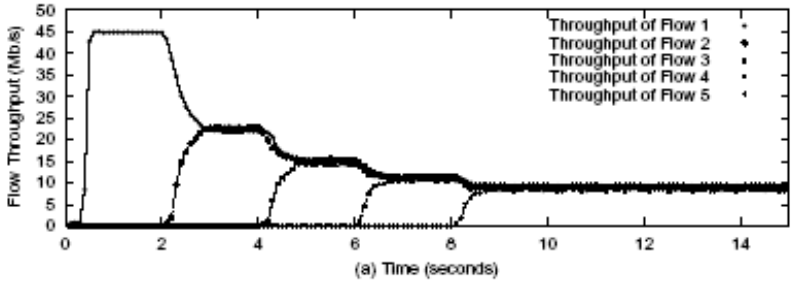


Figure 2.11: 4 XCP flows (from [3]).

2.5.3.2 JetMax

JetMax, described in [4], is a very recent proposition that belongs to the group of ERN protocols. As in the XCP protocol, in JetMax all the mechanisms for controlling congestion in the network are found in the routers. JetMax also needs to modify the packet header of standard TCP, the sender, the receiver, and of course, the routers.

The JetMax packets

The JetMax packets add 7 main fields to the TCP header. They are:

1. R_T: is a one-byte router-ID. This ID is given in terms of hop count from the source. R_T records the router ID of the current bottleneck link b_r for a given flow r.

2. R_C: is a one-byte router-ID. This ID is given in terms of hop count from the source. R_C carries the hop number of the packet (or the current router-ID).

3. R_S: is a one-byte router-ID. This ID is given in terms of hop count from the source. R_S contains the ID of the suggested bottleneck router.

4. *Packet loss* (p_l^+): contains the largest packet loss found in the path. We will describe later how it is calculated at every router.

5. *Fair rate* (g_l^+): contains the smaller sending rate assigned to the flow. We will describe later how it is calculated at every router.

6. *Proposed size* (s_k^+): carries the size of the packet (or the proposed size of a packet).

7. *Inter packet interval* (δ_k): contains the interval time between two packets sent.

The JetMax Sender

The JetMax sender must initialize the fields in the header of the i^{th} packet, sent at the time t, as follows:

R_T and R_S have to be initialized with the ID of the last bottleneck router known by the sender. R_C has to be initialized with 0. p_l^+ is initialized with the largest packet loss that should correspond to the packet loss given by the router R_T. g_l^+ must contain the maximum allowed sending rate, that should correspond with the available bandwidth indicated by R_T. s_k^+ must be initialized with the real packet size of the packet sent and δ_k must contain the interval time between two packets sent, calculated as $\delta_k = s_k/x(t)_r$, where $x(t)_r$ is the current sending rate (sending rate at time t).

During the sending of packets, a JetMax sender must send packets exactly every δ_k time and not by burst of packet. Sending packets every δ_k time will let the JetMax routers compute the exact number of active flows, which is critical for achieving good fairness level.

At the reception of the ACK corresponding to the packet i, if the suggested router R_S is different from the router R_T, the sender must execute $R_T = R_S$. In addition, the sending rate of the sender should be updated as follows:

$$x(t+1)_r = x(t)_r - \tau\left(x(t)_r - g(t+1)_l\right) \tag{2.37}$$

where $\tau > 0$ is the gain parameter. The last equation encourages the sender to increase its rate when the resources are under utilized or to decrease its rate in order to achieve a resources fair share between end users.

The JetMax Receiver

The JetMax receiver works as the XCP receiver. It means that the JetMax receiver has to copy every JetMax parameter from the data packet to the ACK packet in order to let the sender update its rate.

The JetMax Router

Upon each packet arrival, a router l increments R_C from the JetMax header by one and then checks whether its local packet loss $p(t)_l$ (see equation 2.38) is greater than the one carried in the packet. If both packet values are zero, the router checks if its local average rate $g(t)_l$ (see equation 2.39) is less than the one carried in the header. If either case is true, the router overwrites the packet loss and average rate in the packet header and additionally sets the R_S field of the packet to the value of R_C obtained from the header, to indicate that the current router is more congested and that it will manage the sending rate of this data flow.

The packet loss $p(t)_l$ is calculated as follows:

$$p(t)_l = \frac{y(t)_l - \gamma_l C_l}{y(t)_l} \tag{2.38}$$

where $y(t)$ is the input traffic rate during the last control interval, γ_l is the desired utilization level and C_l is the output link capacity.

On the other hand, the local average rate $g(t)_l$ is calculated as follows:

$$g(t)_l = \frac{\gamma_l C_l - u(t)_l}{N(t)_l} \tag{2.39}$$

where $u(t)_l$ is the aggregate rate of unresponsive flows (flow not controlled by the current JetMax router, that could be non-JetMax traffic, ACK packets or simply, JetMax traffic whose rate is controlled by another JetMax router). $N(t)_l$ is the number of responsive flows, calculated as the sum of all the inter packet interval δ_k of every incoming packet received during a control interval, divided by the control interval duration Δ_l:

$$N(t)_l = \frac{\sum \delta_k}{\Delta_l} \tag{2.40}$$

$N(t)_l$ is then a critical value for achieving fairness between JetMax flows, which depends on the value of δ_k and the number of received packets during a control interval Δ_l. Therefore, a wrong calculation of δ_k and/or the failure of sending packets every δ_k milliseconds could be translated in unfairness.

JetMax in action

The Figure 2.12 shows the results of an experiment made by the JetMax creators, using Linux systems configured as JetMax routers. In the experiment, 4 JetMax flows share the bottleneck. Each flow starts with a 15-second delay and lasts for 75 seconds. In Figure 2.12, we can see the high performance and fairness of JetMax.

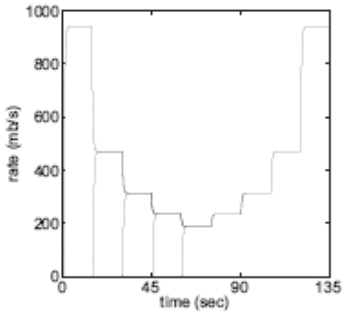


Figure 2.12: 4 JetMax flows (from [4]).

2.5.3.3 TCP Quickstart

TCP Quickstart or simply Quickstart, defined in [79] in January 2007, is a protocol less complex than XCP and JetMax. Quickstart does not need to provide to the routers information

like the RTT, the current *cwnd*, etc. A sender implementing Quickstart only needs to turn on an IP option and to provide the desired rate inside a SYN packet, in order to announce to the routers that this is Quickstart sender and at the same time, ask for the desired rate.

If every router in the flow path are Quickstart-capable, then the sender will be able to use this protocol. In order to discover if all the routers in the path are Quickstart-capable, Quickstart uses a dual-TTL strategy very similar to the one we proposed in 2006 for XCP [80]. This strategy will be widely described in Chapter 4, since it belongs to the set of contributions made by this thesis. If at least one of the routers does not support Quickstart, then the sender must switch to another congestion control protocol.

A router supporting Quickstart must compute its available bandwidth. If the available bandwidth is above or equal to the sender asked rate, the router will not modify the desired rate carried in the packet. In the opposite case, the current router must replace the asked rate carried in the packet by the computed available bandwidth.

It is very important to remark that Quickstart does not execute any fairness mechanism in order to fairly share the resources between the Quickstart-capable senders. This way, one flow could take a large amount of resources, damaging new flows.

2.5.3.4 Limitations of ERN Protocols in Heterogeneous Networks

Using ERN protocols in networks could be very advantageous, since this kind of protocols greatly improves the performance and the fairness of flows. Thus, the throughput of flows using ERN protocols stay in the *knee* without arriving to the *cliff* (see figure 2.2). ERN protocols are also able to stay in the intersection point of the fairness and the efficiency line, with exception of Quickstart that does not have any fairness mechanism (see Figure 2.6).

But the main weaknesses of Routers-assisted congestion control protocols could be summarized in three points that prevent the deployment of such protocols in real heterogeneous LDHP networks. It is exactly over these points that we have focused the contributions of the studies made during this thesis: (*i*) they need the collaboration of all routers in the forward path [80]; (*ii*) they are unable to cohabit with E2E protocols [81]; and (*iii*) all the information are given to the sender by mean of ACK packets [82]. We will not add any more about the problems that prevent the deployment of ERN protocols in heterogeneous LDHP networks in this section, since they constitute the heart of the works presented in this thesis. We will widely discuss the deployment problems of the ERN protocols in the next chapters.

Finally, even if our studies are focused on the problems of inter-operability of ERN protocols in heterogeneous LDHP networks, our contributions are specifically focused on XCP. The reasons of our choice is that XCP was one of the first proposed ERN protocols (the second most known protocol, JetMax, was proposed in 2006). In addition, many analyzes about XCP have been made, both mathematically [3, 83, 82, 80, 81] and experimentally ([84]) to understand how XCP works. However, it is important to remark that since the problems of inter-operability that we tackle are not inherent to the XCP mechanisms, but to the general architecture of the ERN protocols, such problems are common to others router-assisted approaches.

2.6 Conclusion

In this chapter we have shown that congestion control mechanisms are necessary in networking to avoid congestion collapses, that could potentially break the process of exchange of data

between end hosts. We have also presented the existing approaches for controlling congestion inside a network. We have classified these approaches into (*i*) propositions to be implemented in end hosts, that do not require any collaboration from forwarding network nodes, and (*ii*) propositions that use the forwarding network nodes to control the congestion.

The proposition to be implemented in end hosts are known as E2E congestion control protocols and, as we have seen throughout this chapter, they have been unable to solve the problems of performance and fairness in the networks. The main problems of E2E protocols are based on their inefficiency to correctly estimate the state of the network (to predict when congestion is imminent).

On the other hand, approaches especially designed for forwarding network nodes (routers more specifically) try to tackle the deficiencies of E2E congestion control protocols. These approaches try to avoid congestion collapses by discarding packets when congestion is imminent (AQM mechanisms), or providing information to the end hosts about the state of the networks. Since AQM mechanisms do not interact with end hosts, the improvements offered by such mechanisms in the throughput and fairness of active flows is very limited.

Mechanisms for routers, providing notifications about the state of the networks to the end hosts, could be divided in mechanisms feeding back a binary signal (congestion is present or not), and mechanisms feeding back a specific rate. While mechanisms providing binary signals are built on top of AQM algorithms (and therefore inherit the same weakness as these), mechanisms providing ERN to the end hosts rebuild totally the architecture of E2E and AQM congestion control mechanisms, constituting new solutions to the problems of congestion and fairness in networking.

Since ERN protocols do not make assumptions about the state of the networks, they achieve surprisingly high throughput and fairness levels, as shown earlier. Thus, ERN protocols seem to be promising solutions in congestion and fairness control. However, ERN protocols have others problems that prevent the deployment of such approaches in current networks, that could be summarized as follows: the incompatibility (non inter-operability) of ERN protocols with currently existing mechanisms in networking. Thus, the heart of the contributions made in this thesis is focused specifically on the propositions of mechanisms able to become ERN protocols inter-operable in heterogeneous LDHP networks.

Moreover, at the time when the researches about the problems of inter-operability of ERN protocols in heterogeneous networks were made, the XCP was the best-known and the most representative of the ERN protocols, caused by a detailed description of XCP, by its authors, in published papers (unlike the Congestion Avoidance with Distributed Proportional Control / Performance Transparency Protocol (CADPC/PTP) protocol [77] for instance), and a quick publication of the XCP modules for the ns2 [10] network simulator. Consequently, most of the analyzes of performance, theoretical and experimental, concerning ERN protocols were focused on the XCP protocol. For all these reasons (*i*) all the problems about the lack of inter-operability of ERN congestion control protocols have been detected by analyzing the XCP protocol, and (*ii*) all our solutions have been adapted to the XCP protocol. However, it is very important to remark that, since the problems that we discuss are directly related with the architecture of ERN protocols, and are independent to the XCP mechanism in itself, the proposed solutions can be used in any other ERN protocol.

However, before addressing the problems of inter-operability of ERN protocols, in the next chapter, we will first of all test XCP, and we will compare its performance with those of others protocols, like TCP and HSTCP. Thus, Chapter 3 aims at proving that XCP offers the best performance, even under very hard conditions.

Chapter 3

E2E and ERN Protocols on Variable Bandwidth Environments

As we said in Chapter 2, TCP performances have been extensively studied by the research community this last decade, both theoretically [85, 86, 87, 88, 89] and experimentally [90, 91]. Many enhancements and optimizations to the original TCP protocol have been made in order to support a wider range of network conditions ([92, 93, 40] to name a few).

Most of the studies, about TCP and others congestion control protocols, have assumed that the bottleneck bandwidth remains constant over time. However, major changes are foreseen as many telco operators and Internet Service Providers (ISPs) are beginning to deploy QoS protocols, like Resources Reservation Protocol - Traffic Engineering (RSVP-TE) [94]. These protocols introduce reservation-like or priority-like mechanisms in their networks (both access and backbone networks) to guarantee a given QoS level to certain flows. As a result of the aggregation/disaggregation of flows implementing reservation-like or priority-like mechanisms, fluctuations in the available bandwidth can be introduced. Thus, the hypothesis that the available bandwidth for best-effort traffic[1] is usually constant is not always true. In addition, we can note that some network technologies proposed may also introduce dynamic bandwidth features [95, 96], since these technologies propose dynamic bandwidth allocation to improve the QoS services. The new technologies, along with the strong desire to provide bandwidth-on-demand features, may turn wired networks into highly Variable Bandwidth Environments (VBEs) for the best-effort traffic.

Thus, some researchers have analyzed the response of TCP to different level of congestion in low speed networks, by mean of control and system theory. For instance, in [97] the authors show that TCP behavior can become chaotic in networks with UDP background traffic and DropTail routers with small buffers. In addition, other studies in [98] and [99] show that the TCP response suffers from several problem in presence of a step increase of network congestion.

On the other hand, in [100], Dutta and Zhang showed that TCP (New Reno) behaves quite badly when the bandwidth is varied over time (by mean of a sine-based function), in networks with low capacity (between 1 and 1.5 Mbps) and large RTTs ($RTT \approx 200ms$).

In this chapter, since our research is oriented to heterogeneous large BDP networks, we will study congestion control protocols in VBE networks with higher capacity (1Gbps) while keeping large RTTs ($RTT \approx 200ms$). In fact, the performances of congestion control protocols are different in networks with low capacity (as those used by Dutta and Zhang [100] in their

[1] Traffic with no guaranteed resources.

experiment), compared to networks with high capacity (where the available bandwidth exceeds several hundreds of Megabits per second).

Our researches in this field focus on modeling, analyzing and comparing the response of E2E and ERN protocols, in order to make fundamental observations. Congestion Control Protocols studied here are: (*i*) TCP (New Reno), since TCP is currently the most used congestion control protocol in networks; (*ii*) HSTCP because at the time when our research began (September 2004), HSTCP was the most promising approach for large BDP networks; and (*iii*) XCP, since at the time when our research began, XCP was the most representative and best-known freely available ERN protocol (see Chapter 1, Section 1.3).

In a first step, we present a sinusoidal-based bandwidth variation model for LDHP networks (inspired from the works presented by Dutta and Zhang [100]) that aims at representing the aggregation/disaggregation of priority flows in networks. However, since in real life the aggregation/disaggregation of priority flows leads to variation of the available bandwidth while occupying memory and CPU of routers, we believe that our sinusoidal-based model, that only introduces bandwidth fluctuations without affecting the resources in the routers, fails when reflecting environments with aggregation/disaggregation of priority flows.

Thus, we propose a step-based bandwidth variation model, that improves the sine-based bandwidth variation model, since it introduces bandwidth fluctuations while varying the available resources in routers (like the buffer memory). Then, we present the details concerning our sinusoidal-based and step-based bandwidth variation models as well as the results of our experiments on congestion control protocols in VBEs.

3.1 Sinusoidal-based bandwidth variation environments

3.1.1 Definition

So-called continuous variation models are those where the increase or decrease of the available bandwidth for the best effort traffic does not present significant jumps as time is varied. In the real world, pure continuous variations (in the mathematical meaning) do not exist as the granularity of bandwidth is at the packet level. However, if the change in bandwidth is not significant when Δt is small then we will refer to such variations as continuous.

Figure 3.1 shows a 1Gbps link with both pure sine-based bandwidth variations from 1Gbps to 300Mbps with a period of 30s (dashed line) and a more complex variation model (continuous line). The y-axis represents the bandwidth available for best-effort traffic over time (therefore the bandwidth for guaranteed traffic is 1Gbps-y). This bandwidth variation model could represent a large number of guaranteed flows coming in and out, resulting in variations for the best effort traffic.

Dutta and Zhang [100] used a pure sine-based variation model to study TCP's performances. They did not claim that such a simple function could represent closely the real variations (that could be found in the Internet, for example), but that such a model could be used to evaluate TCP in a VBE. We totally agree with this statement and use the pure sine-based variations to model continuous variations, even if we believe that the more complex variation model represents better the reality. In fact, we believe that in real life the bandwidth variations do not follow a symmetric pattern as the one shown in the pure sine-based variation model, since flows with QoS services are started and stopped asynchronously in the network.

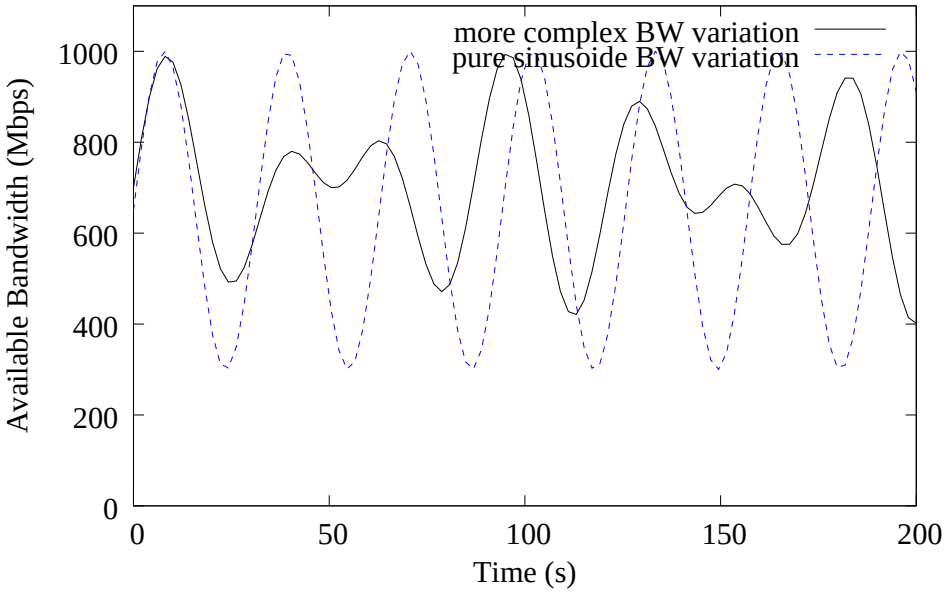


Figure 3.1: Sinusoidal-based bandwidth variation model.

3.1.2 Network topology for simulating sinusoidal-based models

The topology network used to simulate our sinusoidal-based bandwidth variation model is shown in Figure 3.2. Our goal being to test the behavior of TCP, HSTCP and XCP over large BDP networks with VBE. Thus, we will use a bottleneck link of 1 Gbps that will vary using a sinusoidal function. The bottleneck propagation will be fixed to 50ms ($RTT \approx 100ms$ - First case[2]) or 100ms ($RTT \approx 200ms$ - Second case).

Concerning routers, both R0 and R1 have buffers with a capacity equivalent to 12500 MSS, which is large enough to get a throughput of 1Gbps with an $RTT \approx 200ms$ in ns. We believe also that a buffer with a capacity smaller than the BDP value represents better the reality, due to the small capacity of routers buffers in the real life. For TCP and HSTCP we have used DropTail routers, while in the case of XCP we have always used XCP routers.

The function described by the available bandwidth will be the same as the one shown in Figure 3.1 (pure sinusoidal function). The maximum available bandwidth will be 1Gbps and the minimum 300Mbps. For TCP and HSTCP we will use DropTail routers and for XCP, we will use XCP routers only.

3.1.3 TCP on sinusoidal-based bandwidth variation environments

3.1.3.1 Modeling TCP

On a steady-state, TCP is most of the time in the congestion avoidance phase and therefore most of the studies have focused on evaluating TCP performances when the congestion window

[2]The RTT is equal to the sum of the propagation delay of the links crossed by a data packet and its respective ACK, plus the time of treatment γ of those packets in every crossed forwarding device. In order to label cases where the topology remains the same but the propagation delay changes, we will suppose that in absence of congestion $\gamma \approx 0$ and therefore the RTT is approximately equal to the sum of the propagation delay of every link

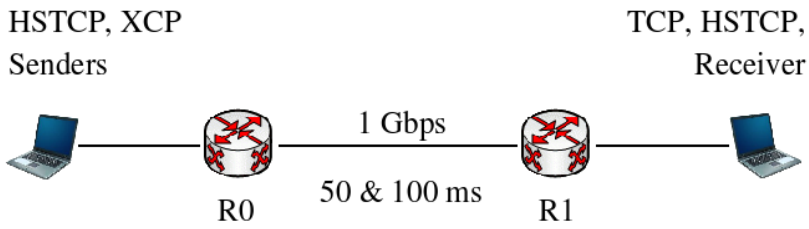


Figure 3.2: Topology network for the sinusoidal-based bandwidth variation model.

is increased linearly (1 packet every RTT). As the RTT increases, the throughput increment per second is decreased. In a general way, we can use a simple relation to link the sender instantaneous throughput (T) to the congestion window size ($cwnd$) as follows:

$$T(t) = \frac{cwnd(t) * 8 * MSS}{RTT} \tag{3.1}$$

where T is expressed in bits/s and the MSS in bytes (usually a value of 1024 is used).

Since we consider the congestion avoidance phase, given an initial congestion window $cwnd(t_a)$ at second t_a, $cwnd$ at second t could be calculated as:

$$cwnd(t) = cwnd(t_a) + \frac{(t - t_a)}{RTT} \tag{3.2}$$

In parallel, in absence of losses and during congestion avoidance, the throughput T could be expressed as a linear function of t as follows:

$$T(t) = \frac{T_b - T_a}{t_b - t_a}(t - t_a) + T_a \tag{3.3}$$

where T_a and T_b are respectively the throughput at time t_a and t_b. Finally, replacing Equations 3.1 and 3.2 in Equation 3.3, $T(t)$ could be expressed as follows:

$$T(t) = \frac{8 * MSS}{RTT^2}(RTT * cwnd(t_a) + (t - t_a)) \tag{3.4}$$

This very simple relation links the throughput increase slope to the RTT. In the remaining of this section, we will use these relations to either get $cwnd$ as a function of T, or T as a function of $cwnd$.

Under sinusoidal-based bandwidth variation models, Figure 3.3 depicts a simple view of a TCP connection and shows as an example a throughput increase slope of about *0.83Mbps* [3] (continuous line).

In Figure 3.3, we have represented with a continuous line one possible evolution of the sender throughput. This is a simple view of a TCP connection since slow-start has not been taken into account. Moreover, when the throughput line crosses the bandwidth variation curve while it is decreasing (it is actually the only possibility), there are drops immediately (we do not take into account the capacity of the routers buffer). Thus, when drops occur, if the throughput increase slope is much smaller than the bandwidth increase slope then the

[3]This acceleration should be valid for TCP flow with a RTT of 100ms

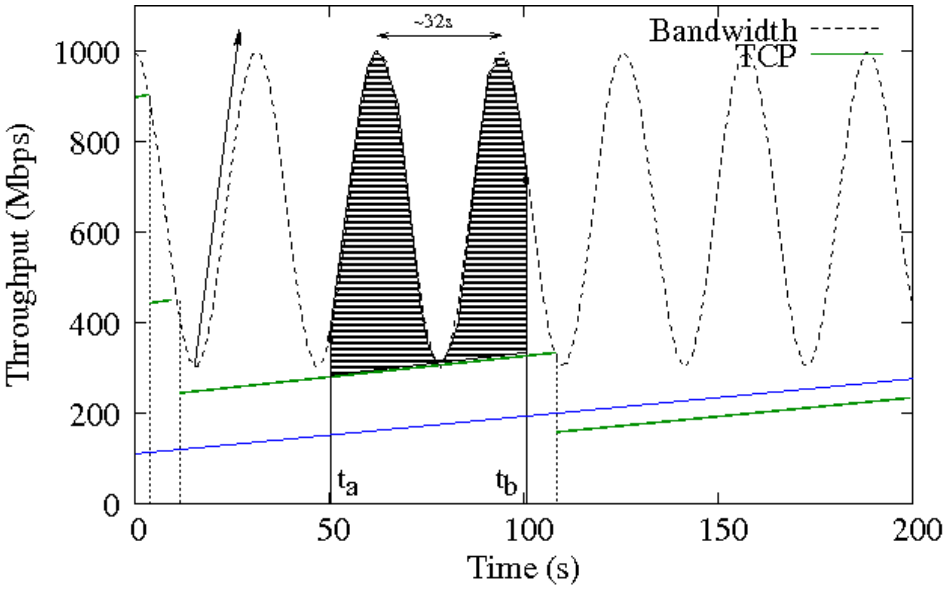


Figure 3.3: Simple view of a TCP connection facing VBE.

efficiency is very low. In fact, to get a good performance, a TCP flow under these conditions should accelerate its sending rate like the black arrow placed at second 15, in parallel to the available bandwidth evolution (dashed line from Figure 3.3).

When the throughput of a TCP flow does not get all the available bandwidth, the amount of wasted bandwidth is the surface captured by the bandwidth variation curve and the throughput increase line. In Figure 3.3, as an example, we have filled in with horizontal lines the wasted bandwidth between time $t_a = 50s$ and time $t_b = 100s$.

Mathematically, we could express the inefficiency between time t_a and time t_b with the following relation:

$$I_{ab} = 1 - \frac{T_{ab}}{B_{ab}} \tag{3.5}$$

where B_{ab} is the available bandwidth and T_{ab} the achieved throughput between time t_a and time t_b. B_{ab} and T_{ab} could be expressed by:

$$B_{ab} = \int_a^b B(t)dt \tag{3.6}$$

$$T_{ab} = \int_a^b T(t)dt \tag{3.7}$$

$T(t)$ is known and has been expressed in Equation 3.4. If we get back to the simple sine variation depicted in Figure 3.3 where we used a sine-based B(t) function defined as follows:

$$B(t) = (max - \frac{max - min}{2}) + \frac{max - min}{2} * sin(\frac{2\pi}{\tau}t + \frac{\pi}{2}) \tag{3.8}$$

with $max = 1Gbps$, $min = 300Mbps$ and $\tau = 32s$; then it is not difficult to obtain the inefficiency in this case (Equation 3.5).

The model presented in Figure 3.3 assumes that, after packet, losses the sender reacts by halving its congestion window and the buffer in the routers cannot store the overload. However, in high speed real systems the buffer of intermediate routers could keep the overload in the bottleneck, delaying the time to detect any congestion problem. This delay could vary, depending on the capacity of the routers. In summary, 2 scenarios are possible when the bandwidth decreases: (i) the buffer is almost empty and (ii) the buffer is almost full. In the first scenario, buffers at the bottleneck link would fill up and compensate in some extend the bandwidth decrease, avoiding losses and resulting in a perfect match between the available bandwidth and the throughput functions (case a). In the second scenario, the buffer saturates and packets are dropped. Depending on the number of lost packets, either fast retransmit/fast recovery would be enough to handle the losses (case b), or a timeout would be triggered (case c). All three cases are better depicted in Figure 3.4.

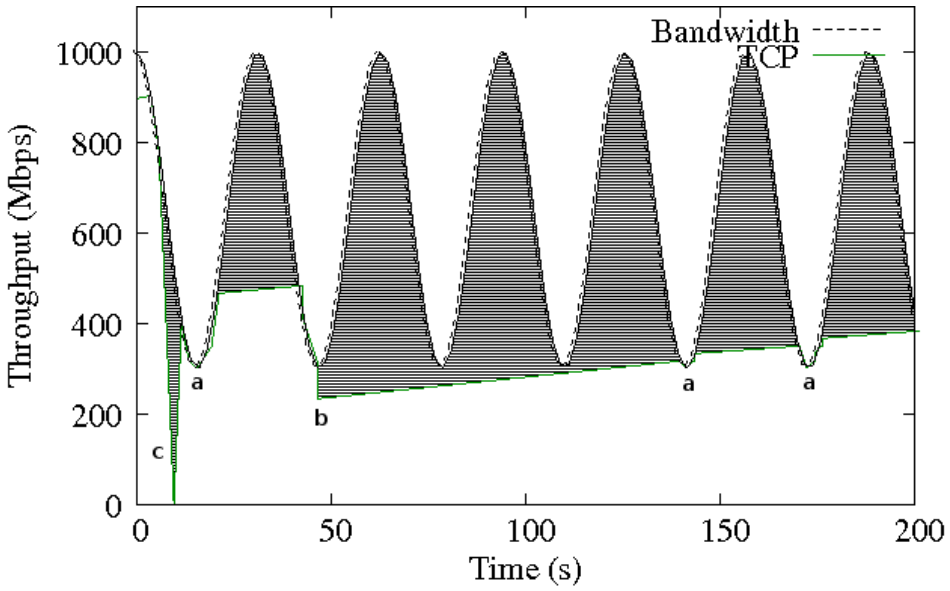


Figure 3.4: More accurate TCP behavior with cases a, b and c.

3.1.3.2 Simulation results for TCP

In order to test TCP in our sinusoidal-based bandwidth variation model, first of all, we have tuned the TCP parameters (the slows-start threshold, $ssthresh$ and the maximum congestion window size, max_win, mainly) to get 1Gbps when the available bandwidth does not change and the $RTT \approx 100ms$. Once having a well tuned TCP stack[4] for this conditions, we executed a TCP New Reno version provided by the ns2 network simulator (version 2.30). Figure 3.5 shows the throughput for this case.

In Figure 3.5, we can see clearly the cases a and c depicted in Figure 3.4. First, at the beginning of the simulation and before the TCP connexion suffers a Timeout, we notice in the Figure 3.5 that for some seconds TCP gets all the available bandwidth, because the buffers in

[4]We say that TCP is well tuned, when some TCP parameters like $ssthresh$ and max_win let a flow to grab all the available bandwidth during the startup (slow-start).

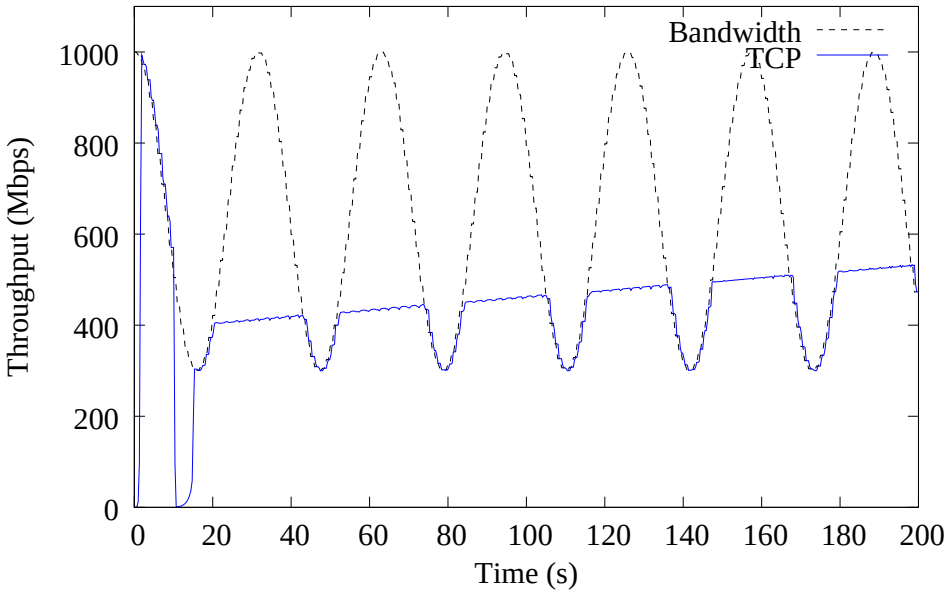


Figure 3.5: TCP New Reno ($RTT \approx 100ms$) on a sinusoidal-based VBE.

the routers compensate the decrease of the available resources. However, since the congestion window becomes very large, when the buffer cannot compensate the loss of resources, a large quantity of packets belonging to the same congestion window are lost, producing a Timeout. This part corresponds to the case c shown in our model (Figure 3.4).

The case a, illustrated in Figure 3.4, can be observed in the simulations results shown in Figure 3.5, after second 20, every time the available bandwidth gets its smaller capacity. The case b does not happen during the period of simulation, because the buffers of the routers can always store the overload produced by the TCP sender.

Complementarily, note that the acceleration computed in Subsection 3.1.3.1 (see Figure 3.4), for a TCP connection with an $RTT \approx 100ms$, is very similar to the acceleration observed in Figure 3.5. Finally, we computed the Inefficiency I in a period between seconds 16 and 48. Using the Equation 3.5 we computed an inefficiency $I = 0.40$, meaning that TCP wasted 40% of the available resources.

An $RTT \approx 100ms$ (for the first case) can be an optimistic scenario for TCP on large BDP networks, since we can find scenarios where the average RTT can be easily doubled. With this assumption, we tested TCP in a scenario where the $RTT \approx 200ms$.

The throughput of TCP, whit an $RTT \approx 200ms$, is shown in Figure 3.6. In this case, TCP will need a lot of time (maybe hours) to get at least 500Mbps. As it is expected, the responsiveness of TCP decreases as the RTT increases. In Figure 3.6, after second 45, we can observe that the throughput of TCP displays frequent pulsations. We believe that this behavior is caused by the burstiness nature of TCP and the frequency at which the throughput is measured. However, we are not sure about the reasons behind this phenomenon, since between seconds 30 and 45 the pulsation are not present.

To finish with the section of TCP in a sinusoidal-based bandwidth variation model, we want to remark that in any of both cases $RTT \approx 100ms$ and $RTT \approx 200ms$, the TCP throughput is not able to follow the bandwidth variation. We can see also that the sinusoidal-

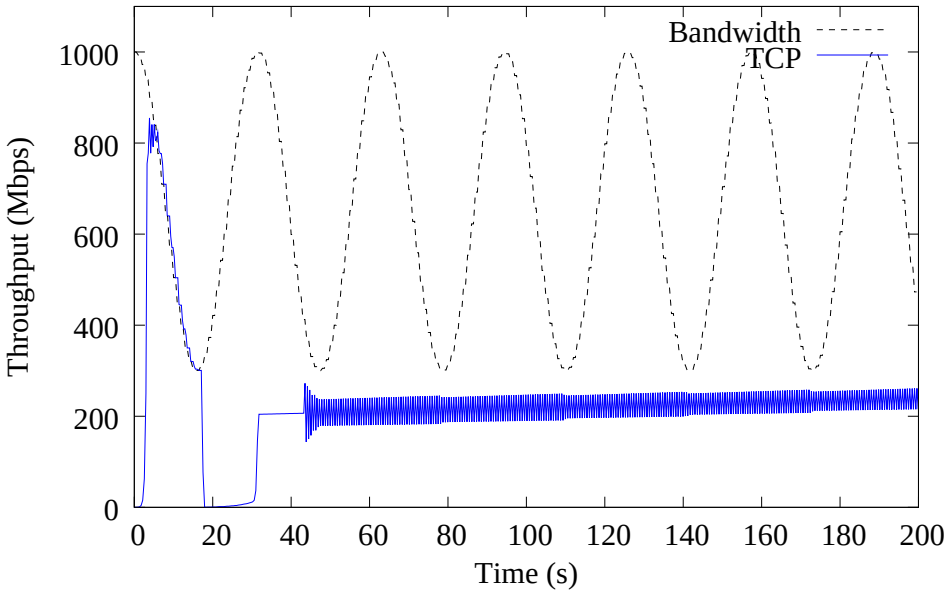


Figure 3.6: TCP New Reno ($RTT \approx 200ms$) on a sinusoidal-based VBE.

based bandwidth variation model does not have a very important impact on TCP, after the first losses (at second 10 approximately). In fact, after the first losses of TCP senders, it is the TCP algorithm rather than the bandwidth variations that prevents the sender grabbing the available resources.

3.1.4 HSTCP on sinusoidal-based bandwidth variation environments

In section 3.1.3.2, we observed that TCP was not aggressive enough to recover the resources when the available bandwidth increased. Going back, when we modeled TCP (see section 3.1.3.1), we said that to let TCP get a good performance, its acceleration rate should be much more aggressive. HSTCP tries to solve the problems of TCP by increasing the aggressiveness of TCP as the available bandwidth increases.

Figure 3.7 shows the behavior of HSTCP in our sinusoidal-based bandwidth variation model, with an $RTT \approx 100ms$. In Figure 3.7, we can perceive that most of the time, the HSTCP throughput (continuous line) matches very well the available bandwidth (dashed line) during the increases or decreases periods of the bandwidth. Only when the available bandwidth exceeds $\sim 800Mbps$, HSTCP is unable to grab quickly all the resources. The fact that the HSTCP throughput matches perfectly the bandwidth when the available resources decrease does not mean that there are no losses. In fact, Figure 3.7 shows some reductions of the congestion window size (the doted line), caused by lost events which occur when the bandwidth decreases.

The decrease of the sending rate, caused by the decreases of the congestion window, cannot be observed in the function described by the throughput (Figure 3.7), since the congestion window is always large enough to keep the bottleneck link (and a percent of the routers buffer) plenty. Later, when the available bandwidth increases, the packets stored in the buffers are sent, while the sender increases its rate. When the buffers become empty and the acceleration

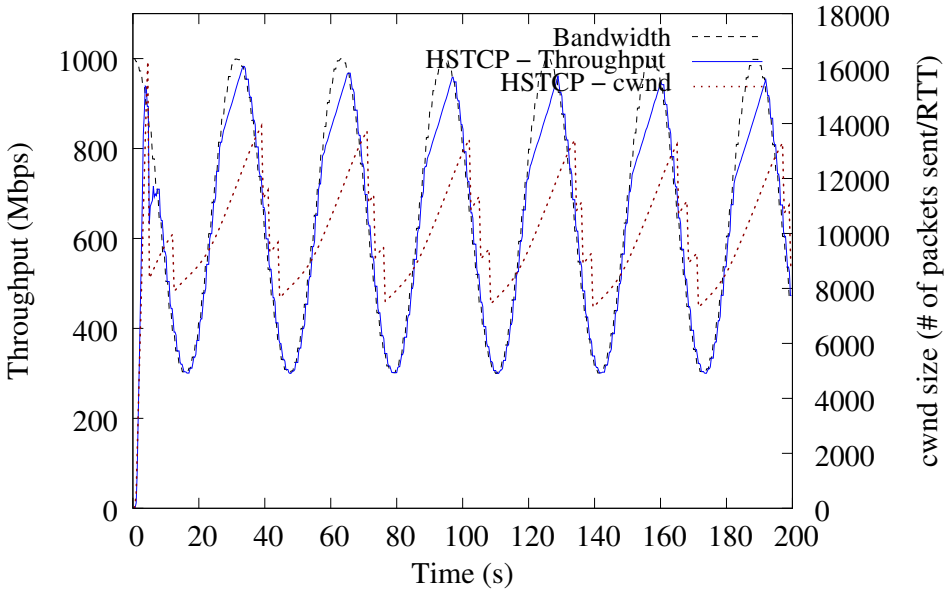


Figure 3.7: HSTCP ($RTT \approx 100ms$) behavior on a sinusoidal-based VBE.

of the sender rate is slow in comparison with the available bandwidth growth, HSTCP is not able to fill up the resources (phenomenon observed every time the available bandwidth exceeds $\sim 800Mbps$). HSTCP with an $RTT \approx 100ms$, in our sinusoidal-based bandwidth variation model, gets an inefficiency I smaller than 5%.

The results of HSTCP when the $RTT \approx 200ms$ are shown in Figure 3.8, where we have plotted the throughput (continuous line) and the bandwidth evolution (dashed line). It is important to remark that the increases of the RTT impact negatively the HSTCP response, maybe more than expected. Using the equation 3.5, we computed an inefficiency of $I = 0.30$ for a period between seconds 16 and 48, proving the bad performance of HSTCP on this scenario. Summarizing, HSTCP with a $RTT \approx 200ms$ is only 10% more efficient than TCP with a $RTT \approx 100ms$. In conclusion, the impact of larger RTTs (e.g., 200ms) on HSTCP can be very important, in large BDP networks with VBE.

3.1.5 XCP on sinusoidal-based bandwidth variation environments

With HSTCP, we observed that increasing the aggressiveness of TCP does not solve all its problems in large BDP networks, since the value of the RTT always influence strongly the responsiveness of the aggressive TCP versions.

In this subsection, we will present the simulation results using the XCP protocol. The main ideas are to determine (i) the behavior of XCP on sinusoidal-based bandwidth variation model, and (ii) the sensibility of XCP to the RTT values. Note that all the routers, from the topology shown in Figure 3.2, have been enabled as XCP routers for the experiments concerning XCP.

The simulations results of XCP on sinusoidal-based bandwidth variation models with an $RTT \approx 100ms$ are shown in Figure 3.9, where we have plotted the available bandwidth evolution (dashed line), the throughput (continuous line) and the congestion window (dotted

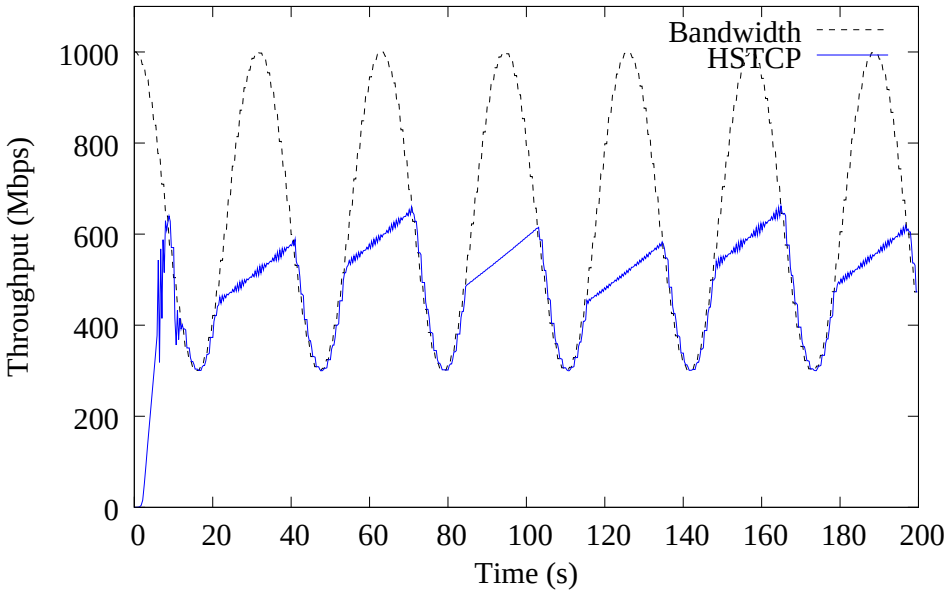


Figure 3.8: HSTCP ($RTT \approx 200ms$) on a sinusoidal-based VBE.

line).

As we can see in Figure 3.9, during the simulation of XCP in our sinusoidal-based bandwidth variation model with an $RTT \approx 100ms$:

1. the throughput of XCP matches very well the available bandwidth evolution,
2. the function corresponding to the congestion window follows perfectly the gradual changes of the available bandwidth, and
3. no packets have been dropped during all the period of simulation.

The success of XCP is based on the fact that XCP senders do not make assumptions about the state of the network. Instead, XCP routers inform the sender what is the optimal rate (see Chapter 2, Section 2.5.2.1). In this way, the sender decreases or increases its congestion window as necessary.

Now, focusing our attention on the simulations results of XCP where the $RTT \approx 200ms$ (Figure 3.10), we can see that the behavior of XCP does not change due to the increase of the RTT, showing a performance higher than the performance of HSTCP. Still, the throughput of XCP matches very well the function described by the available bandwidth. The evolution of the congestion window (not shown graphically) follows perfectly the changes of the available bandwidth and no packet loss occurs, like in the first case ($RTT \approx 100ms$). In fact, the good performance of XCP does not depend on the RTT, but on the accuracy of the information given by the routers to the senders.

3.2 Step-based bandwidth variation environments

To represent continuous variation models in the ns2 simulator, we have used a sine-based function that modifies the available bandwidth as desired. This function changes the available

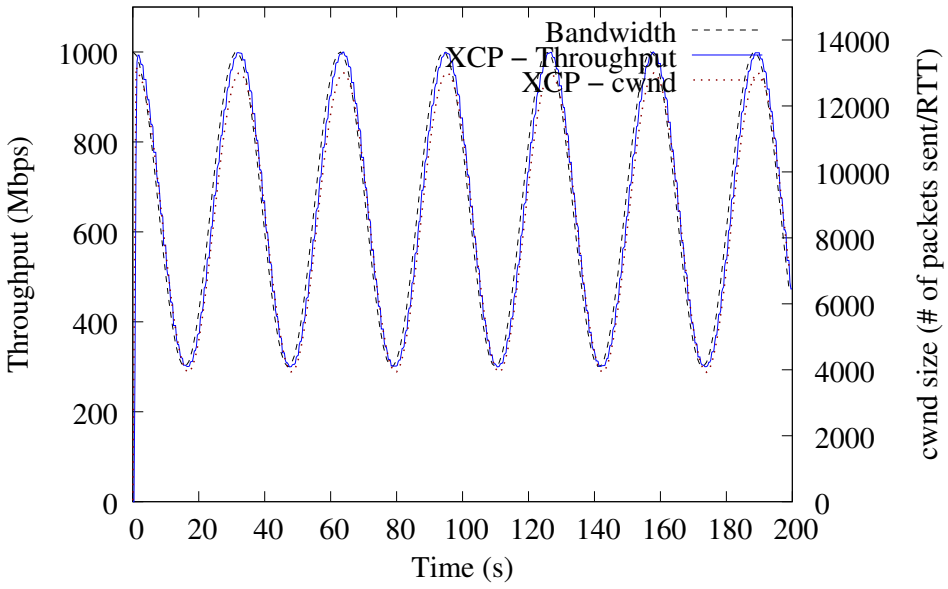


Figure 3.9: XCP ($RTT \approx 100ms$) on a sinusoidal-based VBE.

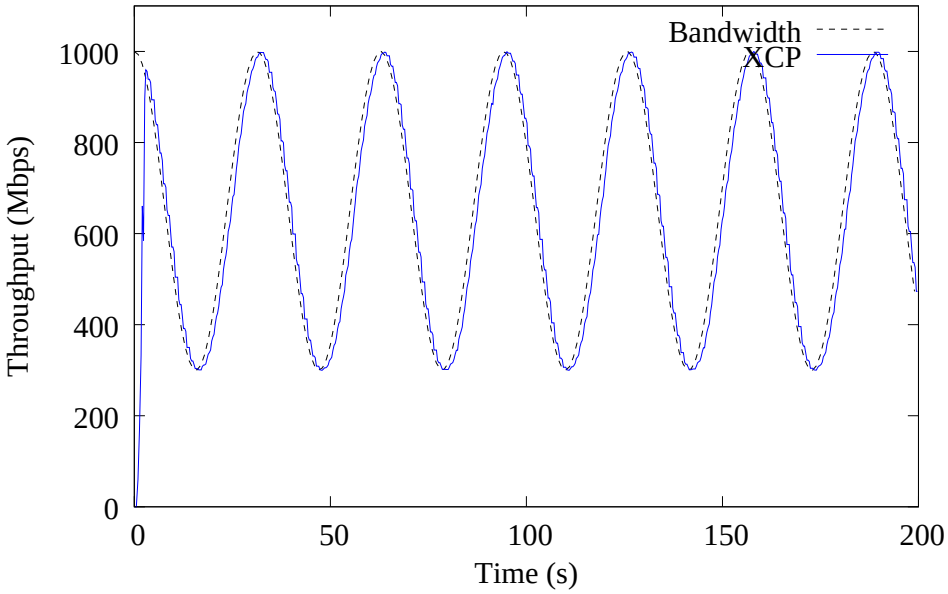


Figure 3.10: XCP ($RTT \approx 200ms$) on a sinusoidal-based VBE.

bandwidth but is not able to modify the size of the buffer in the routers. We believe that on real systems it is different, because the flows using the reserved bandwidth take an important place in the buffer of the routers. Therefore, the simulations results shown in Section 3.1 may be different if we add this new assumption in our experiments.

If we really want to vary the available bandwidth while the available buffer in routers varies also, we need to produce such variations by mean of flows that will simulate the aggregation/disaggregation of highest-priority flows. However, the resulting bandwidth variation model caused by our "highest-priority" flows should describe a discrete function (step-based function), since the granularity of the available bandwidth changes will depend on the rate of flows (from 1MSS/s).

Thus, in this Section we will use a step-based bandwidth variation model, where the bandwidth variations are caused by mean of the aggregation/disaggregation of ON-OFF UDP flows[5]. We want to remark that even though we want to simulate priority flows, the UDP flows used for this purpose, as they are implemented in our script of simulations, have the same priority of all XCP or HSTCP flows, and all flows use the same buffer in the routers. The benefits of this new model are that we will "reserve" and use the available bandwidth, getting a more realistic network with VBE.

We can describe so-called step-based variation models like those where the increase or decrease of the available bandwidth for the best effort traffic does present significant jumps (with discontinuity) as time is varied. Figure 3.11 illustrates this model. When compared to the previous model, the discrete model tries to represent large and sudden variations of the available bandwidth. Such variation patterns could model dynamic bandwidth provisioning of guaranteed traffic, especially in the access networks where the aggregation is not so high.

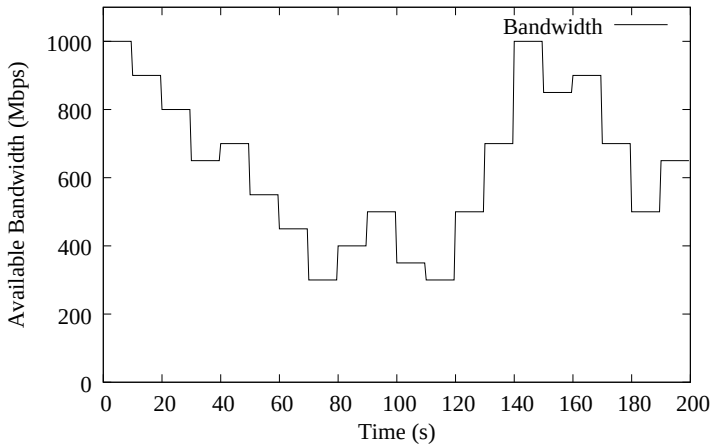


Figure 3.11: Step-based discrete bandwidth variation model.

Before showing our simulation results of congestion control protocols on our step-based bandwidth variation model, we want to remark that standard TCP will not be tested on step-based bandwidth variation model. In Section 3.1.3.2 we saw that TCP is not really adapted for LDHP networks because of the TCP mechanisms themselves. The model used to represent bandwidth variation has only a minimal impact over TCP. Therefore, we consider TCP has

[5]UDP flows transmitting data during 10s with a CBR traffic of 50Mbps each one

no opportunities to get a good performance on this case and we decided in this section to focus on HSTCP and XCP.

3.2.1 Network topology for simulating step-based models

The topology network to simulate our step-based bandwidth variation model is shown in Figure 3.12. We use a bottleneck link of 1 Gbps like in the sinusoidal-based model with a propagation delay of 50ms (first case) or 100ms (second case). Similarly to the sinusoidal model, both routers R0 and R1 have buffers with an capacity equivalent to the 12500 MSS. For HSTCP we have used DropTail routers, while in the case of XCP we have always used XCP routers.

The bandwidth variation introduced by the ON-OFF UDP flows is the same as in Figure 3.11[6]. Note that the available bandwidth will change every 10 seconds, the maximum available bandwidth will be 1Gbps and the minimum 300Mbps. As in the sinusoidal-based bandwidth variation model, we will use DropTail routers when simulating HSTCP and XCP-capable routers for XCP.

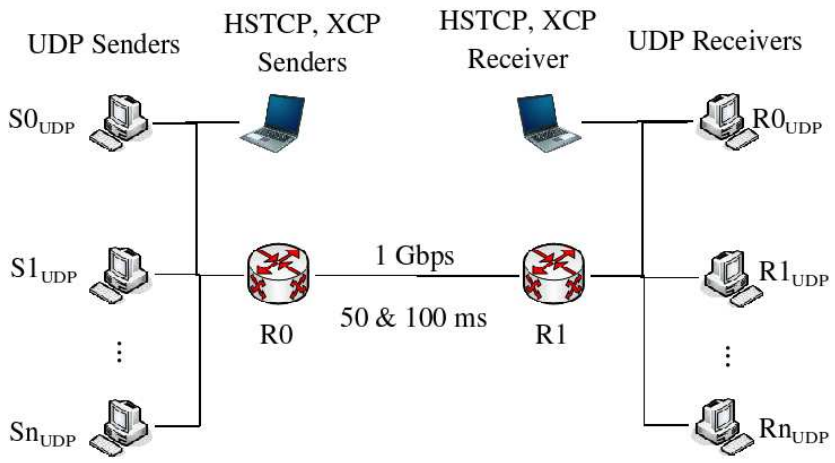


Figure 3.12: Topology network for the step-based bandwidth variation model.

3.2.2 HSTCP on step-based bandwidth variation environments

In the case of $RTT \approx 100ms$, HSTCP succeeds in getting a good performance on step-based bandwidth variation models (see Figure 3.13). According to the Figure 3.13, the throughput of HSTCP (continuous line) follows very well the bandwidth variation (dashed line), adapting its congestion window (represented by the dotted line) to the new conditions. Between seconds 120 and 150, HSTCP has some problems to get the available bandwidth, but the inefficiency level during this period stay lower than 5%.

Some differences between the results obtained with the continuous variation model and the ones obtained with the step-based variation model can be noted. In Section 3.1.4, we saw

[6]We have decided to use the same bandwidth variation pattern shown in Figure 3.11, since it is easier to introduce this kind of bandwidth variation with ON-OFF UDP flows and a constant rate than the bandwidth variation patterns shown in Figure 3.1

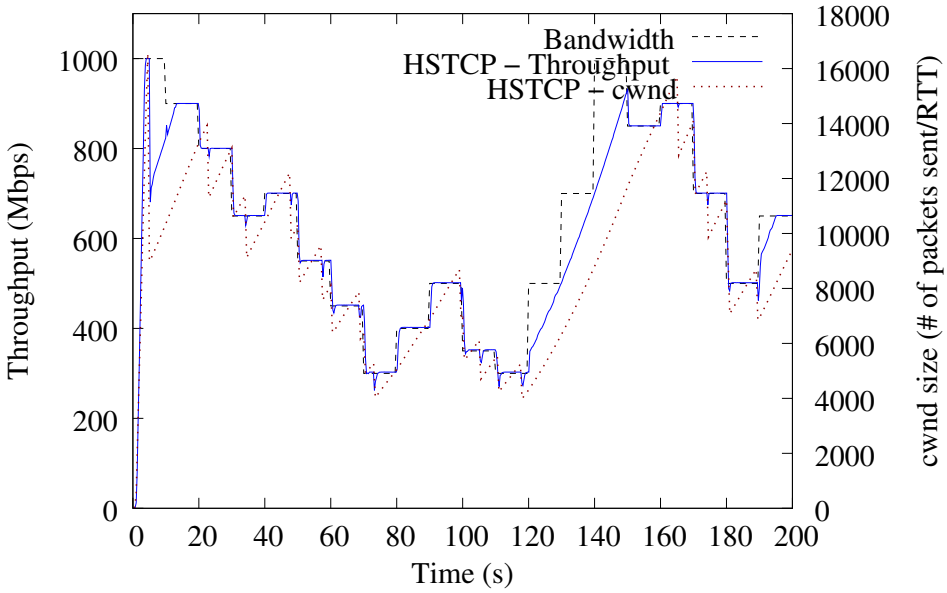


Figure 3.13: HSTCP ($RTT \approx 100ms$) on a step-based VBE.

that when the available bandwidth decreased, the throughput of HSTCP matched very well the available bandwidth function. In the case of step-based bandwidth variation model we observe that every time the available bandwidth decreases, HSTCP describes a behavior like the one shown by Figure 3.14, where four different phases have been perfectly identified.

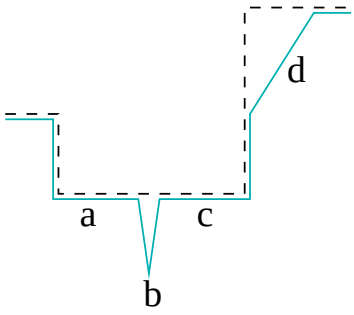


Figure 3.14: Zooming on HSTCP behavior.

The first phase, a, represents the throughput of HSTCP matching perfectly the decreasing available bandwidth. Since the buffers are not full yet, the overload in the bottleneck can be stored. The duration of this phase depends on the size of the routers buffer. The phase b corresponds to the time when the buffers are already full and one (or many) packet has been dropped. HSTCP stops sending data packets, in order to execute Fast Retransmit/Fast Recovery, decreasing significantly the throughput during a very short time (read insignificantly). Note that if the lost packets could not be recovered with Fast Retransmit/Fast Recovery, a timeout could happen. Later, if lost packets are recovered by mean of Fast Retransmit/Fast Recovery and if the congestion window is still large enough to grab all the resources (phase c),

the throughput will go directly to the available bandwidth. This jump does not depend on the RTT value, only on the size of the congestion window after Fast Retransmit/Fast Recovery. Note that after phase c, if the bandwidth remains constant for a long time, phase b could be repeated. Now, if we suppose that the congestion window is small and the routers buffers are empty, HSTCP will take a time to get the resources (phase d). The time needed to take the resources will depend on the RTT value.

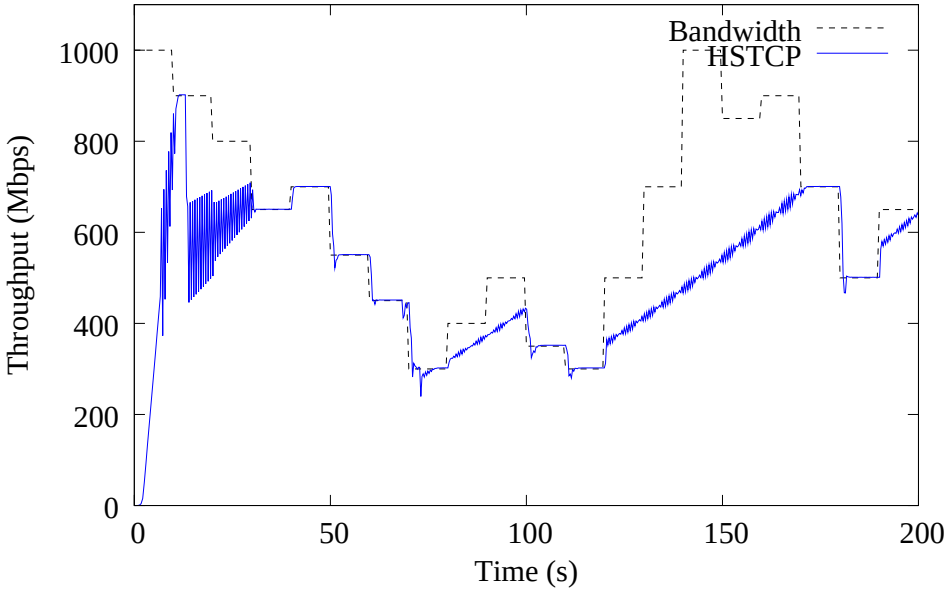


Figure 3.15: HSTCP ($RTT \approx 200ms$) on a step-based bandwidth VBE.

For the second case ($RTT \approx 200ms$), HSTCP struggles to acquire correctly the resources every time that the available bandwidth increases, as we can see in Figure 3.15 where we have plotted the bandwidth evolution (dashed line) and the HSTCP throughput (continuous line). The majors problems of HSTCP appear from second 120 to second 170, where the available bandwidth increases faster. During the period included between seconds 120 and 170, the computed inefficiency was equal to 0.35 (meaning that there were 35% of wasted bandwidth). The inefficiency value shows the negative impact of the increases of the RTT on HSTCP.

After examining the performance of HSTCP on VBEs, we can conclude that HSTCP is not really sensible to the bandwidth variation model. With smaller RTT, HSTCP succeeds in getting a good performance on both sinusoidal-based and step-based bandwidth variation models. However, HSTCP is sensible to bigger RTT (in our case, 200ms approximately). In fact, with bigger RTT values, both sinusoidal-based and step-based bandwidth variation models affect negatively the HSTCP response.

3.2.3 XCP on step-based bandwidth variation environments

The simulation results of XCP on the step-based bandwidth variation model and an $RTT \approx 100ms$ are shown in Figure 3.16, where we have plotted the available bandwidth evolution (dashed line), the throughput (continuous line) and the congestion window (dotted line) functions.

On our step-based bandwidth variation model, with an $RTT \approx 100ms$, XCP shows better performance than HSTCP. Having a look at the congestion window in Figure 3.16, we can see that its size is quickly adapted to the new network conditions. Like in the sinusoidal-based model (i) the throughput of XCP matches very well the available bandwidth evolution, (ii) the congestion window follows very well the changes of the available bandwidth, and (iii) no packets have been dropped during all the period of simulation.

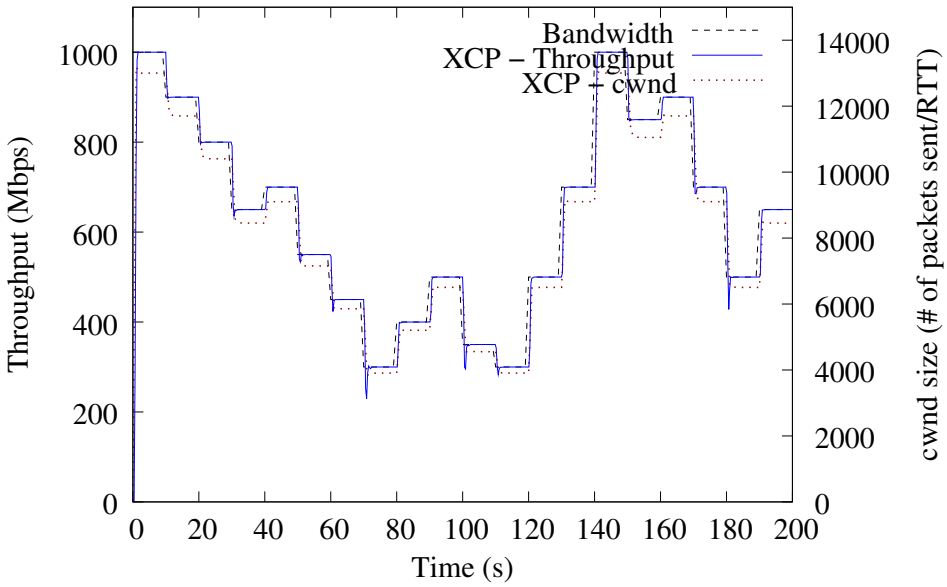


Figure 3.16: XCP ($RTT \approx 100ms$) on a step-based VBE.

Nevertheless, on step-based bandwidth variation model, when the bandwidth decreases by more than 150Mbps in one step, the throughput of XCP displays a strange behavior (see Figure 3.17).

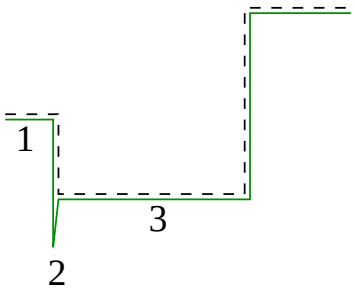


Figure 3.17: Zooming on XCP behavior.

When the throughput of XCP matches perfectly the available bandwidth (step 1) the congestion window stays constant. Later, the available bandwidth decreases because of the aggregation of ON-OFF UDP flows. The overload introduced by new active UDP flows will cause the storage of the XCP packets in the routers buffers, delaying the XCP packets in the path. This delay will decrease the throughput measured by the receiver during a very

short time (step 2). Finally, when the XCP sender adapts its congestion window to the new conditions, the throughput measured by the receiver will become stable (step 3).

The simulation results of XCP concerning the case where the $RTT \approx 200ms$ are shown in Figure 3.18, where we have plotted the available bandwidth (dashed line) evolution and the XCP throughput (continuous line). In this case, the simulation results show that XCP overshoots at seconds 50, 70, 100, 170 and 180. Thus, we can see that the XCP response becomes slower as the feedback delay increases. However, comparing both Figures 3.15 and 3.18, we can see that the increase of the RTT impacts more strongly the responsiveness of HSTCP than the one of XCP. Thus, XCP proves to get higher performance than HSTCP under similar conditions.

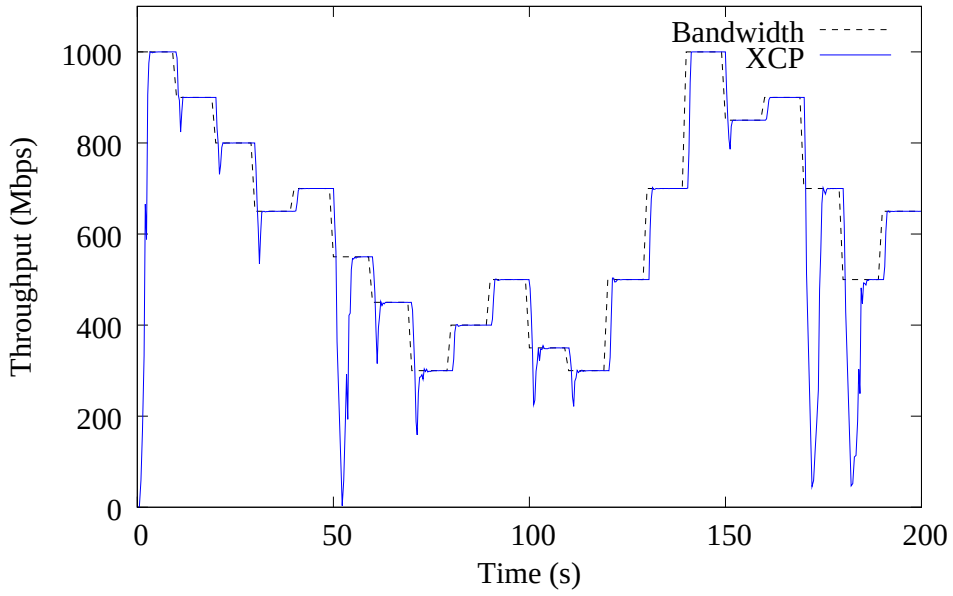


Figure 3.18: XCP ($RTT \approx 200ms$) on a step-based VBE.

3.3 Conclusion and lessons learned

Most studies about Congestion Control Protocols have analyzed and tested different approaches using network models where the available bandwidth remains constant over time. Moreover, most telco-operators or ISPs are currently implementing strong QoS mechanisms to satisfy multiple exigencies of customers running very important applications or Real-Time Multimedia Applications, resulting in VBEs for the best-effort traffic.

Thus, we have studied in this chapter the response and the general performance of three of the most representative congestion control protocols (TCP, HSTCP and XCP) in a continuous bandwidth variation model (inspired from [100]) and a discrete bandwidth variation model.

In order to represent continuous variation model in the ns2 simulator, we used a sine-based function that modifies the available bandwidth as desired. But it is unable to modify the size of the buffer in the routers. However, in real life, flows reserving bandwidth in the links will also take important resources (memory and CPU) in the routers. Therefore, the simulations

results using our sinusoidal-based bandwidth variation model may be different if we add this new assumption in our experiments.

If we want to vary the available bandwidth while the available buffer in routers varies also, we need to produce such variations by the aggregation/disaggregation of simulated highest-priority flows. However, the resulting bandwidth variation model produced by our "highest-priority" flows should describe a discrete function (step-based function), since the granularity of the available bandwidth changes will depend on the rate of flows (from 1MSS/RTT).

Thus, we decided to replace the continuous by a discrete (step-based) bandwidth variation model, where the bandwidth variations are introduced by mean of the aggregation/disaggregation of ON-OFF UDP flows. The benefits of this new model are that we will "reserve" and use the available bandwidth, getting a more realistic network with VBE.

Both models, sinusoidal-based and step-based models, lead us to conclude that:

- Standard TCP, which has been taken as the point of reference since TCP is currently the most widely used congestion control protocol in networks, is not really adaptive to large BDP networks. In fact, the bandwidth variation model has not an important impact over TCP. The TCP mechanisms prevented TCP from achieving good performance on large BDP networks.

- HSTCP, a very promising approach to replace standard TCP on large BDP networks, performs very well under sinusoidal-based and step-based models when RTT is small. However, the increased RTT can degrade the performance of HSTCP under both bandwidth variation models, preventing HSTCP from quickly grabbing the resources when the available bandwidth increases (the increase of the RTT impacts negatively the response of HSTCP in VBE). The problem of HSTCP in VBEs lies on the fact that E2E protocols do not know the state of the network. Instead, E2E protocols must adapt their rate carefully to avoid creating congestion. As a result of this, HSTCP, as well as others TCP-like protocols, are unable to grab and yield the resources quickly on VBEs.

- XCP, the most widely known of the ERN protocols, has proved to get higher performance than standard TCP and HSTCP on both sinusoidal-based and step-based bandwidth variation models whatever the value of the RTT. In our step-based bandwidth variation model, with an $RTT \approx 200ms$, XCP showed some difficulties to adapt the congestion window size to the bandwidth reductions. However, the capacity of XCP to grab the available resources is not degraded in the same proportions as those shown by TCP or HSTCP.

Finally, we want to add that in real networks, there is no pure sinusoidal or step based bandwidth variation models. We believe that in real networks, the bandwidth variation follows a pattern mixing both sinusoidal and step based models, where the granularity of the variations depend on the initial rate and the number of new incoming senders. In such a variation model, HSTCP could suffer losses leading to underutilization of resources. The degree of underutilization will increase as both the RTT and the frequency of the bandwidth variations increase. On the other hand, since the XCP response is weakly impacted by the RTT, XCP would provide the best performance even in highly VBEs.

Chapter 4

Enabling inter-operability between ERN protocols and non-ERN routers

ERN approaches, which are protocols where routers compute the available bandwidth and assign resources to the senders, claim to avoid congestion, to maximize the resources utilization and to fairly share the resources, better than E2E protocols (Chapter 2).

In Chapter 3, by mean of the XCP protocol, which is the most representative of ERN protocols, we have tested and shown the benefits of using an ERN protocol in LDHP networks. Our studies have proved that XCP behaves better than TCP-like protocols, like HSTCP, in LDHP networks even in aggressive environments where the available bandwidth can vary due to the aggregation/disaggregation of flows implementing priority-like or reservation-like mechanisms.

In the experiments shown in Chapter 3, we have tested XCP over networks exclusively of XCP enabled devices (routers and end hosts were XCP-capable elements). However, to benefit from any ERN protocol, as XCP, in heterogeneous LDHP networks, such protocol must be able to cohabit with others protocols already implemented in the networks. For instance, ERN protocols must cohabit with routers without ERN functionalities (classical IP routers like DropTail, RED, etc.), since the use of such routers is very common in current large BDP networks.

Reprogramming every router to install XCP capabilities, or replacing IP routers by new XCP equipments, in large BDP networks may involve stopping the services for a long time (days, weeks or more), which is not realistic.

Thus, in this Chapter 4, we present our studies about the behavior of XCP in presence of routers without XCP functionalities. Mainly, our research activities concern :

1. the detection of the inter-operability problems of XCP with classical IP routers, that can be common to others ERN protocols.

2. the propositions of solutions to those inter-operability problems

3. the evaluation of our propositions, by mean of analysis and simulations on a number of topologies that reflect the various scenarios of incremental deployment on heterogeneous networks.

The works presented here represent the first steps to begin building an ERN protocol that is able to be deployed in heterogeneous large BDP networks.

4.1 Defining the non-XCP router and non-XCP cloud terms

In this chapter, we will need to frequently refer to IP classical routers sets, where the set can be integrated by one or more units. To simplify the presentation of this chapter, we introduce two new useful keywords:

1. **Non-XCP router:** used to designate a traditional IP router, like DropTail, RED, etc. This term refers therefore to a router with no XCP functionalities.

2. **Non-XCP cloud:** used to designate a continuous set of n non-XCP routers, where $n \geq 1$.

4.2 Sensitivity of XCP to non-XCP routers

Since XCP relies on specialized routers to estimate the available bandwidth all along the path, from the source to the destination (Chapter 2, Section 2.5.3), it can easily be foreseen that XCP will behave badly if there are IP classical routers on the path with bottleneck link capacities. Moreover, we can also predict that XCP can perform worse than TCP in this case because the feedback computation will only take into account the XCP elements on the path, ignoring the existence of the bottleneck link. This assumption has been first illustrated in [84]. We review below some simulation results exhibiting this problem for the purpose of making this chapter clearer to the reader.

Figure 4.1 presents 3 scenarios: (a) shows a typical Internet network with non-XCP routers, (b) shows an all-XCP network with 100% XCP-routers and (c) shows a more realistic scenario of an incremental deployment of XCP around a non-XCP router. In all these scenarios, the bottleneck capacity is 30 Mbps while the other links have a capacity of 80 Mbps.

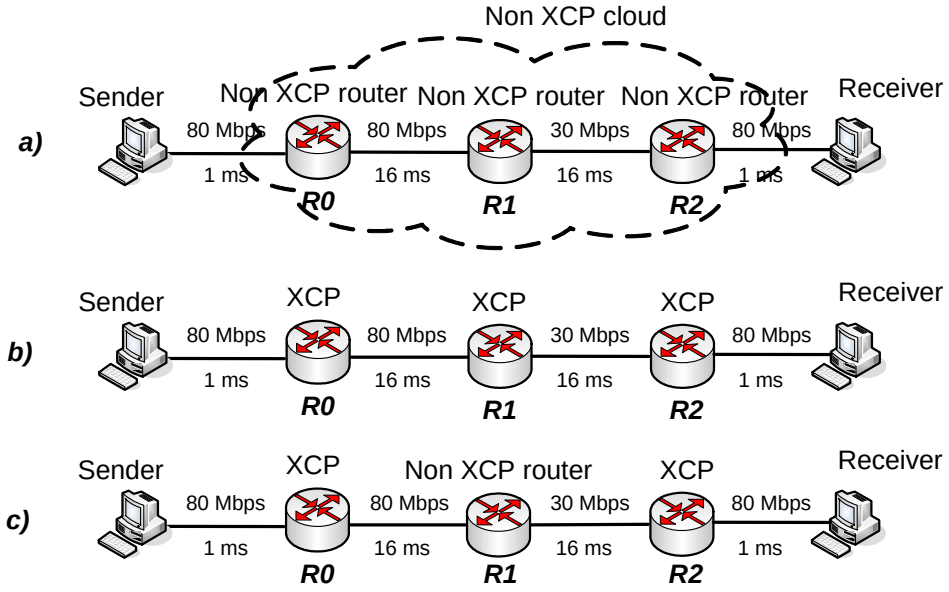


Figure 4.1: (a) scenario for TCP, (b) and (c) scenarios for XCP.

Figures 4.2 shows the throughput and the congestion window evolution of one TCP flow on scenario (a), and one XCP flow on scenarios (b) and (c). The congestion window evolution (Figure 4.2(a)) shows the typical saw-tooth curve of TCP and the typical XCP curve that directly jumps to the optimal congestion window size (no packet losses).

In the first scenario (a) TCP has some problems at the beginning (caused by a very aggressive slow-start). However, at second 10, when the autotuning mechanisms of TCP have found the appropriate values for the network characteristics, TCP becomes stable with minor problems at second 30 at 50. Both problems occur when the congestion window of TCP exceeds the capacity of the routers.

For XCP on scenario (b), the sender gets quickly the available bandwidth and stays stable during all the simulation, like others simulation results given by its authors in [3]. However, in the scenario (c), the congestion window size of the XCP sender is very unstable and frequently goes well beyond the maximum value found by the linear search of TCP congestion avoidance mechanism, causing a high amount of packet losses. The explanation is as follows: since the non-XCP router (R1 on Figure 4.1c) is unable to update the feedback value carried in XCP packets to indicate the bottleneck, the XCP routers ($R0$ and $R2$) that surround the non-XCP router ($R1$) use a feedback value that reflects the available bandwidth outside the non-XCP cloud, which is much greater than the 30 Mbps of the bottleneck in our topology.

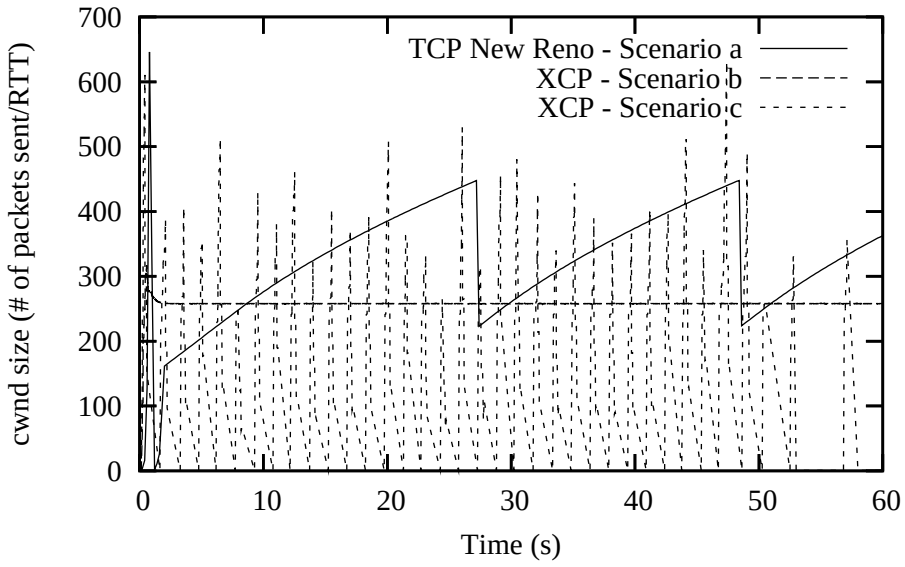
In these *ns*-based simulations, TCP on scenario (a) successfully sent 215.004 MBytes, XCP on scenario (b) sent 223.808 MBytes and XCP on scenario (c) sent only 52.426 MBytes during the one minute experiment.

4.3 Enhancing XCP for heterogeneous internetworking

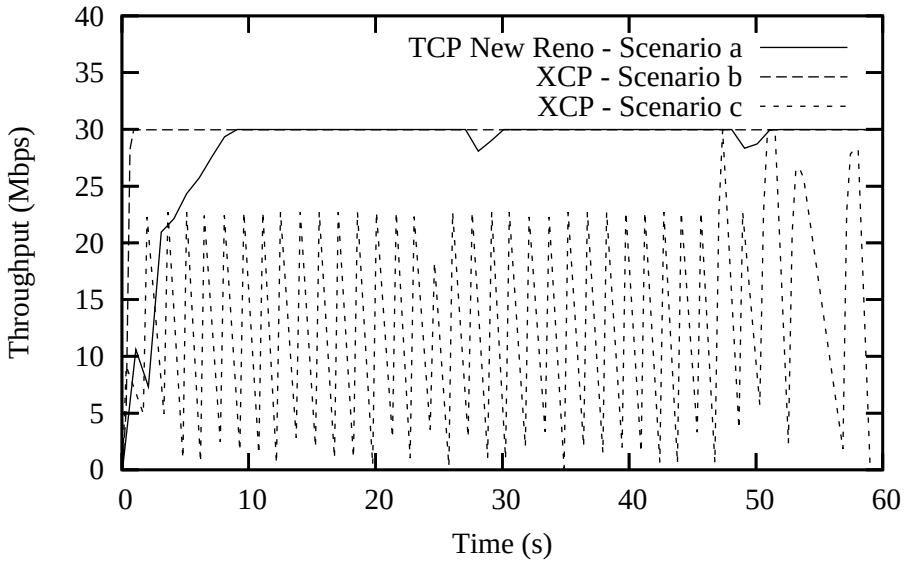
In order to solve the limits of XCP, caused by the presence of non-XCP routers, we propose the architecture of XCP-i routers (the i letter standing for *interoperable*), which are XCP routers with extended capabilities. XCP-i routers enable the inter-operability of XCP with non-XCP routers.

The mechanisms used in our XCP-i routers keep the core of the XCP control laws unchanged, while the changes needed in the XCP routers are minimum. The main reason for doing this is that there are already some XCP implementations available (which have shown that the XCP computations are not trivial to implement [84]) and therefore major changes in the protocol require a lot of time in new software development. Also, XCP-i maintains the XCP philosophy which is to avoid keeping state variables per flow. Finally, since our XCP-i algorithm is independent from the XCP architecture, it is easy to transport the XCP-i mechanisms to any ERN protocol.

The XCP-i algorithm introduces 2 main new functionalities: (i) it detects when an XCP packet has gone through a non-XCP cloud and (ii) it takes into account the available bandwidth in the non-XCP cloud in the feedback computation. In the following sections, we present how these new functionalities have been incorporated into the XCP protocol while keeping the core of the XCP control laws unchanged.



(a) Congestion window evolution



(b) Throughput

Figure 4.2: Congestion window evolution and Throughput for scenarios a,b,c.

4.4 XCP-*i*: architecture and algorithm in routers

4.4.1 Detecting non-XCP clouds

A dual-TTL strategy

XCP-*i* detects non-XCP clouds by using the Time To Live (TTL) counter (defined in [5]). Here, we suppose that all the XCP-*i* routers, and at least one of the routers inside each non-XCP cloud, support the regular TTL operations, especially the one that decreases the TTL value in the IP packet header before forwarding the packet. With this assumption, we add a new field in the XCP packet header (which is different from the IP header) named `xcp_ttl_`. The `xcp_ttl_` will be decremented only by XCP-*i* routers.

In order to discover a non-XCP cloud, TTL and `xcp_ttl_` have to be initialized by the sender with the same value. In this way, on an all-XCP network, the TTL and `xcp_ttl_` fields will always have the same value, because both fields will be decremented by one unit by every XCP-*i* router.

However, when a packet crosses a non-XCP cloud, if at least one of these routers decreases the TTL by one unit, then the XCP-*i* router located after the non-XCP cloud will detect a difference between the TTL and the `xcp_ttl_` fields (TTL < `xcp_ttl_`). Thus, our XCP-*i* router will conclude that the packet has gone through a non-XCP cloud.

After processing a packet, the XCP-*i* router will update `xcp_ttl_` = TTL in order to hide this non-XCP cloud to the others XCP-*i* routers and to detect new non-XCP clouds if they are present in the forwarding path. Figure 4.3 shows graphically how our algorithm works. This solution is easy to implement, does not require any special message between the routers and the overhead for processing this additional field is small.

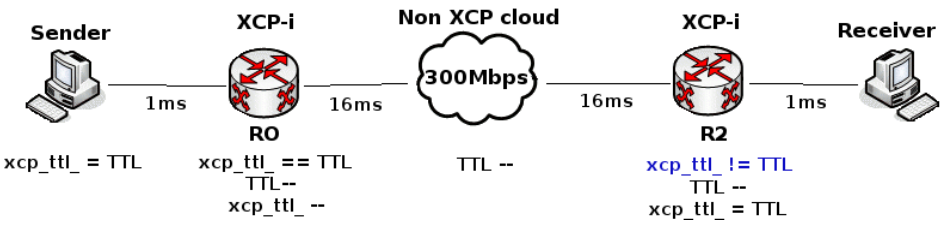


Figure 4.3: Discovering non-XCP clouds in the data path.

When to execute the dual-TTL strategy

Our dual-TTL strategy aims at detecting non-XCP clouds located in the forwarding path of the XCP flows. Non-XCP cloud can be detected easily "on the fly" when a new flow starts, or before by mean of an early non-XCP cloud detection mechanism.

Detecting non-XCP cloud "on the fly"

To detect non-XCP cloud "on the fly" means that non-XCP clouds are detected when a flow starts sending data. When using the "on the fly" detection the SYN packet of a given flow must be used to find the non-XCP clouds not yet detected in the forwarding path. Later,

it is also necessary to delay the transfer of data packets, in order to let XCP-i routers execute completely all their needed mechanisms.

However, the delay introduced in one flow will have a positive effect over the next flows crossing the same non-XCP cloud. In addition, since this thesis is focused on the transfer of large amount of data (like the transfer of Data Bases of several GB), we believe that the delay of the transfer of data, during 1 or 2 seconds, will be compensated by a fast and reliable transfer. Thus, the benefits from the use of XCP-i routers can largely compensate its disadvantages.

Early non-XCP cloud detection

The early non-XCP cloud detection is presented only like a preliminary idea about another way to discover non-XCP clouds. The set of mechanisms that could be needed by this strategy of detection require much more analysis, which is not provided here.

Similarly to the dynamic routing methods, this mechanism, in combination with our dual-TTL strategy to discover non-XCP clouds, could be used by a XCP-i router to discover its XCP-i neighbors and to detect at the same time non-XCP routers.

The idea is to broadcast special packets immediately after an XCP-i equipment is inserted in the network, in order to discover the XCP-i routers and non-XCP clouds neighbors. Thus, when a non-XCP cloud is detected between the new equipment and the discovered XCP-i neighbor, the set of algorithms required by the XCP-i routers can be executed without impacting flows negatively. In addition, when an XCP-i router receives such special messages they do not forward them.

However, it is important to broadcast messages carefully since we could flood the network causing an important overhead. Since this broadcast mechanism can have an important impact in the network, the way that it could be executed by the XCP-i routers (TTL parameters, IP destination, etc), or how it should be forwarded by non-XCP capable routers has to be widely analyzed.

Finally, when a new XCP-i router is installed in the network, the administrator could be aware about the presence of non-XCP enabled devices linked to one of the network interfaces of this new router. In this case, the administrator can manually indicate the presence of non-XCP enabled devices.

Effectiveness of the dual-TTL strategy for detecting non-XCP clouds

In heterogeneous LDHP networks, many mechanisms have been enabled to improve the services and security provided to the customers, such as Multiprotocol Label Switching (MPLS) domains and IPsec tunnels. Since when packets cross these MPLS domains or IPsec tunnels, they are generally not aware about the number of routers crossed, we can think that our dual-TTL strategy will not detect non-XCP routers representing non-XCP clouds. However, below we present arguments showing why our dual-TTL strategy should work successfully even in the presence of MPLS domain or IPsec tunnels.

*XCP-*i *and MPLS domains*

First of all, let a MPLS [101] domain represent a non-XCP cloud, surrounded by XCP-i routers (one XCP-i upstream router and one XCP-i downstream router).

In the case of Uniform Model Label Switched Paths (with or without Penultimate Hop Popping), according to [102], "*when an IP packet is first labeled, the TTL field of the label stack entry (iTTL) MUST BE set to the value of the IP TTL field*". Later, "*when a label is popped, and the resulting label stack is empty, then the value of the IP TTL field SHOULD BE replaced with the outgoing TTL value*" (oTTL is equal to the iTTL less the number of hops within the MPLS domain).

In [102], the authors add also that "*there may be situations where a network administration prefers to decrement the IPv4 TTL by one as it traverses an MPLS domain, instead of decrementing the IPv4 TTL by the number of LSP hops within the domain*".

For the other cases (Short Pipe Model Label Switched Paths and Pipe Model Label Switched Paths), a packet leaving the MPLS domain has a TTL value equal to the IP TTL before the LSP ingress less two units. For more documentation, the reader can refer to [103, 94, 104, 105], between others.

Thus, we can conclude that XCP-*i* routers sending and receiving packets from the MPLS domain will be able to detect the non-XCP cloud represented in this case by the MPLS domain.

*XCP-*i *and IPsec tunneling*

In the case of IPsec tunnels, in [106] we can read that the TTL in the inner header is decremented by the encapsulator before forwarding the packet inside the tunnel, and by the decapsulator if it forwards the packet outside of the tunnel. Thus, the encapsulator modifies the inner IP header only to decrement the TTL. After the modifications executed by the encapsulator, the inner IP header remains unchanged during its delivery to the tunnel exit point.

We can therefore conclude that two XCP-*i* routers sending packets through the IPsec tunnel, will detect the IPsec tunnel, that in this case represents the non-XCP cloud.

There are many others IP tunnels, with equivalent properties of both MPLS domains and IPsec tunnels, already commented here. We will not make an exhaustive study of each kind of tunnels. We think these two specific cases illustrate well why our dual-TTL mechanism should successfully work in most cases.

4.4.2 Detecting the XCP-*i* edge routers

After detecting a non-XCP cloud, XCP-*i* requires the identity of the last XCP-*i* router (the upstream XCP-*i* router). The reasons are that :

- XCP-*i* will try to determine the available bandwidth between both XCP-*i* routers located at the edge of the non-XCP cloud.
- XCP-*i* will create a XCP-*i* virtual router to compute a feedback that will reflect the available bandwidth between both XCP-*i* edge routers.

Note that we will only estimate the available bandwidth inside a non-XCP cloud detected, in order to avoid inserting unnecessary overhead in the remaining parts of the network.

In order to discover the upstream XCP-*i* edge router, we add a new field in the XCP packet header named `last_xcp_router_` which contains the IP address of the last forwarding XCP-*i* node. An XCP-*i* router (and the sender) would simply put its own IP address in this field before sending the packet on the wire. This way, when a non-XCP cloud is detected by

an XCP-i router, this router will automatically know which XCP-i router is located at the other side of the non-XCP cloud.

Once again, this solution is easy to implement, does not require any special message between the XCP-i routers and the CPU usage to process this additional field is kept to a minimum. Figure 4.4 illustrates this algorithm.

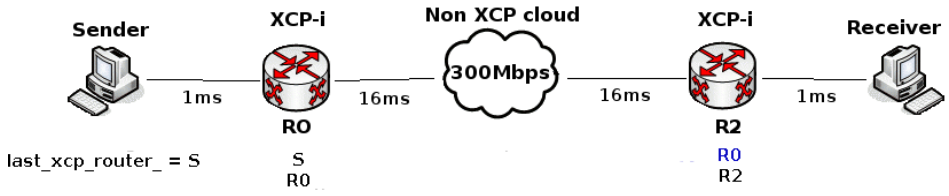


Figure 4.4: Discovering XCP-i edge routers.

4.4.3 Estimating the available resources into the non-XCP cloud

Communicating XCP-i edges routers to compute the available bandwidth inside a non-XCP cloud

Let's suppose that XCP-i_{k-1} and XCP-i_k are the upstream and the downstream routers respectively, located at the edge of the non-XCP cloud already discovered. The idea in the XCP-i algorithm is to initiate a bandwidth estimation procedure at XCP-i_{k-1} when a request is received from XCP-i_k. In the next paragraphs, we explain in more details the executed steps to estimate the available bandwidth between XCP-i_{k-1} and XCP-i_k.

After a non-XCP cloud has been detected by XCP-i_k and after that XCP-i_k knows the identity of the last XCP-i router (XCP-i_{k-1}), XCP-i_k must send an available bandwidth estimation request to XCP-i_{k-1} and must wait for an ACK during a $xcp_req_timeout$ period. If this ACK does not arrive the process is restarted. After a new unsuccessful request, XCP-i_k concludes that the path between XCP-i_{k-1} and XCP-i_k is broken. The bandwidth estimation request will only be resent on reception of a new packet from XCP-i_{k-1}. Now, upon reception of a request, XCP-i_{k-1} will acknowledge the request and will try to find the available bandwidth, $ABW_{k-1,k}$, between XCP-i_{k-1} and XCP-i_k. We will not go into details of which algorithm must be used to compute the available bandwidth, and we will only suppose that the router will implement one of these to find the most accurate value.

After having obtained $ABW_{k-1,k}$, XCP-i_{k-1} will send the estimated available bandwidth value to XCP-i_k, which will add an entry in a hash table based on the IP address of XCP-i_{k-1} and the sum of $ABW_{k-1,k}$ and the input traffic rate[1] computed from XCP-i_{k-1}. Later on, the bandwidth estimation procedure should be performed periodically (for instance, 1 second after finishing a bandwidth estimation procedure). Thus, for every available bandwidth estimation received from XCP-i_{k-1}, XCP-i_k will update its hash table using the next equation:

$$BW_{k-1,k} = I_{k-1} + new_ABW_{k-1,k} \tag{4.1}$$

where I_{k-1} is the current input traffic rate received from the XCP-i_{k-1} router.

[1] the XCP traffic since we are not able to know whether TCP traffic comes from an XCP-i router

The available bandwidth estimation should be stopped after a few minutes of inactivity of XCP-i_{k-1}. The timeout before stopping the available bandwidth estimations must be defined by the router manager[2]. This action aims at decreasing the load introduced in the network. To do this, XCP-i_k must send a message to XCP-i_{k-1}, signaling the end of the bandwidth estimation operations. At the same time, XCP-i_k should remove XCP-i_{k-1} from the hash table, in order to keep the hash table as small as possible

Even if XCP-i_{k-1} computes the available bandwidth $BW_{k-1,k}$, it is very important that XCP-i_k stores the available bandwidth (and therefore performs the feedback computation as this will be explained in the next section), since XCP-i_{k-1} does not know the final destination of flows crossing the same non-XCP cloud.

Finally, it is important to take into account the fact that the available bandwidth as it is estimated could not be sent to the sender, because the sender will not be able to share the resources fairly with other active flows crossing the same non-XCP cloud. Actually, protocols like TCP Westwood which calculate the available bandwidth between end hosts to know the best sending rate, still need the TCP mechanisms to ensure fairness between active flows.

This solution does not keep any state per flow: only 1 entry in the hash table needs to be kept per upstream XCP-i router neighbor.

Viability of using Available bandwidth estimation techniques in XCP-i routers

We have proposed to use one of the available bandwidth estimation techniques to compute the resources into the non-XCP cloud, and we said that we are not proposing a special method to execute this work. The study of available bandwidth estimation algorithms covers a large domain out of the scope of this thesis.

However, the success of XCP-i lies, in a great extent, on the accurate results that one of the available bandwidth techniques could give to our XCP-i routers. For this reason, we have analyzed some studies concerning the performance and comparison of the available bandwidth estimation tools, where we found some results that encourage the idea of using available bandwidth estimation tools into the XCP-i routers.

First of all, in [107] the authors present a very complete work of comparison of Public E2E estimation tools on large BDP networks. In this work, they reported that *pathChirp* [108] or *Pathload* [109] present very accurate bandwidth estimations. In terms of traffic probing, *pathChirp* had a very low overhead, consuming only between ∼0.01% and ∼0.2% of the total available bandwidth on a GigE link, and introducing practically no additional traffic into the network as it measures. *Pathload* introduced between 3 and 7%. Finally, *pathChirp* converged on average in 5.4s, and *Pathload* from 7.2 to 22.3s.

In [110], the authors present a mix of two packet pair techniques to characterize the available bandwidth on a network path: Initial Gap Increasing (IGI) and Packet Transmission Rate (PTR). The experiments using *IGI/PTR* were made in an Internet path and the results show that *IGI/PTR* can estimate the available bandwidth with a high accuracy level (in average less than 20% of error), fast (∼1s) while the overhead is low (6 rounds of a packet train composed from 16 to 64 packets of 500B or 700B). Tests using the *IGI/PTR* tool[3] in our LAN confirmed the results given by the authors.

[2]More studies are needed to get upper and lower bounds balancing performance of flows and overhead in the network

[3]available at `http://www.cs.cmu.edu/~hnn/igi/`

Another promising available bandwidth estimation technique is *QuickProbe* [111]. The authors of *QuickProbe* claim that *QuickProbe* is accurate (estimation errors smaller than 30% of the available bandwidth), fast (between 3 and 5 RTTs) and no intrusive (19 probe packets) when estimating the available bandwidth.

The results given in [107, 110, 111] show that:

1. the overhead introduced by the available bandwidth estimation algorithms would be low. Thus, if *pathChirp* is used by 1000 XCP-i to estimate the available bandwidth inside of the same bottleneck, then the overhead introduced should be equal to 10% of the total available bandwidth (XCP-i should be scalable).

2. the available bandwidth would be estimated relatively fast: *IGI/PTR* computes the available bandwidth in 1s approximately.

3. the available bandwidth estimation algorithms would be accurate: the average error for *IGI/PTR* is approximately 20% and smaller than 30% for *QuickProbe*.

Finally, note that we have proposed to estimate the available bandwidth between edge XCP-i routers of a non-XCP cloud, which has the benefits of improving the time and the accuracy of the estimations procedures. In fact, the accuracy of the available bandwidth estimation improves as the path becomes shorter, while the time needed for estimations decreases.

4.4.4 The XCP-i virtual router

When an XCP-i router, R_{XCP-i}, receives a packet that has gone through a non-XCP cloud, and if an available entry BW exists in the hash table for `last_xcp_router_`, R_{XCP-i} will use a virtual router, XCP-iv, to compute a feedback that will reflect the network condition in the non-XCP cloud. The purpose of the virtual router is to simulate an XCP-i router, located downstream from `last_xcp_router_` with a virtual output link connected to R_{XCP-i}, and which capacity is the estimated available bandwidth in the non-XCP cloud. Figure 4.5 shows the logical architecture of the R_{XCP-i} router with one virtual router per non-XCP cloud. We can consider the virtual router as a logical entity that replaces the non-XCP cloud.

In Figure 4.5, we can see that one non-XCP cloud can generate more than one virtual router. The number of virtual routers depends on the number of couples of upstream-downstream routers formed by the edge XCP-i routers surrounding such a non-XCP cloud.

To compute the feedback corresponding to the state of the network into the non-XCP cloud, a virtual XCP-i router re-use the XCP code available inside the XCP-i router, where the virtual router has been created. Thus, the feedback equation used by the virtual XCP-i router must be based on the XCP feedback equation (Equation 2.30 from Chapter 2, Section 2.5.3):

$$\phi = \alpha.rtt.(O - I) - \beta.Q$$

where $\alpha = 0.4$, $\beta = 0.226$, rtt is the average RTT of all the flows crossing this XCP router, O the output link capacity, I the input traffic rate and Q the persistent queue size.

However, a virtual XCP-i router has neither its own output/input physical link (O and I are unknown), nor its own buffer (Q is unknown also). Therefore, minor changes are necessary in the feedback equation of XCP as explained below.

Since the hash table is updated as the sum of the estimated available bandwidth and the detected input traffic rate from the upstream XCP-i router (Equation 4.1), the value that we

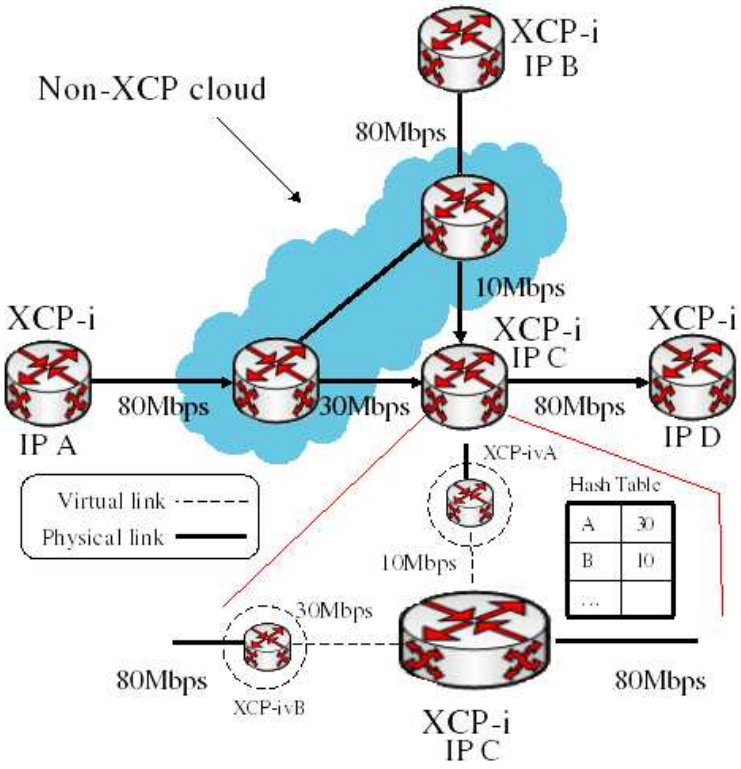


Figure 4.5: An XCP-i router hosting 2 virtual routers.

are obtaining represents actually the virtual output link capacity O for the virtual router. In addition, a virtual router is able to compute the XCP input traffic rate from the upstream XCP-i router, which represents the input traffic rate I_v for our virtual router. Concerning the Q parameter, since our virtual XCP-i router does not have its own buffer, $Q_v = 0$, then $\beta.Q = 0$. The feedback equation for our virtual XCP-i router can be written as

$$\phi_v = \alpha.rtt.(BW - I_v) \tag{4.2}$$

Once the feedback is updated by the virtual router, XCP-i will start its normal feedback computation as usual.

We believe that our solution based on the creation of virtual XCP-i routers is scalable since the hash table used by the virtual routers contains only one state per upstream router. In addition, to calculate a general feedback (ϕ_v), the virtual routers must only access the hash table and then reuse the code already available in the real routers (once every control interval). Note also that most operations needed by every virtual router can be executed in parallel.

4.5 XCP-i : architecture in end-hosts

It is possible that during an incremental deployment of XCP, either the source or the receiver, or both, are not directly connected to an XCP router. For example, Figure 4.6 shows an asymmetric deployment scenario where XCP-i routers are deployed near the receiver side with a non-XCP cloud at the sender side.

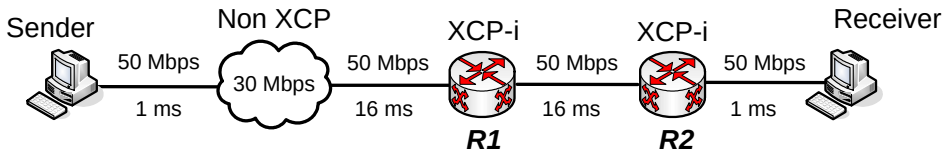


Figure 4.6: Asymmetric deployment: optimized receiver side

In these cases, some parts of the XCP-i algorithm must also be supported by the end-hosts. If the XCP-i router is located at the receiver side (Figure 4.6), the sender must be able to initiate a bandwidth estimation procedure upon reception of a request from the first XCP-i on the path. When the XCP-i router is located at the sender side, the receiver can either act as an XCP-i router by implementing both non-XCP cloud detection and feedback computation. These solutions have the benefit of simply duplicating the XCP-i router code in the end hosts.

4.6 Simulation results

XCP-i has been simulated with ns, by extending Katabi's XCP code where we have incorporated the enhancements of XCP-i, on a wide number of topologies that reflect the various scenarios that we could find when deploying gradually XCP-i on heterogeneous LDHP networks. Thus, in this section, we present a first set of simulations that will let us analyze the

performance and fairness of the XCP-i virtual routers. Later on, in a second set of simulations, we will analyze the impact of the available bandwidth estimation accuracy in XCP-i.

It is important to note that in all our simulations we have supposed that the virtual routers "replacing" the non-XCP clouds already exist when XCP flows are executed.

4.6.1 Performance and Fairness in virtual XCP-i virtual routers

In the simulations presented along this section, the virtual XCP-i routers always calculate feedbacks from an accurate available bandwidth value of the non-XCP clouds. By supposing that we have always the correct available bandwidth value of the non-XCP cloud, we aimed at testing whether our virtual XCP-i routers are able to ensure both good fairness and good performance by only reusing the XCP code from the real router. In *ns*, the available bandwidth is found easily by subtracting the incoming traffic load to the bottleneck link capacity, which are known in the simulation.

We want to remark that we will test XCP-i in several topologies, which reflect many of the scenarios that can be found whether XCP-i is deployed gradually in heterogeneous large BDP networks.

4.6.1.1 Incremental deployment around non-XCP clouds

The first scenario, on which XCP-i is tested, consists of a symmetric incremental deployment depicted in Figure 4.7, which could be viewed as an optimized peering point scenario where two non-XCP clouds are located along the forwarding path of an XCP flow.

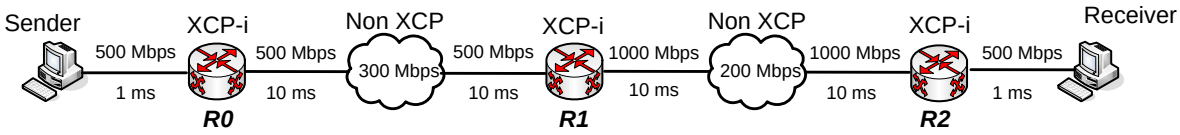


Figure 4.7: Incremental deployment at peering point

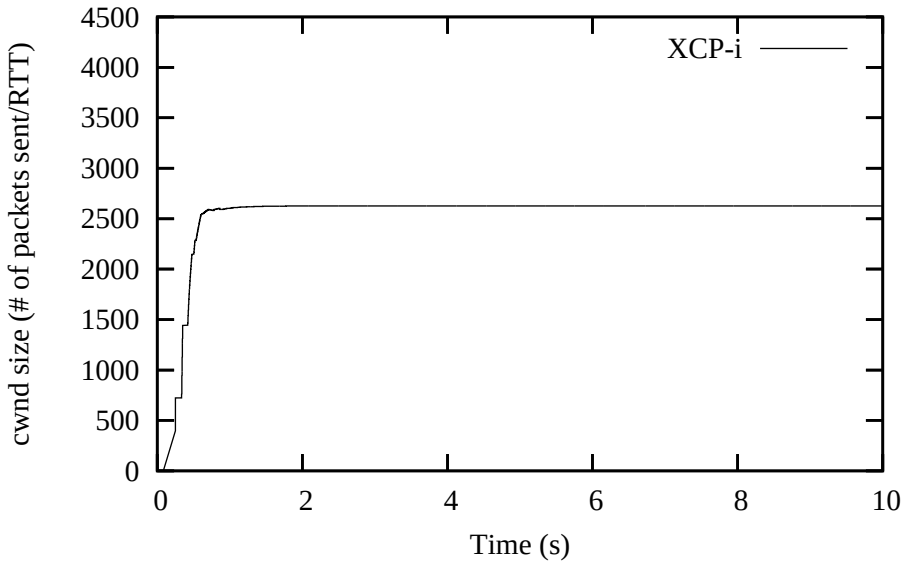
Figures 4.8(a) and 4.8(b) show the *cwnd* evolution of the sender and the throughput seen by the receiver respectively. As we can see, both *cwnd* and throughput are stable with identical results when compared to the all-XCP scenario. Although not shown, there were no timeouts nor packet losses during this simulation. The XCP-i virtual routers in R1 and R2 are able to compute accurate feedback value according to the networks conditions of the non-XCP clouds. These results show that XCP-i is able to efficiently run in an heterogeneous network even though it is deployed only at some strategic locations.

4.6.1.2 Merge scenario: n non-XCP clouds share one XCP path

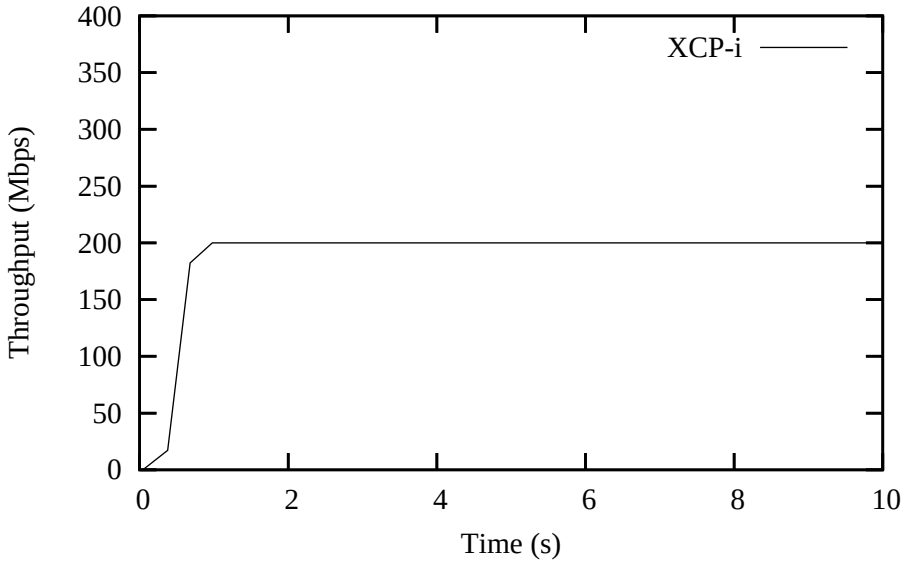
Case where Σ***input_capacity*** $=$ ***output_capacity***

The second topology is a merge scenario where 2 non-XCP clouds share 1 XCP path as depicted in Figure 4.9.

In this case, the XCP-i router at the merging point (R1 from Figure 4.9) has to create one virtual XCP-i router for each incoming non-XCP cloud. In addition, the sum of the bottleneck



(a) Congestion window evolution



(b) Throughput

Figure 4.8: XCP-*i* in the topology shown by Figure 4.7.

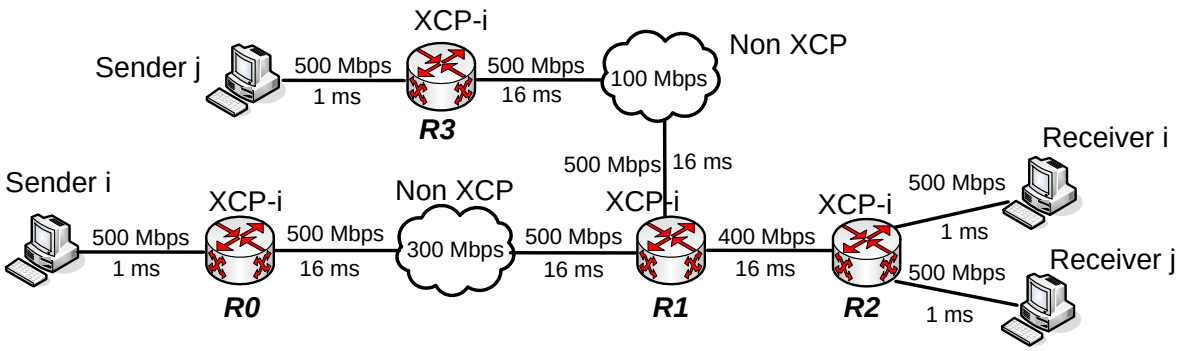


Figure 4.9: 2 upstream non-XCP queues, $\Sigma input_capacity = output_capacity$

bandwidth of each non-XCP cloud is equal to the output link capacity of the XCP-i merging point. Thus, we aimed at proving that the the real router R1 does not interfere in the feedback computation of virtual routers when congestion does not occur.

The simulation results in Figure 4.10 show that XCP-i virtual routers in the R1 router act as independent entities, since sender j can get an optimal throughput of 100 Mbps while sender i can get approximately 280 Mbps. The reason why sender i only gets 280 Mbps instead of 300 Mbps is due to XCP control laws and is explained in more details in [83]. Therefore, this problem is not produced in any case by our XCP-i solution.

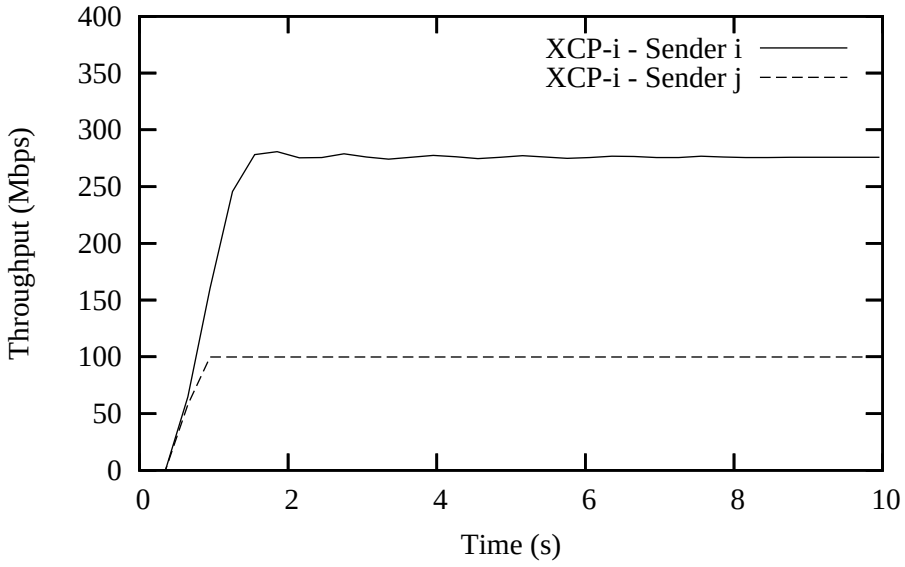


Figure 4.10: Throughput of XCP-i on Topology of Fig. 4.9

Case where $\Sigma input_capacity > output_capacity$

Figure 4.11 shows a more complex scenario where we have 2 non-XCP clouds and 1 path

exclusively composed of XCP equipments. All share a single XCP-i router (R2 router). In addition, the non-XCP cloud on the top carries 2 flows, j_0 and j_1, which should share an available bandwidth of 100 Mbps. Also, if we consider the sum of all incoming links at the XCP-i merging point, it is much higher than the output link capacity. In this scenario we test the ability of XCP-i to correctly use the legacy XCP feedback computation procedure to insure fairness between all merging flows.

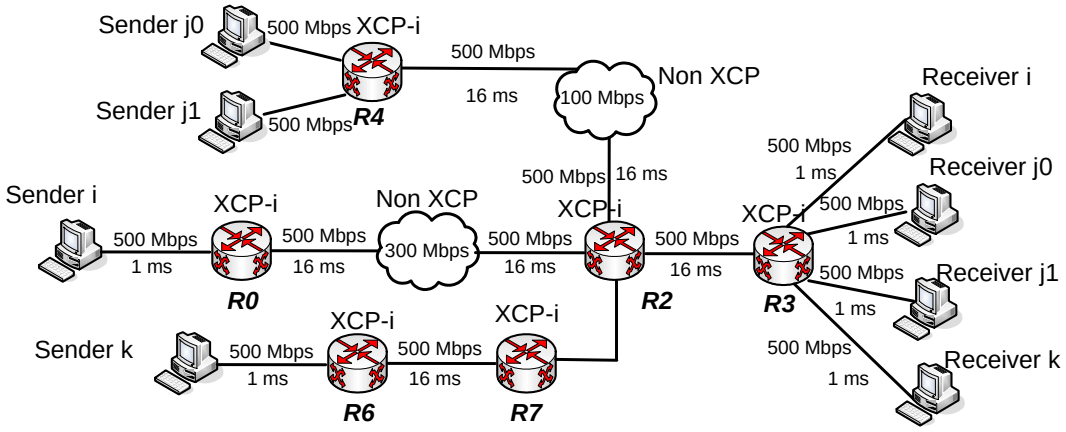


Figure 4.11: 2 upstream non-XCP queues competing with an XCP path, $\Sigma input_capacity > output_capacity$

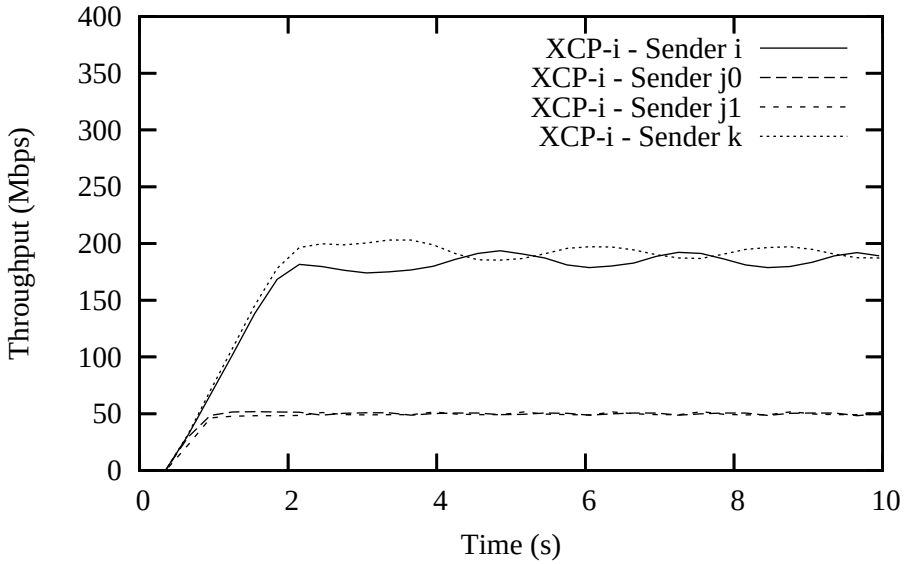


Figure 4.12: Throughput of XCP-i on Topology of Fig. 4.11

The simulation results for the topology shown in Figure 4.11 can be found in Figure 4.12, where we have plotted the throughput evolution seen by receivers. As we can see in Figure 4.12, XCP-i succeeds in getting a good performance even in complex scenarios. Thus, the

real XCP-i router R2 insures fairness while avoiding congestion. In parallel, the virtual router located in R2 which manages the resources of the non-XCP cloud with 100 Mbps, successfully shares the available resources, limiting senders j_0 and j_1 at 50 Mbps each.

Figure 4.12 allows us to observe the impact of the virtual XCP-i routers on the performance of flows. Comparing the throughput of the senders k and i, we can observe that the flow following the path composed of XCP-only equipments (k flow) gets a throughput slightly higher than the flow crossing a non-XCP cloud (i flow). We believe that this difference in the throughput is caused by the additional delay introduced by our XCP-i virtual router.

4.6.1.3 Fork scenario: one non-XCP cloud serves n XCP paths

Figure 4.13 shows a topology with a non-XCP cloud connected to 2 XCP paths. In this topology, the path of the sender i, as well as the path of the sender j have an available bandwidth of 300 Mbps. However, since the bottleneck is located in the R0 router with a link capacity of 500Mbps, both senders i and j only get a throughput of 250Mbps, as we can see in Figure 4.14.

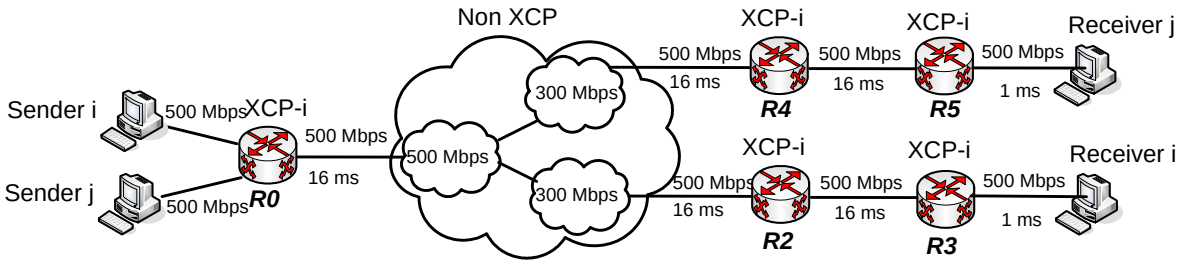


Figure 4.13: 1 non-XCP queue shared by XCP-capable downstream nodes

4.6.2 Impact of the bandwidth estimation accuracy on XCP-i

We supposed so far that the bandwidth estimation found by the routers are always accurate. This is not always true [107] and under certain conditions, the tools used to estimate the available bandwidth could overestimate or underestimate it. In current subsection, we use again the topology described in Figure 4.9 and we suppose that the available bandwidth estimation is inaccurate: we randomly overestimate or underestimate the available bandwidth by a maximum of 20% and a minimum of 5% (20% was the error shown by *IGI/PTR* on an Internet path in [110] and less than 5% was the error shown by *pathChirp* in a testbed in [107]). In addition, routers inside the non-XCP cloud with an available bandwidth of 100Mbps and 300Mbps were initialized with a buffer capacity of 1200 and 3500 MSS respectively (capacities slightly smaller than the BDP value).

In this specific case, we execute the available bandwidth estimation approximately every 2 seconds. In fact, considering Subsection 4.4.3, if we implement an estimation bandwidth mechanism like *IGI/PTR*, we could estimate the available bandwidth in 1s and then wait 1s before the next estimation.

In Figure 4.15, we present first the throughout of TCP, where we can see that TCP is not able to get all the available bandwidth (bottleneck link capacities are 300Mbps and 100Mbps) and sender i and j sent respectively 329Mbytes and 172MBytes in 20s. In fact, we already saw

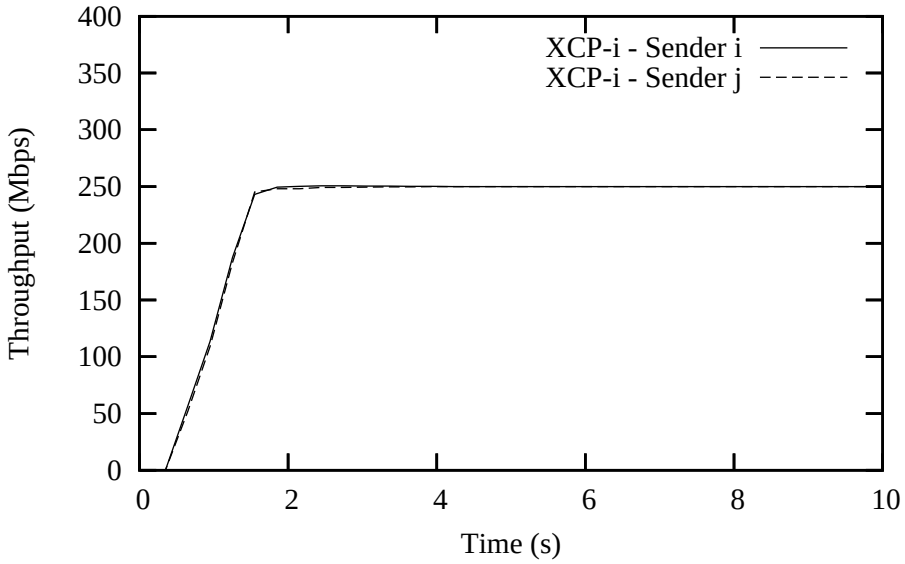


Figure 4.14: Throughput of XCP on Topology of Fig. 4.13

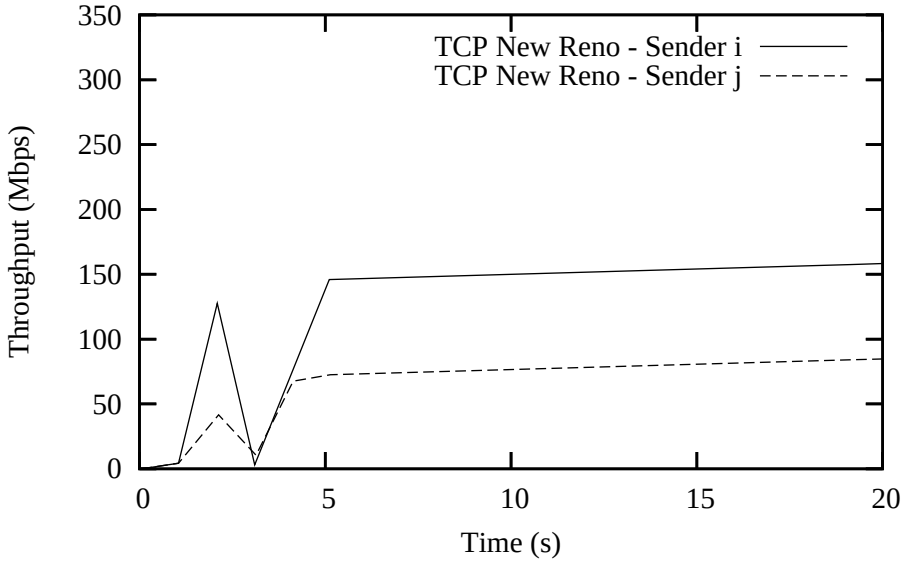


Figure 4.15: Throughput of TCP on Topology of Fig. 4.9

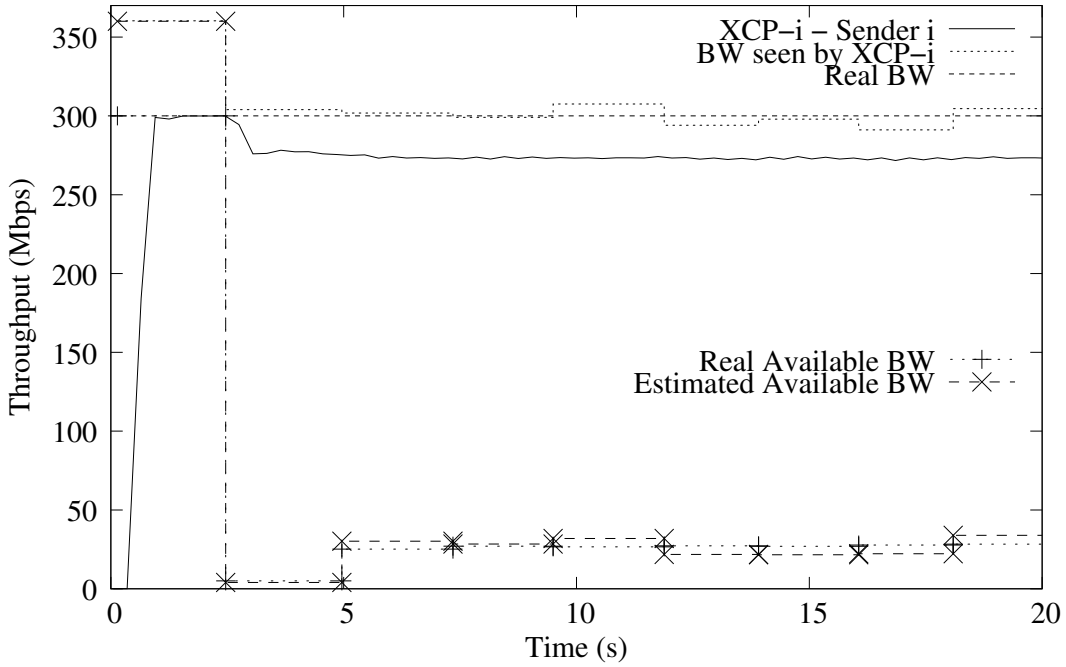


Figure 4.16: XCP-i - Sender i - max error 20% - min error 5%

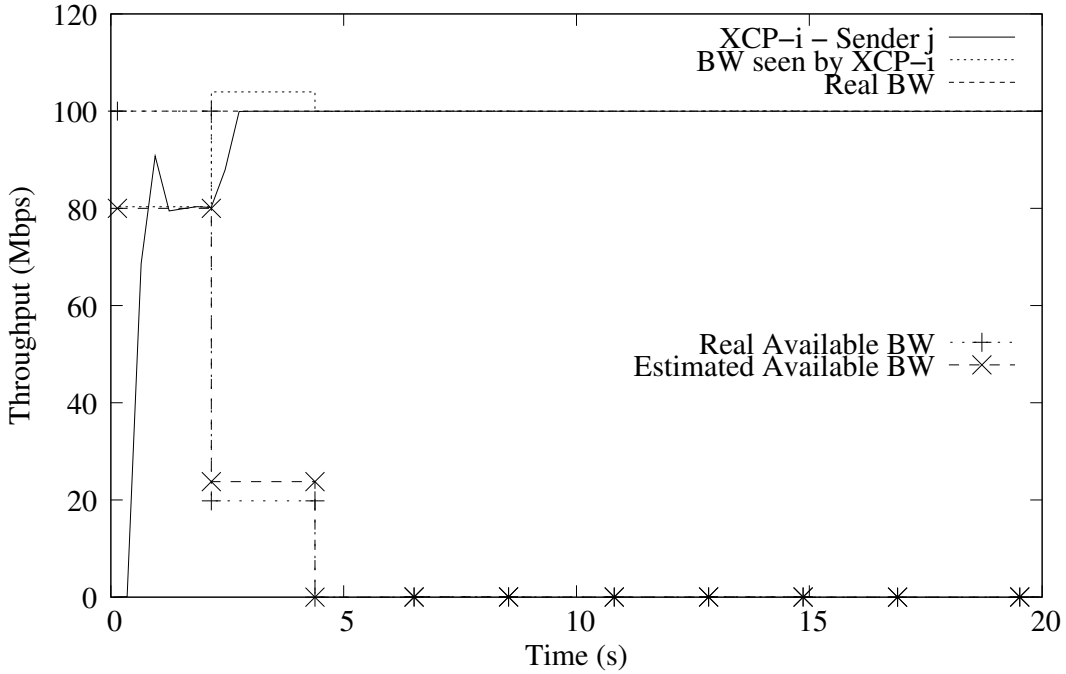


Figure 4.17: XCP-i - Sender j - max error 20% - min error 5%

in Chapter 3 that TCP does not perform well in LDHP networks. XCP-i with an estimation error between 5% and 20% still performs well: senders i and j sent respectively 675.4MBytes and 240.7MBytes.

The main consequences of the available bandwidth overestimations should be packet drops and timeouts. In the case of the sender i (Figure 4.16), overestimating by 20% the resources of the non-XCP cloud located between routers $R0$ and $R1$, during the seconds 0 and $\sim$2.5, was not enough to produce neither dropped packets nor timeouts. In fact, between seconds 0 and $\sim$2.5 the overestimation improved the performance of XCP, letting the sender i grab all the resources (contrary to the case shown in Figure 4.10 where the XCP fairness mechanism prevents sender i grabbing all the resources). In addition, the overload sent by the sender i was successfully stored in the routers of the non-XCP cloud. Thus, the impact of the overestimation depends directly on the routers capacity located in the non-XCP cloud to store the overload. Note also that, as new estimation were executed, the error on the estimation of the available bandwidth between routers $R0$ and $R1$ decreased.

In parallel, sender j shows clearly that the underestimation could be translated in underutilization of the resources of the network (Figure 4.17). Between seconds 0 and $\sim$2, underestimating by 20% the capacity of the non-XCP cloud located between routers $R3$ and $R1$, resulted in a throughput of 80Mbps for sender j. The wasted available bandwidth between seconds 0 and $\sim$2 was equal to 20Mbps. In this case also, as new available bandwidth estimations were executed, the error on the estimations decreased.

As we remarked at the end of the last two paragraphs, multiples available bandwidth estimations tend to improve the accuracy of the estimations. This phenomenon is caused by the fact that the impact of the overestimation and underestimations decreases as the available bandwidth decreases. For instance, in a network where the bottleneck capacity is limited to 1Gbps and where the total available bandwidth is 300Mbps, an overestimation of 20% (equal to 60Mbps) results in 360Mbps. However, when the available bandwidth will be only 10Mbps, we will need an overestimation of 100% to get 20Mbps. Thus, 60Mbps for the first case is much higher than 10Mbps for the second case.

4.7 Open issues in our XCP-i approach

The topology depicted by Figure 4.18 is currently not fully supported. In the case where a bottleneck link in an non-XCP cloud is shared by 2 XCP paths two problems could happen. The first problem is as follows: let assume in a first step that all links are unload. If router b detects the non-XCP cloud, it will request a bandwidth estimation procedure from router a. The result will be $BW = 300Mbps$ if the link is not loaded. Now, suppose that almost at the same time router c also detects the non-XCP cloud and requests a bandwidth estimation procedure from router d. Again, $BW = 300Mbps$. Then senders i and j will both try to transmit at 300 Mbps resulting in a 600 Mbps load for the bottleneck link, and probably causing losses. This problem will not occur when the bandwidth estimations will be unsynchronized.

However, unsynchronized bandwidth estimations can be the origin of the second problem of senders sharing the bottleneck. By re-using the example given in the last paragraph, when a first bandwidth estimation is executed, it will estimate a value near to 300Mbps. Thus, the first sender will take the estimated bandwidth. When another estimation will be triggered from a different XCP-i router, the available bandwidth estimated will be 300Mbps less the value of the first estimated bandwidth, that can be a value near to zero.

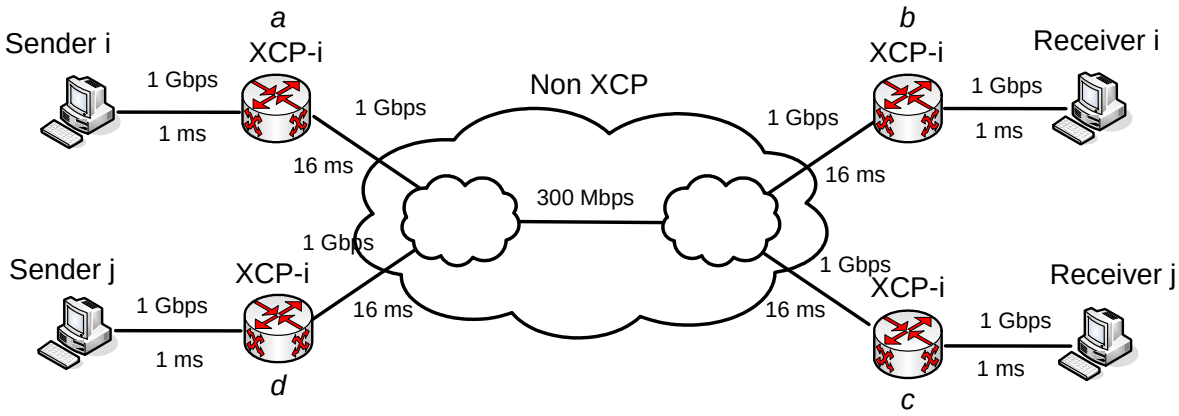


Figure 4.18: 1 bottleneck link shared by n XCP paths

4.8 Conclusion

In this chapter we analyzed the performance of XCP, when a XCP flow crosses XCP and non-XCP routers.

First of all, we proved that when the bottleneck of a XCP flow is composed by non-XCP routers, this XCP flow is not able to know the capacity of the bottleneck, resulting in an overload that could produce many dropped packets and timeouts (Figure 4.2). The problem described is this paragraph prevents an incremental deployment of XCP in current heterogeneous LDHP networks.

To tackle this problem, we designed a new XCP version that we have called XCP-i (XCP inter-operable). XCP-i implements the following mechanisms to decreases the impact of non-XCP routers in the path of the flows:

1. Discovering of non-XCP routers in the path of a XCP flow: this mechanism is implemented by mean of a dual-TTL strategy. In Section 4.4.1 we have shown why the dual-TTL strategy can successfully detect non-XCP routers, even in presence of various mechanisms currently deployed in heterogeneous networks, like MPLS domains and IPsec tunnels.

2. Computing the available bandwidth into the non-XCP cloud: even if this step is based on the use of an available bandwidth estimation tool, to compute the available resources into the non-XCP cloud, we did not suggest the use of any specific tool currently known. We only presented the characteristics of some of the proposed available bandwidth estimation tools (*IGI/PTR* and *QuickProbe* more specifically), as well as arguments to show why some bandwidth estimation tools can provide information about the network state, according to the requirements of our XCP-i solution and the scenario in which we are interested: long-life flows in large BDP networks.

3. Creation of a virtual XCP-i router: we proposed the use of a virtual XCP-i router to compute a new feedback reflecting the estimated available bandwidth. Since our virtual XCP-i router reuses the XCP code already available in the real XCP-i routers

to compute the feedback, virtual routers avoid introducing new complex mechanisms to the XCP protocol stack in real routers.

Various simulations in Section 4.6.1 show that XCP-i improves the behavior of XCP flows when non-XCP routers are located in the bottleneck. Thus, XCP-i is able to maximize resources utilization while keeping high fairness in a wide range of topologies encountered if XCP-i is incrementally deployed in heterogeneous large BDP networks.

However, in section 4.6.2 we have also seen that the inaccuracy in the available bandwidth estimation can impact negatively the performance of flows. Overestimations can be translated in dropped packets causing timeouts, and underestimation in underutilization of available resources. Nevertheless, the impact of overestimations and underestimations are reduced as the available bandwidth decreases. Thus, when XCP flows get resources from the non-XCP clouds, multiple bandwidth estimations tend to improve the accuracy of measures.

Some open issues need to be more widely explored, as shown in Section 4.7. We believe that the impact of the problems introduced by the topology shown in the Open Issues Section can be decreased with very frequent bandwidth estimations. However, more studies are needed to know the impact of the overhead introduced by such a solution.

XCP-i represents the first steps towards an ERN protocol able to be deployed in heterogeneous large BDP networks. Supposing now that an ERN protocol is used in heterogeneous large BDP networks, then the ERN protocol could suffer some losses caused by other mechanisms and congestion control protocols running into the networks. If the losses suffered by ERN protocols are packets containing the state of the network (ACK generally), senders using the ERN protocol could experience problems. Thus, in the next chapter we study the impact of feedback losses in an ERN protocol like XCP.

Chapter 5

Improving robustness of ERN protocols against feedback losses

In Chapter 4, we proposed a set of mechanisms and algorithms to enable the inter-operability of ERN protocols with IP classical routers. Our propositions, tested by extending the routers XCP code, keep the high performance and high fairness of long-life ERN flows in a wide range of topologies. Such topologies were specially designed to reflect the various scenarios that we can find during the incremental deployment of ERN protocols in heterogeneous large BDP networks.

However, the aggressiveness and the burstiness nature of uncontrolled flows (like flows using any E2E congestion control protocol) can produce losses in heterogeneous networks. Thus, flows using any ERN protocol can suffer from these losses of data and/or ACK packets. Since the feedback provided by ERN routers to the senders are generally inserted in ACKs, the losses of ACKs imply losses of feedbacks.

For this reason, we present in the current chapter our studies concerning the impact of feedback losses on ERN protocols. These studies have been made by analyzing the XCP protocol under ACK losses, since in XCP, the feedbacks are provided to the senders within ACK packets.

Some studies about XCP under ACK losses already exist [84]. These studies show that XCP performs well even under ACK losses, in networks where the available resources (like bandwidth) remain unchanged over time. But, as we demonstrate in Chapter 3, in heterogeneous LDHP networks the available bandwidth can fluctuate due to the deployment of QoS mechanisms (reservation-like or priority-like mechanisms) to satisfy customers requests and due to the aggregation/disaggregation of flows demanding such QoS mechanisms. We have hence analyzed, in this chapter, the performance of XCP under ACK losses in VBE.

Thus, we show that XCP becomes unstable under the presence of ACK losses when the bandwidth decreases. The instability degree of XCP depends directly on the RTT, the quantity of lost ACKs and the amount by which the available bandwidth is reduced.

In order to improve the robustness of the ERN protocols under feedback losses in VBE, we propose a new architecture for the ERN protocols. Our new ERN architecture does not add any extra mechanism to the ERN protocols and only requires a more active receiver. That is why we call our solution the "r" architecture. Finally, our experiments show that the r architecture (implemented and tested on the XCP protocol) improves significantly the robustness of the ERN protocols, in benefits of the fairness and throughput of flows.

5.1 Impact of ACK losses on XCP

As opposed to TCP, where ACKs only indicate good reception of data packets, in XCP each ACK packet also carries the feedback value (provided by the XCP routers) that controls the evolution of congestion window at the sender. This feedback represents an increment (or a decrement if the value is negative) to be applied to the congestion window size.

With TCP, if some ACKs are lost the only consequence is to delay the release of data buffers at the sender. For XCP, lost ACKs will cause a mismatch between the real network conditions in term of bandwidth availability and the conditions perceived by the sender. The impact of the difference between the real network state and the state perceived by the senders on the performance of the XCP flows depends, to a great extend, on the network conditions where XCP is executed.

5.1.1 Impact of ACK losses in fully controlled friendly networks

Let us define a fully controlled friendly network like a network where:

- the available resources, like the available bandwidth, remain constant over time and
- the flows are isolated and protected against mechanisms, running inside the network, with abilities to degrade the performance of these flows.

Currently, it is not very difficult to find fully controlled friendly networks. Some ISPs can guarantee bandwidth, jitter, etc., simulating dedicated networks to satisfy demands of customers. Most cases remain, however, dedicated networks used in Laboratories for analyzing different network protocols (testbeds).

In such fully controlled friendly networks, the losses of ACK packets have little impact on the ability of XCP to adjust the congestion window size of the sender, as shown in [84]. The explanation is as follows.

Case 1: ACK losses and smaller congestion window

Let an XCP flow with id i have a very small congestion window at time t, so it cannot grab all the available bandwidth ($cwnd(t)_i/RTT_i \ll ABW$). Thus, during the next RTT, the XCP routers will indicate a growth of the congestion window by mean of the feedbacks in the ACK packets, as indicated in Equation 5.1. The feedbacks should urge the sender to get all the available bandwidth, $cwnd(t+1)_i/RTT_i \approx ABW$.

$$cwnd(t+1)_i = cwnd(t)_i + \sum_{j=1}^{cwnd(t)_i} H_feedback\text{-}i_j \tag{5.1}$$

If the rate of flow i at time t is very low ($cwnd(t)_i/RTT_i \approx 0$) and we lose 30% of the total of ACK packets, the available bandwidth can be underutilized by 30% ($cwnd(t+1)_i/RTT_i \approx 7/10 * ABW$). 30% of the unused available bandwidth could be very important: 300Mbps in a 1Gbps link. However, in the next RTT the XCP routers will detect that the flow i does not grab yet all the available resources. Therefore, routers will calculate new feedback values to force the increases of the congestion window of the flow i.

If for any reason we still lose 30% of ACKs during the following RTT, 30% of the 300Mbps of remaining bandwidth will be unused (90Mbps). However, the rate of the sender will exceed

90% of the total link capacity (910Mbps). In conclusion, after a few RTTs (3 for example), a XCP sender can get almost all the available bandwidth even though ACK losses occur.

Case 2: ACK losses and larger congestion window

When the flow i will have a large enough congestion window to grab almost all the available bandwidth ($cwnd(t+n)_i/RTT_i \approx ABW$), the feedback sent to the sender must be almost zero. In this case, $cwnd(t+n+1)_i \approx cwnd(t+n)_i$, meaning that the increases of the congestion window during one RTT is approximately zero.

By intuition, the loss of some very small feedbacks will not change the result and the variation of the congestion window during the RTT must be similar: approximately zero.

5.1.2 Impact of ACK losses on VBE

Our studies concerning the impact of ACK losses on XCP in VBEs have been made by using the step-based bandwidth variation model. As we presented in Chapter 3, the step-based model reflects better than the sinusoidal-based bandwidth variation model the reality of VBE.

While the loss of ACK packets has little impact on fully controlled friendly networks, it is amplified in step-based bandwidth variation model networks when bandwidth reductions occur. In the analyzes made in the current subsection, we will suppose that the XCP flow suffering ACK losses has already gotten all the available bandwidth, but at time t the bandwidth increases (case 1), or decreases (case 2).

Case 1: ACK losses while bandwidth increases

This case is very similar to the case where the losses of ACK occur while the congestion window size is small and the available bandwidth remains constant. Thus, when the available bandwidth increases, XCP will only need a few RTTs (e.g., 3 RTTs) to grab all the available resources, even if ACK losses occur.

Case 2: ACK losses while bandwidth decreases

While the ACK losses when the available bandwidth increases affect insignificantly the utilization of the link, the case when the available bandwidth decreases could have more important consequences.

Imagine that our XCP flow with id i have a congestion window large enough to get all the available bandwidth at time t_2 ($cwnd(t_2)_i/RTT_i \approx ABW(t_2)$). If the available bandwidth decreases ($ABW(t_2+1) = D * ABW(t_2)$, where $0 < D < 1$), the congestion window must be adjusted to the new condition by mean of negative feedbacks (see Equation 5.1) to avoid congestion problems. In XCP the necessary changes to the *cwnd* value can be made in only one RTT.

However, if a percent of ACKs are lost, for instance 30%, and all the negative received feedbacks have similar values, after one RTT (at time t_2+1) the throughput of the flow i can be expressed as follows:

$$\frac{cwnd(t_2+1)_i}{RTT_i} \approx ABW(t_2+1) + 0.3 * (ABW(t_2) - ABW(t_2+1))$$

When the decrease of the available bandwidth represents only some Megabits per second, for instance 10Mbps, an error of 30% in the reduction of the *cwnd* (that represents approximately an overload of 3Mbps) could not produce major problems. The explanation is as follows: network operators require that router manufacturers provide 200ms (or more) of buffering [112] in order to accomplish the rule-of-thumb, as described in [113]. This way, in a 1Gbps router, the total buffer memory capacity can be of 125MB. However, in the standard configuration of a 1Gbps router, only 512KB is always available and the remaining memory is provided dynamically when memory is needed. Thus, an overload of 3Mbps (equivalent to 375KB) can be easily stored if the buffers are almost empty. It does not mean that overloads larger than 512KB will produce necessarily losses. However, overloads larger than 512KB could produce losses if the needed memory is not provided by the router as fast as needed.

In the case where the decrease of the available bandwidth represents a few hundreds of Megabits, for instance 300Mbps, an error of 30% in the reduction of the *cwnd* (that represents approximately 90Mbps) could produce some problems in the flows. Since 90Mbps (11MB) exceeds the fixed available memory (512KB), the probability that packet losses occur increases.

In addition, when ACK losses occur and several XCP flows share the bottleneck, the probability that all flows lose the same amount of ACKs is too low. Thus, some flows will be able to update more accurately the congestion window than other flows. Therefore, flows with a more accurate *cwnd* value can suffer fewer losses (or maybe no losses) than flows with a wrong congestion window, impacting negatively the fairness.

5.2 Simulating XCP on VBE and ACK losses

In this section, we show the simulation results of XCP on VBE (in a step-based bandwidth variation model more specifically) and random ACK losses. These simulations aimed at validating the hypothesis given in Subsection 5.1.2 by mean of the ns2 network simulator.

5.2.1 Topology and variation model

The topology used to analyze XCP on step-based bandwidth variation models is shown in Figure 5.1, where the bottleneck is the link connecting both routers R0 and R1, with a capacity of 1 Gbps. The delay in the bottleneck was fixed to 75ms. We chose a delay of 75ms in the bottleneck, because in some tests that we have made in Internet, we have found that this is the minimum delay between one computer in the center of France and another one in the center of US.

Concerning routers R0 and R1, both have buffers with a capacity equivalent to 12500 MSS, which is large enough to get a throughput of 1Gbps with an $RTT \approx 150ms$ in ns. We believe that a buffer with a capacity smaller than the BDP value represents better the reality, due to the small capacity of buffers in the real life.

Concerning the XCP flows, unless specified, in our experiments we will always send 10 XCP flows in parallel. This will allow us to observe the impact of ACK losses on both throughput and fairness.

On the other hand, the fluctuations in the bandwidth will be produced by ON-OFF UDP flows with a rate of 50Mbps each, similarly to Chapter 3. However, in this occasion we will turn on a maximum of 5 ON-OFF UDP flows, producing a minimum available bandwidth

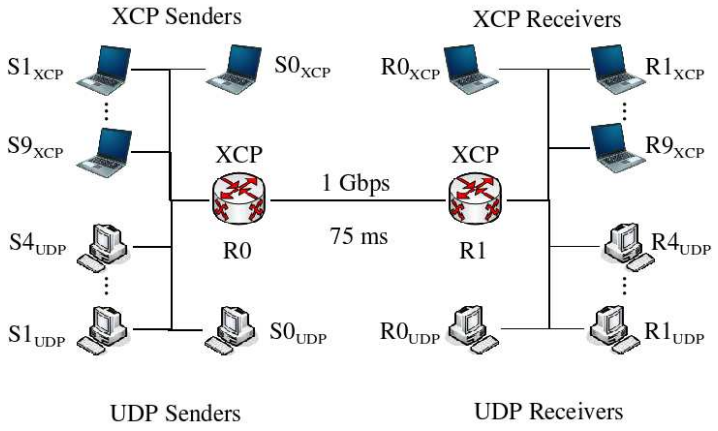


Figure 5.1: Topology to test XCP on a VBE with ACK losses.

of 750Mbps[1]. The minimum number of active ON-OFF UDP flows will be zero, letting a maximum rate of 1Gbps to XCP flows. Figure 5.2 shows how we will vary the available bandwidth during our simulations.

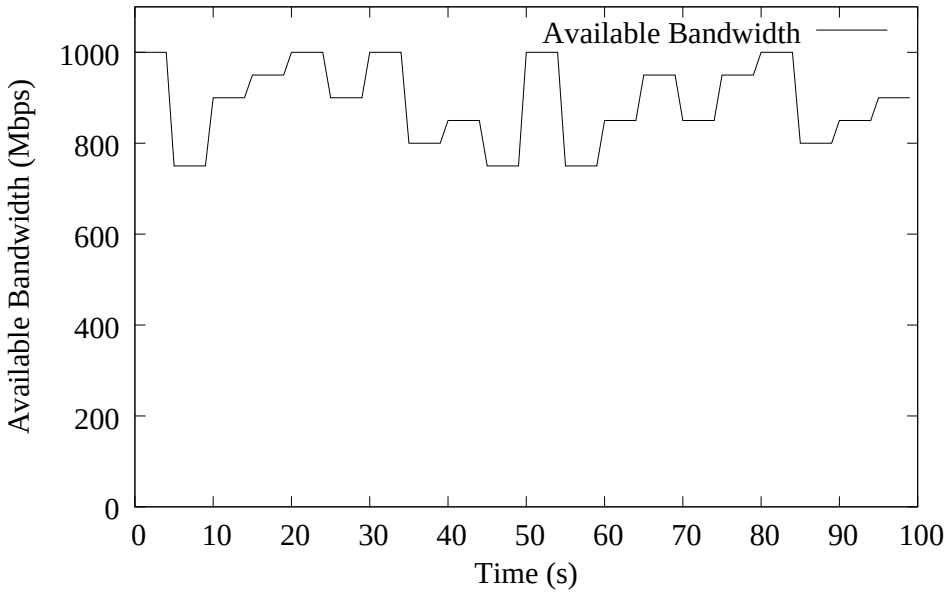


Figure 5.2: Variation model caused by ON-OFF UDP flows.

5.2.2 No ACK losses

In Chapter 3, we already showed that XCP reacts well in step-based bandwidth variation model (see Section 3.2.3). XCP proves (i) to grab quickly bandwidth to maximize link utilization

[1] We have used a minimum of 750Mbps to show that even small changes in the available bandwidth can have an important impact on the XCP flows.

and (*ii*) to yield quickly bandwidth to avoid losses. However, when losses occurred, XCP is able to recover packet losses and get all the available bandwidth in 5 seconds in average.

Now, we test XCP in the step-based bandwidth variation models presented in Subsection 5.2.1 without ACK losses. The main goal being to get a point of reference for future simulations results that will be presented along the current chapter, where the problems of ACK losses will be introduced.

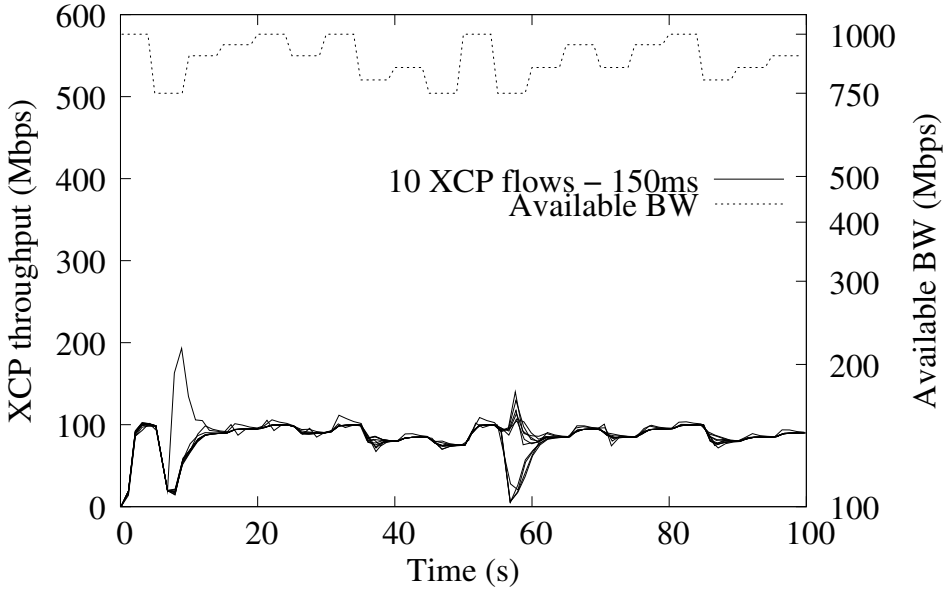


Figure 5.3: 10 XCP flows on a step-based VBE without ACK losses.

The performance of XCP in a step-based bandwidth variation model in absence of ACK losses is presented in Figure 5.3, where we have plotted the throughput of all the 10 XCP flows (continuous lines), as well as the available bandwidth evolution (dashed line). As we can see in Figure 5.3, during almost all the simulation, XCP reacts well to the decreases of the available bandwidth, keeping a high fairness between flows and maximizing bandwidth utilization. XCP has some problems only at seconds 5 and 55, when the bandwidth suffers the more drastic changes (the available bandwidth decreases from 1Gbps to 750Mbps).

The problems that we can observe at seconds 5 and 55 were packet losses due to the decreases in the bandwidth. In both occasions, lost resources by flows suffering losses of packets are acquired by those flows avoiding losses. Thus, at second 5, one flow gets a maximum throughput of 200Mbps. However, the unfairness problem at second 5 (the more important during all the simulation) is solved in only 5 seconds approximately. Thus, the impact of packet losses is minimum in the global fairness.

After packet losses, the time needed to recover from such losses is strongly influenced by the number of lost packets:

- When losses are detected by DUPACKs, the lost packets are resent by executing Fast Retransmit/Fast Recovery (also available in XCP, since it is build at the top of TCP). Fast Retransmit/Fast Recovery is the fastest mechanism to recover from losses, since the lost packets are resent in only one RTT. However, generally Fast Retransmit/Fast

Recovery is executed when the number of lost packets is very small in comparison to the number of sent packets during one RTT ($losses \ll cwnd$).

- When the number of lost packets increases and there are not enough received packets after losses to transmit three DUPACKs, a timeout is triggered at the expiration of an RTO timer. Since the RTO is calculated at every ACK received, the RTO value is strongly impacted by the number of lost packets increases.

In the case of XCP in the step-based bandwidth variation model without ACK losses, the logs given by our simulations show that the total of lost packets at seconds 5 and 55 were 107 and 167 packets respectively.

5.2.3 ACK losses

In wired networks, losses of packets are frequent and are produced mainly by bursts of packets in senders. Such bursts can easily drop a large amount of packets belonging to other flows. Many studies about the impact of burstiness can be found in the literature ([114, 115, 116] between others).

In some of our simulations, when a flow starts sending packets (flows in the slow-start phase) in a networks where all the resources are busy, the number of dropped packets could be very important. In fact, we can lose more than 30% of total packets crossing the bottleneck. With this idea in mind, we decided to introduce an ACK loss rate of 10% with a constant distribution in our simulation scripts, to analyze the behavior of XCP under ACK losses. We believe that the constant distribution does not represent exactly the reality. However, a constant distribution facilitates the analyzes of the XCP behavior under ACK losses and VBEs.

Figure 5.4 shows the performance of 10 XCP flows with an $RTT \approx 150ms$, in a step-based bandwidth variation model and an ACK loss rate of 10%. In Figure 5.4 we can observe that when the decrease of the available bandwidth is smaller than 250Mbps, XCP behaves similarly to the case where no ACK losses are present.

Now, let us concentrate in the parts where the available bandwidth decreases up to 250Mbps (seconds 5 and 55). Comparing Figures 5.3 (no ACK losses) and 5.4 (10% ACK loss rate), we can see that at second 5, the behavior of XCP becomes chaotic when ACK losses are introduced. In fact, after losses, XCP flows need approximately 10 seconds before stabilizing their throughput and getting fairness. When ACK losses are introduced in the simulation, the number of lost data packets is increased to 204 packets, which represents 97 packets more than the case where no ACK loss occurs.

When we compare Figures 5.3 (no ACK losses) and 5.4 (10% ACK loss rate) at the second 55 of the simulation, we can see that the impact of ACK losses on XCP is much more important. In fact, when ACK losses occur, one XCP flow gets approximately 550Mbps, meaning that this flow is exceeding the fairness point by approximately 450Mbps, while some other flows stay inactive during 3 or 8 seconds in the worst case. In addition, in the case where ACK losses are introduced, approximately 15 seconds are needed before stabilizing the throughput of flows at the fairness point.

Finally, we want to remark that simulation results shown in Figure 5.4 prove our hypothesis given in Section 5.1.2, in the sense that flows are not similarly affected during bandwidth reductions when ACK are lost, impacting fairness negatively.

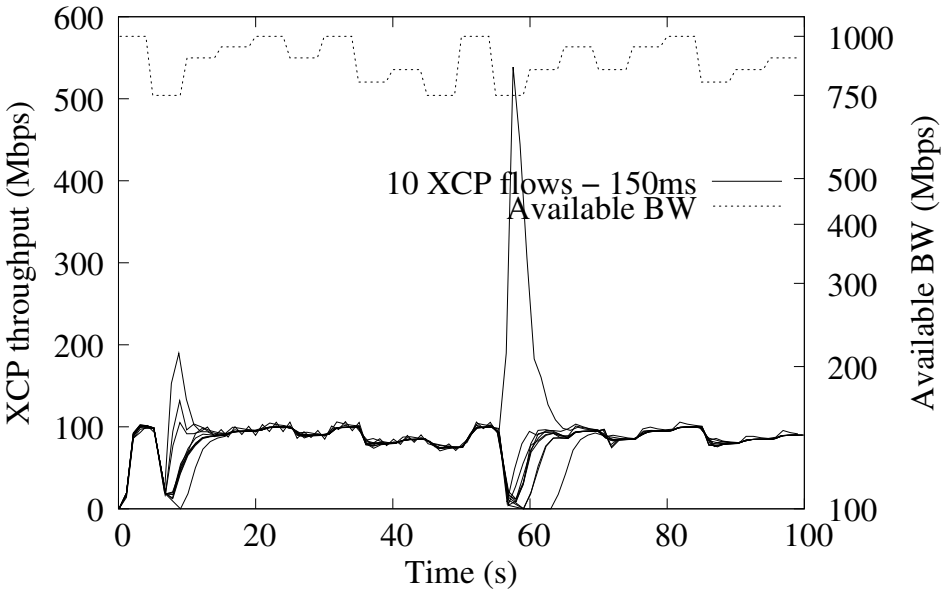


Figure 5.4: 10 XCP flows on a step-based VBE with 10% of ACK losses.

5.3 XCP-r: Building a robust XCP version

The problem of instability of XCP on VBE and ACK losses lies on the fact that feedbacks are summed up by the sender in order to incrementally compute the optimal congestion window. The reason for doing so is that using ACKs to transfer feedback from routers to senders avoids introducing an unnecessary overhead in the network (no specific packets are needed to transfer feedbacks). However, in previous sections, we have shown that on dynamic networks, XCP could have unstable behavior due to losses of ACKs.

In this section, we describe a new XCP protocol architecture able to limit the impact of ACK losses. The new XCP architecture lies in the migration of some parts of the protocol stack from the sender to the receiver, that is why we call this new architecture the "r" architecture. Thus, we will call as XCP-r all the XCP flows using the r architecture. Since our r architecture is transparent to the routers, this new architecture keeps the complex router code untouched.

The main idea behind XCP-r is to avoid having the incremental feedbacks summed up by the sender, while keeping the incremental feedbacks at the router level for flexibility and robustness. Hence, we propose to calculate the congestion window size at the receiver and to send this value to the sender in every ACK. Thus, as each ACK packet now carries the target *cwnd*, instead of a value that the sender must add to its congestion window size, the time to update the *cwnd* value correctly decreases. In fact, after ACK losses, the next ACK arriving to the sender will contain the correct *cwnd* value (no more ACK will be needed to correctly update the congestion window).

In a first version of XCP-r, the receiver simply uses the H_cwnd field of the data packet (which stores the current congestion window size of the sender) to recompute the target *cwnd* from the $H_feedback$ field (see Chapter 2, Section 2.5.3). The formula was

$$cwnd = (H_feedback * H_cwnd) + H_cwnd$$

which simply relies on the fact that the $H_feedback$ field was computed by the routers so as to distribute the total increase amount in congestion window into H_cwnd increments of $H_feedback$ value. However, since the routers change the feedback value every interval time that corresponds to the average RTT of all the flows crossing the router, if this average RTT is different from the current RTT seen by the sender, then one congestion window can have assigned different $H_feedback$ values. Later, our simulation confirmed the validity of our hypothesis.

We have hence proposed to reproduce at the receiver the computations that are performed at the sender in the original proposition. To do so, the receiver maintains a mirror variable of the sender congestion window size per flow, $cwnd'$, which is created on reception of an XCP SYN packet (that has the same meaning than a TCP SYN packet) and set to 1. $cwnd'$ evolves according to the $H_feedback$ value received in data packets as shown in Equation 5.2. Finally, in every ACK sent to the sender, we will insert the $cwnd'$ value instead of the $H_feedback$ carried in the data packet.

$$cwnd' = cwnd' + H_feedback \tag{5.2}$$

Note that whenever the sender decides to modify $cwnd$, it must synchronize with the receiver beforehand. In XCP, such a modification happens only when there is a congestion that either set cwnd to 1 or to half of its value. In both cases, the sender must indicate, in a one-bit congestion flag field of the XCP header, that the receiver should read the value of H_cwnd before adding a new $H_feedback$ value ($cwnd' = H_cwnd$). Figure 5.5 illustrates the new set of operations of the XCP-r receiver.

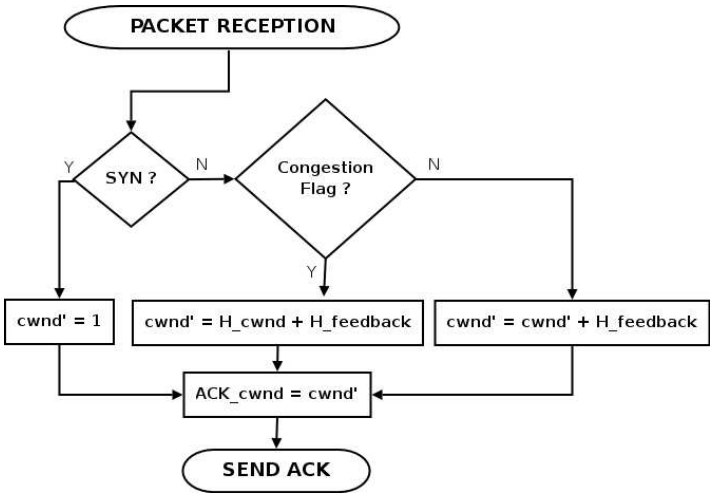


Figure 5.5: The XCP-r receiver algorithm.

Regarding the XCP-r sender, the only modifications are:

1. instead of doing $cwnd = cwnd + H_feedback$, simply do $cwnd = H_feedback$.

2. set the congestion flag whenever the $cwnd$ is reset.

5.4 Simulating XCP-r on VBE and ACK losses

5.4.1 Scenario of simulation

To test XCP-r on VBEs and ACK losses, we will use the topology and the bandwidth variation model shown in Figure 5.1 and 5.2 respectively. Concerning the ACK losses, we will introduce an ACK loss rate of 10% with also a constant distribution, in order to compare XCP and XCP-r under the same conditions.

5.4.2 Simulation results

Improving the robustness with XCP-r

The performance of XCP-r, in a step-based bandwidth variation model with an ACK loss rate of 10%, is shown in Figure 5.6, where we have plotted the performance of 10 XCP-r flows (continuous lines) and the bandwidth evolution (dashed line). As we can see in Figure 5.6, at second 5, the behavior of the XCP-r flows when 10% of ACK losses is introduced in the simulation, is almost similar to the case of XCP under the same conditions. We could think that there is no difference between XCP and XCP-r. The similarities between XCP and XCP-r under an ACK loss rate of 10% can be explained by the number of dropped packets. In fact, at second 5, XCP-r flows has lost a total of 192 packets, that represents only 12 packets less than XCP under the same conditions. However, the number of lost packets proves also that XCP-r improved slightly the performance of XCP, even though it is not easily perceivable on the presented performance graph.

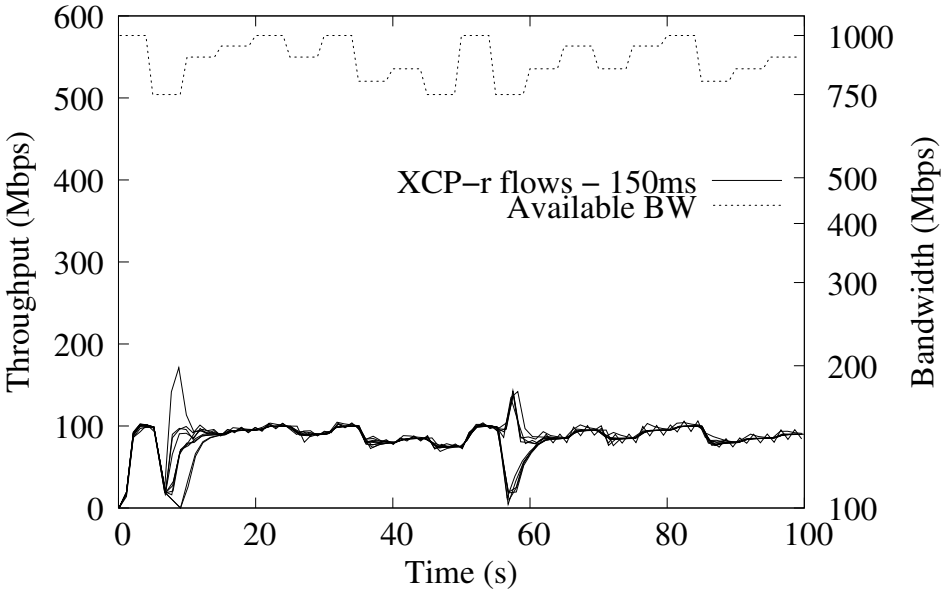


Figure 5.6: 10 XCP-r flows on a step-based VBE with 10% of ACK losses.

However, comparing Figures 5.4 (XCP with a 10% ACK loss rate) and 5.6 (XCP-r with a 10% ACK loss rate) at second 55, we can see that XCP-r is able to decrease significantly the impact of ACK losses on the flows. Thus, the very chaotic behavior of XCP at second

Losses at second	Dropped Packets		
	XCP No ACK losses	XCP 10% ACK loss rate	XCP-r 10% ACK loss rate
5	107	204	192
55	167	414	302

Table 5.1: Lost packets by XCP and XCP-r

55, where some flows stay inactive during many seconds (up to 8 second in the worst case) and where one of the flows which do not suffer losses gets up to 550Mbps (impacting strongly the fairness), was changed by XCP-r to more stable behavior, where no flow stays inactive due to timeouts and where flows which do not suffer losses get at most 150Mbps. In fact, the behavior of XCP-r under ACK losses, at second 55, is very similar to the behavior of XCP when no ACK losses are introduced in the simulation (Figure 5.3).

The difference between XCP and XCP-r at second 55, when an ACK loss rate of 10% is present in the simulation, can also be explained by the number of dropped packets. In fact, XCP-r suffered a total of 302 lost packets, which represents 112 packets less than XCP under the same conditions. Table 5.1 summarizes the losses of packets suffered by XCP and XCP-r presented so far.

Improving the fairness with XCP-r

The simulations results of XCP without and under ACK losses and XCP-r under ACK losses have been plotted in Figure 5.7. In this figure, we can observe the average throughout of:

1. 10 XCP flows without ACK losses, each flow represented by a point at every designed radius of the polygon and linked by a dotted line.

2. 10 XCP flows under an ACK loss rate of 10%, each flow represented by a square at every designed radius of the polygon and linked by a line.

3. 10 XCP-r flows under an ACK loss rate of 10%, each flow represented by an inverted triangle at every designed radius of the polygon and linked by a dashed line.

When we plot the average throughput of every flow crossing a network during a period, in a diagram as the one presented in Figure 5.7, if we link each neighboring flow by a line, the symmetry of the resulting regular polygon is perfected as the fairness increases. Thus, with Figure 5.7 we can compare easily the impact of ACK losses in the achieved fairness by XCP and XCP-r.

In this case, our point of reference will be the polygon given by the XCP protocol without ACK losses. As we can see, when no ACK losses are introduced in the simulation, XCP shows a very good fairness level denoted by an almost regular polygon. In fact, the lowest average throughput is 83.5Mbps and the highest 87.5Mbps. However, the polygon given by XCP when an ACK loss rate of 10% is introduced in the simulation, is very far from a regular polygon. The variations are easily visible in Figure 5.7, where the lowest average throughput is 81Mbps and the highest 98Mbps.

On the other hand, the polygon given by XCP-r when an ACK loss rate of 10% is introduced in the simulation, is much more symmetrical than the one given by XCP under the

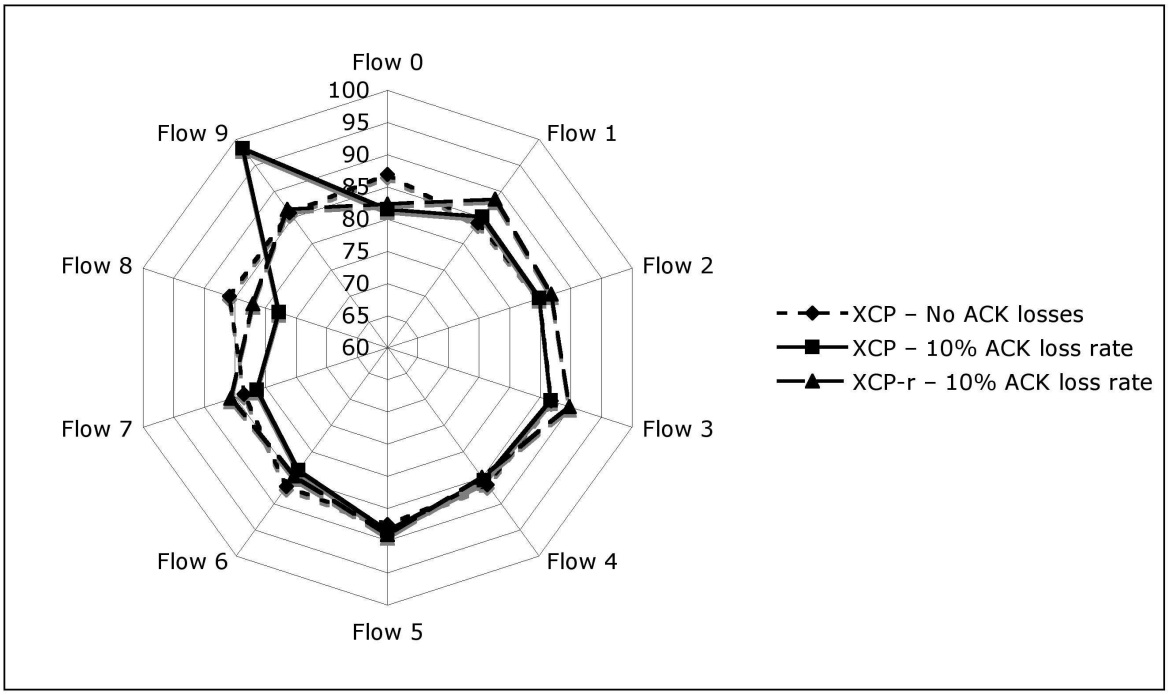


Figure 5.7: Average throughput of flows.

same conditions. The lowest average throughput is 82Mbps and the highest 89.5Mbps (results close to the one of XCP without ACK losses). Thus, Figure 5.7 illustrates very well the improvement in the fairness provided by XCP-r on VBEs and ACK losses.

5.5 Other simulations of XCP & XCP-r under ACK losses

In order to reinforce the benefits of XCP-r in VBEs, we carried out other simulation results proving that XCP-r is able to improve the performance of XCP in a wide variety of network conditions. The set of simulations that we present in the current section were executed over a topology similar to the one shown in Figure 5.1, but with the following differences:

1. the maximal capacity of the bottleneck is 200Mbps.
2. the minimal capacity of the bottleneck is 50Mbps.
3. the delay is set to 50ms ($RTT \approx 100ms$).
4. we introduce an ACK loss rate of 30%.

First of all, Figure 5.8 shows the behavior of 3 XCP flows in a step-based bandwidth variation model without ACK losses. We will not give a detailed description of the performance of the XCP flows. We want only to remark that XCP seems to be stable during all the simulation. The more dramatic changes in the available bandwidth (at seconds 5 and 30) do not produce inactivity in the data transfer of flows.

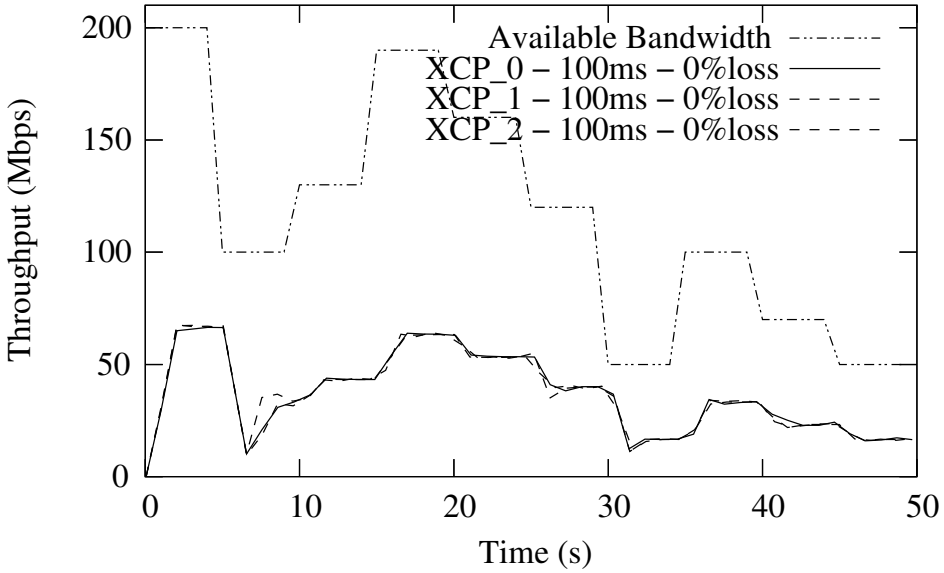


Figure 5.8: XCP - no ACK losses.

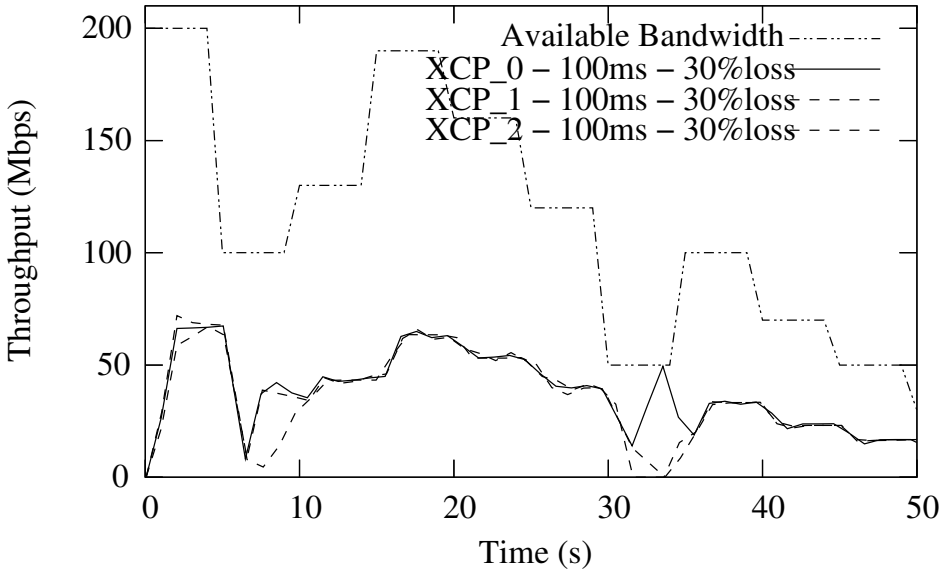


Figure 5.9: XCP - 30% ACK loss rate.

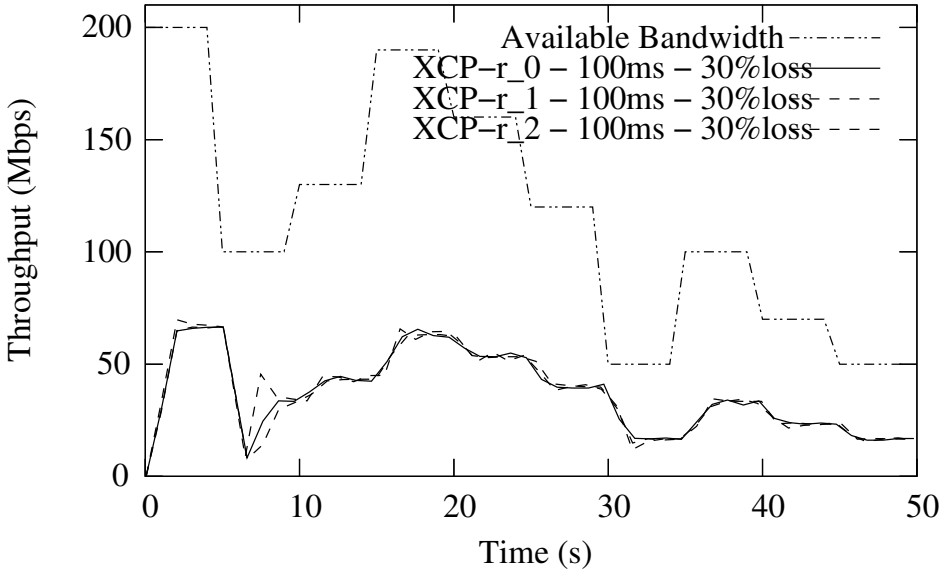


Figure 5.10: XCP-r - 30% ACK loss rate.

However, when an ACK loss rate of 30% is introduced, XCP becomes unstable, as we can see in Figure 5.9 at seconds 5 and 30 (when the available bandwidth suffers the more drastic changes). At second 5, we can observe a flow suffering a timeout. At second 30, two flows suffering timeouts are not able to restart the connections quickly, while the remaining flow gets all the resources during a few RTTs.

On the other hand, Figure 5.10 shows that XCP-r improves the stability and robustness of flows, in benefits of the throughput and the fairness when ACK losses occur. XCP-r flows in this case do not suffer timeouts and their performance is very similar to the performance of XCP in absence of ACK losses.

5.6 Conclusion

The success of the XCP protocol (and most of the ERN protocols) is based mainly on the information given back by the routers to the XCP senders, which are carried in the ACK packets. However, in IP networks the losses of packets (and therefore, losses of ACKs) are frequent. In IP networks, losses of packets are caused mainly by burst of packets. Such losses can negatively influence the performance of flows, as shown in several studies available in the literature.

Since our main goal in this thesis is to provide an ERN protocol able to be deployed gradually in IP wired networks, we analyzed in this chapter the behavior of XCP (theoretically and by simulations) in a step-based bandwidth variation model without ACK losses and under ACK losses. With these studies, we have found that ACK losses can impact negatively the performance of XCP when the available bandwidth decreases, since ACK losses produce a congestion window which is not adapted to the network conditions. The result of an unadapted congestion window in some cases is losses of data packets, that can potentially produce timeouts, unfairness and instability of flows.

To diminish the impact of ACK losses, we proposed a new architecture for the ERN protocols, called the r architecture, since this new architecture is based on the transfer of some parts of the code from the sender to the receiver. We implemented the r architecture on XCP to test this solution. This is why we call XCP-r all the XCP flows using the new r architecture. With XCP-r, the receiver uses the computed feedback included in every data packet to compute the congestion window of the sender. Thus, every ACK packet sent to the sender carries the accurate *cwnd* value, instead of a value that must be used by the sender to compute the *cwnd* value.

The r architecture keeps the policies in the routers unchanged since this new architecture is transparent to the routers. The r architecture only needs to add a synchronization mechanism between the sender and the receiver in order to keep the consistency of the *cwnd* value, when losses force some changes in the congestion window size. This synchronization mechanism is based on the use of a one-bit flag. Thus, the r architecture avoids introducing overhead in the network.

It is important to note that the r architecture has limited benefits when the available bandwidth does not change over time and/or when all ACKs are correctly received by the sender. However, in VBEs and under ACK losses, the performance of flows using the r architecture increases, since one ACK received by the sender, after several ACK losses, allows the sender get a correct *cwnd* value. Thus, in our tests of XCP-r flows in a step-based variable bandwidth model, we have seen that our new architecture improved the robustness of flows when ACK losses occur, in benefit of the stability, the fairness and the throughput of flows.

Chapter 6

Enabling inter-operability between TCP and ERN protocols

In precedent chapters we proposed an extension for the routers algorithm of the ERN protocols, to enable the inter-operability between ERN protocols and non-ERN routers (Chapter 4). Anticipating also the problems of feedback losses, that ERN protocols can suffer in heterogeneous LDHP networks, we proposed a new architecture that improves the robustness of ERN protocols in presence of such losses in VBEs (Chapter 5).

However, as we will see later, most of ERN protocols do not have any mechanism to compete successfully against E2E congestion control protocols, like TCP. This problem, which denounces a lack of inter-operability between ERN and E2E protocols, lies on the fact that ERN protocols take all the resources leaving nothing to E2E protocols, or leave all the resources to E2E protocols, taking only the remaining resources.

In this chapter we will present the studies that we have made to provide a fair sharing of resources between TCP and ERN protocols, subject that has never been explored. To study the behavior of ERN protocols when they share the resources with E2E protocols, we have created scenarios where the bottleneck is shared by XCP and TCP flows.

Thus, we will first identify and analyze the problems at the origin of unfairness between XCP and TCP, and in a second step, we will make some propositions to enable the inter-operability between these two kinds of congestion control protocols: the ERN protocols and the E2E protocols.

We believe that the inter-operability between ERN and E2E protocols must be covered if we want to benefit from an ERN protocol (like XCP or any other) in large data transfers on heterogeneous networks. Finally, we want to remark that the solutions presented in this chapter, implemented and tested on the XCP protocol, have been designed to be easily applied to any ERN protocol. This flexibility is possible because of the independence of the mechanisms used in our solutions and the XCP algorithm.

6.1 Why XCP flows are unable to compete against TCP flows

As described in Chapter 2 Subsection 2.5.3, the XCP router executes two control laws by every incoming XCP data packet:

1. The EC that maximizes the link utilization while minimizing losses of packets.

2. The FC that assigns resources fairly between XCP flows.

To maximize the link utilization, the EC computes a value named feedback, denoted by the symbol ϕ, that reflects the state of the link (the available resources):

$$\phi = \alpha.rtt.(O - I) - \beta.Q$$

where α and β are constant with values of 0.4 and 0.226 respectively. rtt is the average RTT of all the packets crossing the router, O the output link capacity, I the input traffic rate seen by the XCP router during a control interval and Q, the persistent queue size. Note that in the feedback equation, ϕ decreases as the input traffic rate increases and the available bandwidth decreases.

Later on, the FC translates the computed general feedback, ϕ, in a feedback per packet, that will increase or decrease the rate of each sender, in order to assign the same amount of bandwidth to each flow. The feedback per packet, computed by the FC, will replace the *H_feedback* value carried in the data packet if the computed feedback is smaller than the one carried in the header. After that a data packet arrives to the XCP receiver, two ways are possible to allow the sender update its congestion window size:

1. if we are using the XCP standard protocol, the receiver will copy the $H_feedback$ from the data packet to its ACK packet. At the reception of the ACK, the sender will add the $H_feedback$ value to the current *cwnd* value.

2. if we are using XCP-r (as described in Chapter 5), the receiver will calculate the new congestion window size, and that value will be sent to the sender back by mean of the ACK. At the reception of the ACK, the sender will replace its current *cwnd* value by the new value found in the ACK.

The FC computes a feedback per packet and makes XCP flows to achieve a very high fairness level, as shown in some simulations presented in Chapter 4 or 5. However, the FC does not have any mechanism allowing XCP achieve fairness between XCP and non-XCP flows, like TCP. But the problem starts earlier, in the EC.

The EC maximizes the resources utilization and for this reason, it computes the available bandwidth, also called *spare bandwidth*, given by $S = O - I$. Analyzing the spare bandwidth equation, we can see that it takes into account the total input traffic rate seen by the router, without any mechanism to differentiate between traffic generated by XCP and non-XCP flows. Hence, the input traffic rate could be generated by any flow using any UDP or TCP version, forcing XCP flows to take only the remaining available bandwidth. Logically, non-XCP flows are not able to take into account the XCP feedback (that is why non-XCP flows are called also as uncontrolled flows). Thus, XCP routers are not able to limit the sending rate of non-XCP flows, like TCP flows.

6.2 XCP vs TCP

Using the simple topology shown in Figure 6.1, we analyzed the behavior of one XCP flow ($Sn_{XCP} = S0_{XCP}$ and $Rn_{XCP} = R0_{XCP}$), when competing with two TCP flows ($Sm_{TCP} = S1_{TCP}$ and $Rm_{TCP} = R1_{XCP}$) by mean of a simulation. In this topology, both routers R0 and R1 have buffers with a capacity equivalent to 5000 MSS, which is large enough to get an XCP

throughput of 1Gbps with an $RTT \approx 100ms$. We believe also that a buffer with a capacity smaller than the BDP value represents better the reality, due to the small capacity of routers buffers in the real life. Buffers are shared by XCP and TCP flows. Routers use a First-In First-Out (FIFO) algorithm and no priority policies have been defined for the treatment of packets.

The simulation results can be found in Figure 6.2(a) and Figure 6.2(b), where we have plotted the congestion window and the throughput respectively of the all three flows.

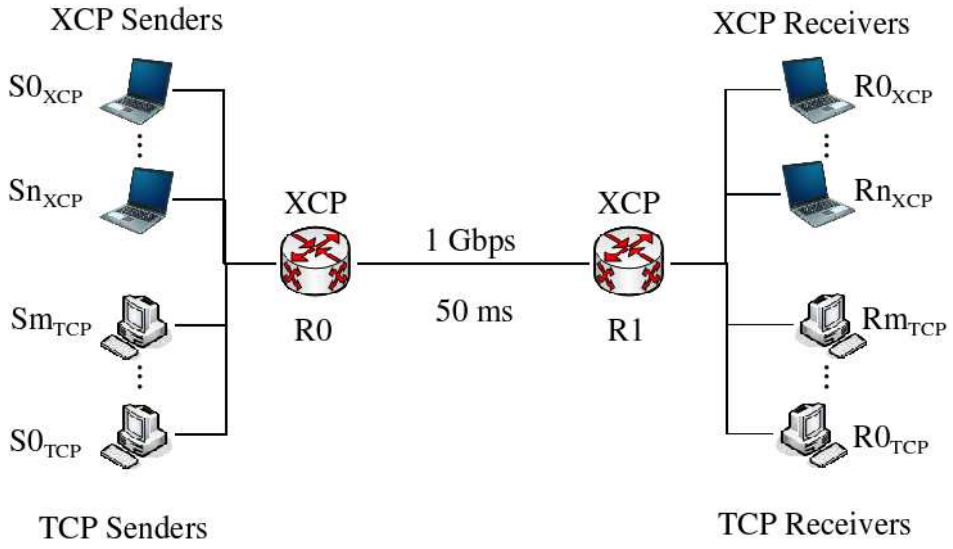


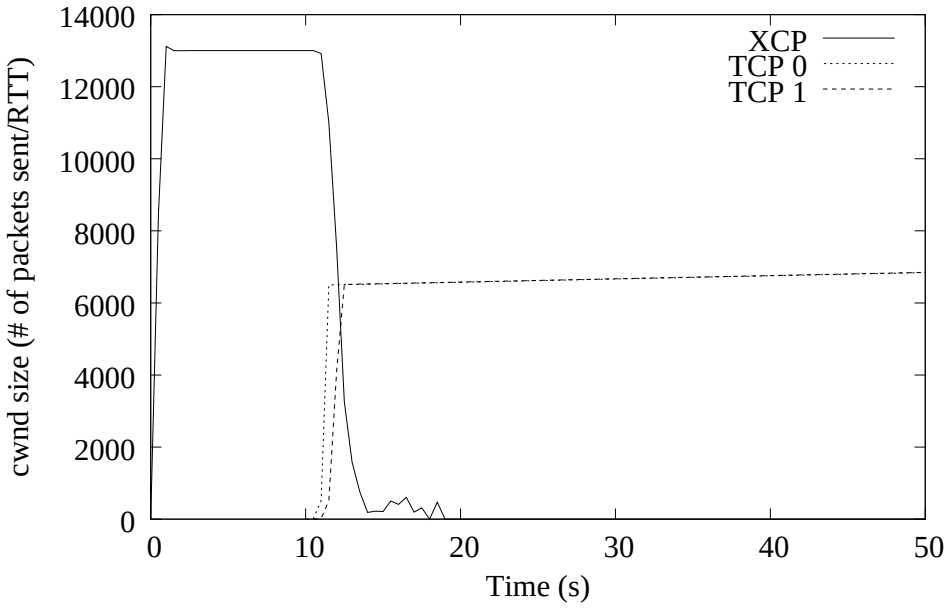
Figure 6.1: Topology: n XCP and m TCP flows sharing the bottleneck.

First, as we can see in Figure 6.2(b), the XCP flow starting at second 0 takes all the available bandwidth while it does not share the resources with concurrent TCP flows. The XCP flow describes the classical XCP performance that gets all the resources after a few RTTs.

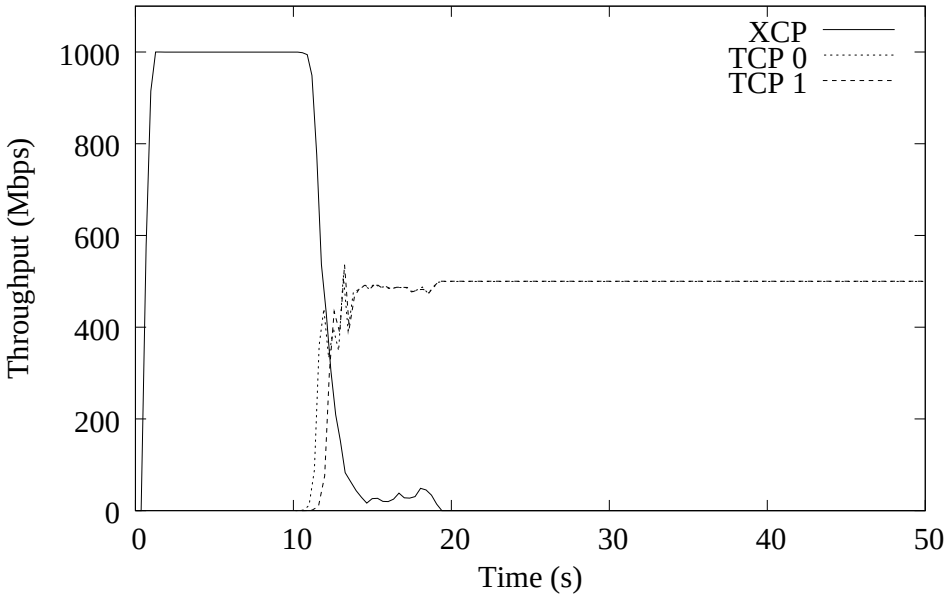
However, at second 10 when two TCP flows appear in the scenario, the performance of the XCP flow is strongly degraded. In fact, after the second 15, the performance of XCP is reduced from 1 Gbps to only ~30 Mbps. After second 19, the XCP flow has almost disappeared and the congestion window of the XCP flow stays at its minimum value: only one packet. Figure 6.2(a) shows the dramatic fall of the XCP congestion window.

Between seconds 15 and 19, we observe a strange behavior of the XCP flow: it seems that the XCP flow tries to get some bandwidth but without success. The explanation of this phenomenon is as follows: since the congestion window size of both TCP flows are not yet large enough to keep a continuous transfer of packets, the XCP router placed in the bottleneck detects a positive difference between the output link capacity and the input traffic rate during a few milliseconds. This very small remaining bandwidth is assigned to the XCP sender, which has still the opportunity of transferring some data packets.

However, after second 19, the congestion window size of both TCP flows are large enough to keep a continuous transfer of packets, inclusively, to keep some packets in the buffer of the routers permanently. The result is a constant negative feedback that will decrease the XCP congestion window to the smaller size ($cwnd = 1MSS$). The behavior of XCP between second 15 to 19 is therefore not motivated by a fairness mechanism between XCP and non-XCP flows



(a) Congestion window evolution



(b) Throughput

Figure 6.2: XCP flow dealing with TCP concurrent flows.

as we could think.

6.3 Propositions for an XCP-TCP friendliness mechanism

We have seen in the previous section that XCP has no mechanism providing friendliness between XCP and non-XCP flows. Currently, to achieve bandwidth fair share between XCP and non-XCP flows, only the Dynamic Queue Weight [117] strategy has been proposed in the web page of the ns2 network simulator. The Dynamic Queue Weight, which consists in dividing the bandwidth at the middle and that has been added to the XCP protocol in the ns2 simulator (2.29 release), is only given as an early idea about how to ensure XCP-TCP friendliness. We do not know currently any Dynamic Queue Weight formal scientific document.

In the next subsections we will present in detail a new mechanism that could be added to the XCP routers, in order to ensure friendliness between XCP and TCP flows. In the remaining of this chapter, we will replace the term non-XCP by TCP, because our main goal is to enable the inter-operability between TCP and XCP.

6.3.1 Defining the XCP-TCP friendliness

The main goal of our XCP-TCP cohabitation mechanism is to ensure the fair share of resources between XCP and TCP flows. In any case we are not looking for the improvements of the intra-protocol fairness of TCP or XCP. Here, we will define an XCP-TCP friendliness criteria, like the fact that TCP allows to XCP to get a given amount of bandwidth, equivalent to the ratio between the link capacity, and the sum of the number of XCP and TCP flows, multiplied by the number of XCP flows, even though TCP could get more bandwidth. Equation 6.1 shows this relationship.

$$BW_{XCP} = \frac{LinkCapacity}{N+M} * N \tag{6.1}$$

In Equation 6.1, BW_{XCP} is the amount of resources needed by XCP to consider that a TCP friendliness exists, N is the number of active XCP flows and M the number of active TCP flows. In addition, we can estimate the amount of resources needed by TCP, BW_{TCP}, as a function of the resources needed by XCP:

$$BW_{TCP} = LinkCapacity - BW_{XCP} \tag{6.2}$$

If we apply the Equations 6.1 and 6.2 to the experiment shown in Figure 6.2, we find that the optimum bandwidth repartition between XCP and TCP is obtained when $BW_{XCP} \approx 341.33 Mbps$ and $BW_{TCP} \approx 682.66 Mbps$.

We wish to remark that the discussions about the sharing of resources between flows belong to the area of inter or intra protocol fairness. However, studies about fairness require the definitions of multiple metrics that we are not considering [118]. Therefore, the way of studying the XCP-TCP cohabitation issues in this chapter is more related to the problems of *friendliness* between XCP and TCP.

6.3.2 Estimating the resources needed by XCP

In order to provide TCP friendliness, we can deduce from Equation 6.1 that we will need to know or estimate the number of XCP and TCP active flows. Calculating the number of active flows crossing a router is a very hard task. Some strategies have been proposed in the literature to compute or estimate the number of active flows in a router during a given period of time.

The best known strategy consists in looking at the ID flow of every incoming packet in a router, and searching for this ID in a table keeping the ID of each active flow crossing the router. If the searched ID is not found, then it is added into the table.

Thus, when the router will need to know the number of active flows, the router must only compute the number of entries in the ID table. This strategy has the benefit of exactly computing the number of active flow crossing a router during an interval of time. However, this strategy is very expensive. In fact, we will search for the ID of every incoming packet in a table that may keep many thousands of IDs. Therefore the time needed to find an specific ID can be very large. To impose this strategy to compute the number of XCP and TCP flows becomes is not realistic.

Another algorithm to estimate the number of active flows is based on the Bloom filter algorithm [119]. The Bloom filter algorithm was applied by NRED [120] in order to get an idea about the number of active flows crossing a bottleneck. The NRED's Bloom filter algorithm works as follows:

When a packet arrives, the NRED's Bloom filter algorithm hashes the source-destination pair of the packet into a bin. A bin is a 1 bit memory to mark 0 or 1. The estimator maintains $P \times L$ bins. The bins are organized in L levels, each of which contains P bins. The estimator also uses L independent hash functions, each of which is associated with one level of bins, that maps a flow into one of the P bins in that level. For every arriving packet, the L hash functions can be executed in parallel. At the beginning of the measurement interval, all bins and a counter N_{act} are set to zero. When a packet arrives at the router, the L hashed bins for the source-destination IP address pair of the packet are set to 1 and N_{act} increases by one unit if at least one of the L hashed bins is zero before hashing. However, it could happen that packets belonging to two different flows are hashed into the same bins of L levels causing a "misclassification". The authors claim that the probability of having a "misclassification" decreases as the number of L hash functions increases.

The problem of the Bloom filter algorithm is based on the fact that to avoid "misclassification", an important number of hash functions is needed, and since every hash function is executed one time for every incoming packet, this algorithm could be expensive in terms of CPU requirements. In addition, even if the hash functions can be performed in parallel, those hash functions must synchronize in order to update N_{act}, making this algorithm expensive in terms of time.

Finally, the last of the best known algorithm for estimating the active flows in a router is the *zombie estimator* used in the SRED algorithm [71], already described in Chapter 2, Subsection 2.5.1. The zombie estimator keeps a table called *zombie*, able to keep up to 1000 ID flows (the ID flow could be src_addr:src_port::dest_addr:dest_port). The zombie table is filled in at the beginning with the ID flows belonging to the first 1000 incoming packets.

When the zombie table has been filled in, every packet arriving to the router is examined. First, the ID flow of the packet is taken and compared with an ID flow randomly taken from the zombie table. If the IDs are equal, an event *hit* is declared. Otherwise, an event *mis* is

declared. When a *mis* is detected, with a probability of 25%, the old stored ID in the zombie table will be replaced by the new flow ID. Later on, in both cases *hit* or *mis* an equation that computes the probability to make a *hit*, $P(t)$, is updated (see Equation 2.28 in Chapter 2). Now, if the total of active flows in a router is N, then the probability to get a *hit* is $1/N$. However, the probability to get a *hit* is already contained in $P(t)$. Therefore, $1/P(t)$ represents the estimation of the number of active flows seen by a router at time t.

The operations executed by the zombie estimator, contrary to other methods, do not require a big utilization of CPU, since this estimator only needs one comparison, to generate two random numbers, and in some cases, one writing for every incoming packet. However, one of the weakness of the zombie estimator lies on the fact that the zombie estimator does not calculate the exact number of active flows, but only makes an estimation of this number.

We believe that with current technologies it is very difficult to know the exact number of active flows during a given period (we need faster CPU and bigger memory than those currently available in routers). We believe also that having an estimation of the number of active flows is enough to ensure friendliness between XCP and TCP. For this reason, we have chosen the mechanism proposed in the SRED algorithm in order to estimate the number of active TCP and XCP flows.

The way to implement the zombie estimator in the XCP routers is very similar to the way proposed in SRED. However, in our case it is necessary to keep two zombie tables: the first one will be filled in with the ID flow of the first 1000 XCP incoming data packets, while the second one will be filled in with the ID flow of the first 1000 TCP incoming data packets. Thus, we will have one variable $P(t)_{XCP}$ that will keep the probability to get a XCP_{hit}, and another one $P(t)_{TCP}$ that will keep the probability to get a TCP_{hit}. By applying the described zombie estimator, we can have an idea about the number of XCP flows $N = {P(t)_{XCP}}^{-1}$, and the number of TCP flows $M = {P(t)_{TCP}}^{-1}$ at a given time t.

Finally, if we know approximately the number of XCP and TCP flows, we can estimate the resources needed by the XCP flows (BW_{XCP}) by mean of Equation 6.1. We have proposed to estimate the resources needed by XCP and TCP at every control interval of the XCP router, which is equal to the average RTT of all incoming XCP packets.

6.3.3 Limiting the TCP throughput

At the same time that we estimate the resources needed by the XCP flows, it is necessary to compute also the real amount of resources taken by XCP. Thus, by comparing both the needed and the real amount of resources taken by XCP, we will be able to make a decision to provide bandwidth fair share.

Computing the real amount of resources taken by XCP

As we said in Chapter 2, for every incoming packet, a XCP router needs to access the IP header, in order to get the following parameters: the packet size, the packet type and the congestion control protocol used. If the congestion control protocol used is XCP, and the packet type reveals that the incoming packet is not an ACK, then the XCP router must access the XCP header and take the following informations: *H_cwnd*, *H_rtt* and *H_feedback*.

All the informations collected by XCP routers will let them compute a per packet feedback. In order to compute this per packet feedback, a XCP router needs to compute an input traffic rate (*in_traffic_rate*) at the end of every control interval (*ctrl_int*), that corresponds to the

average RTT signaled in every incoming data packet ($ctrl_int = rtt$). In fact, a XCP router only makes decisions every control interval.

The *in_traffic_rate* variable is calculated by dividing the total amount of data received by the router during the control interval (*in_traffic*), by the duration of such control interval. *in_traffic* is calculated by adding the packet size of every incoming (data or ACK) packet.

Since the XCP routers already implement a procedure to compute the input traffic rate, to estimate the XCP input traffic rate requires only minor changes in the XCP mechanism. Thus, all we need to do is to add a new variable, *xcp_in_traffic*, that will store the sum of the size of every XCP packet, and divide this variable by the duration of the control interval *ctrl_int*, to get the XCP input traffic rate *xcp_in_traffic_rate*.

Ensuring friendliness between XCP and TCP flows

After calculating the XCP input traffic rate, *xcp_in_traffic_rate*, and the needed XCP bandwidth, BW_{XCP}, we can make decisions to provide bandwidth fair share between XCP and TCP. In order to limit the TCP throughput, with a p_{drop} probability, our XCP-TCP friendliness solution will determine if an incoming TCP packet should be discarded.

p_{drop}, which is initialized with a value of 0.001, is updated at every control interval as follows:

if *xcp_in_traffic_rate* $> BW_{XCP}$, the probability p_{drop} is updated as $p_{drop} = p_{drop} * D_{drop}$, where $0.99 < D_{drop} < 1$. If *xcp_in_traffic_rate* $< BW_{XCP}$, the probability p_{drop} is updated as $p_{drop} = p_{drop} * I_{drop}$, where $1.01 > I_{drop} > 1$. When *xcp_in_traffic_rate* $= BW_{XCP}$, then p_{drop} keeps its last value.

In order to avoid always having p_{drop} with a positive value when our friendliness mechanism only tends to decrease its value, if p_{drop} is below a threshold, low_thresh, we can set $p_{drop} = 0$. When $p_{drop} = 0$ and its value must be increased, then $p_{drop} = 0.001$.

6.3.4 Special remark of our XCP-TCP friendliness mechanism

It is very important to emphasize the fact that the XCP routers will execute the XCP-TCP friendliness mechanism only :

1. when both XCP and TCP protocols have been detected during a control interval and
2. if the total input traffic rate exceeds a given threshold γ, where γ must be slightly smaller than the output link capacity (e.g., 97% of the output link capacity).

We will not execute our XCP-TCP friendliness mechanism when the input traffic rate is smaller than γ, since it means that the current router is not the bottleneck of the flows crossing this point. Concerning the γ parameter, we advise to set γ to a value slightly smaller than the output link capacity, since even in a bottleneck router, the computed rate during a control interval is not always equal to the output link capacity, due to the burstiness nature of the senders. Thus, during a certain control interval we could compute an input traffic rate slightly smaller than the output link capacity, and in the next control interval, see the router dropping packets.

In addition, when the total input traffic rate becomes smaller than our γ threshold, the p_{drop} variable will be reinitialized to the original value (0.001).

Finally, since our friendliness mechanism aims at improving the inter-protocol fairness between XCP and TCP, it is not useful to apply our mechanism when only one of these two protocols, XCP or TCP, have been detected. In fact, both XCP and TCP have already an intra protocol fairness mechanism that we do not desire to influence unnecessarily.

6.4 Testing our XCP-TCP friendliness solution

In order to test our XCP-TCP friendliness solution, we have implemented our propositions in the ns2 XCP modules provided by Dina Katabi. In our simulations, we will focus on 2 different scenarios : national small distance Grid (such as the French Grid5000 [36] infrastructure) with an $RTT \approx 20ms$ and larger scale Grids with an $RTT \approx 100ms$.

The topology used to test our XCP-TCP friendliness solutions will be the one shown in Figure 6.1. However, we will vary every time the number of XCP and TCP flows. In our experiments we have used $D_{drop} = 0.9999$, $I_{drop} = 1.0001$ and $low_thresh = 0.0001$.

6.4.1 1 XCP flow & 2 TCP competing flows

For our first set of experiments, we evaluate the same scenario as Figure 6.2 where an XCP flow competes with two new TCP streams.

Figure 6.3(a) shows the average throughput computed every 10 seconds[1] for XCP and TCP with an $RTT \approx 20ms$, while Figure 6.3(b) shows the average throughput computed every 10 seconds for XCP and TCP with an $RTT \approx 100ms$. In both Figures 6.3(a) ($RTT \approx 20ms$) and 6.3(b) ($RTT \approx 100ms$) we can clearly see the benefits of our XCP-TCP friendliness solution: after a period impacted by the slow start phase of TCP (seconds 10-20), XCP is able to obtain some bandwidth.

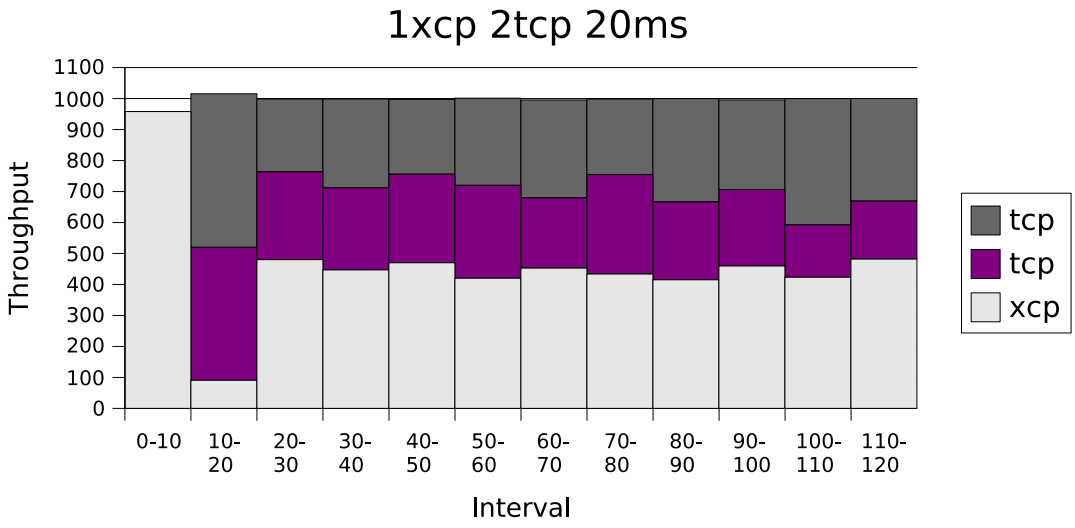
In case when the $RTT \approx 20ms$ (Figure 6.3(a)), it seems that TCP never gets more than 600Mbps, even though the fairness point between TCP and XCP is found at $BW_{XCP} \approx 341.33Mbps$ and $BW_{TCP} \approx 682.66Mbps$. The explanation of this phenomenon is as follows: when our friendliness solution drops TCP packets, TCP flows drastically decrease theirs rate, resulting in a average throughput for the 10-seconds interval lower than the highest TCP throughput ($\sim 700Mbps$). Due to the unstable nature of TCP and the fast evolution of the TCP throughput when the $RTT \approx 20ms$, we think that some fairness metrics, like the Jain index, are not able to correctly reflect the XCP-TCP fairness in a 10-seconds interval.

Figure 6.3(b) shows the same experiment with an $RTT \approx 100ms$. In this case we can observe that after the very aggressive behavior of the TCP slow-start (seconds 10-20), TCP flows are penalized with the execution of our friendliness algorithm. Packet losses added to a very slow increases of the TCP throughput result in a very low throughput of TCP flows. It is very important to remark that after second 20, the drop probability used by our mechanism is at its minimum value, and TCP flows are no longer penalized with dropped packets. Thus, the unfairness that we can see after second 20 is caused mainly by the incapacity of TCP to regain the available bandwidth when RTT is large.

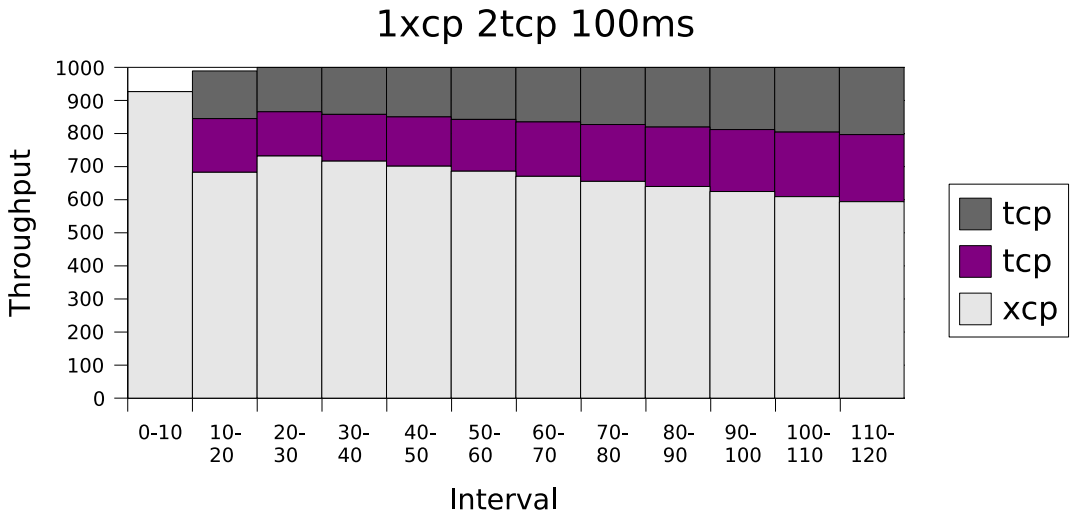
6.4.2 10 TCP flows & 3 XCP competing flows

The impact of the RTT on our XCP-TCP friendliness solution can be decreased when the number of TCP streams increases. In Figure 6.4(b) we can see the results of a simulation

[1] All the remaining simulations results from this chapter will be shown in a similar way



(a) $RTT \approx 20ms$



(b) $RTT \approx 100ms$

Figure 6.3: One XCP flow dealing with two concurrent TCP flows

where 3 XCP flows are incorporated gradually after second 10 (each flow is incorporated with a 20-seconds delay), among 10 TCP flows (all TCP flows are started at second 0), with an $RTT \approx 100ms$.

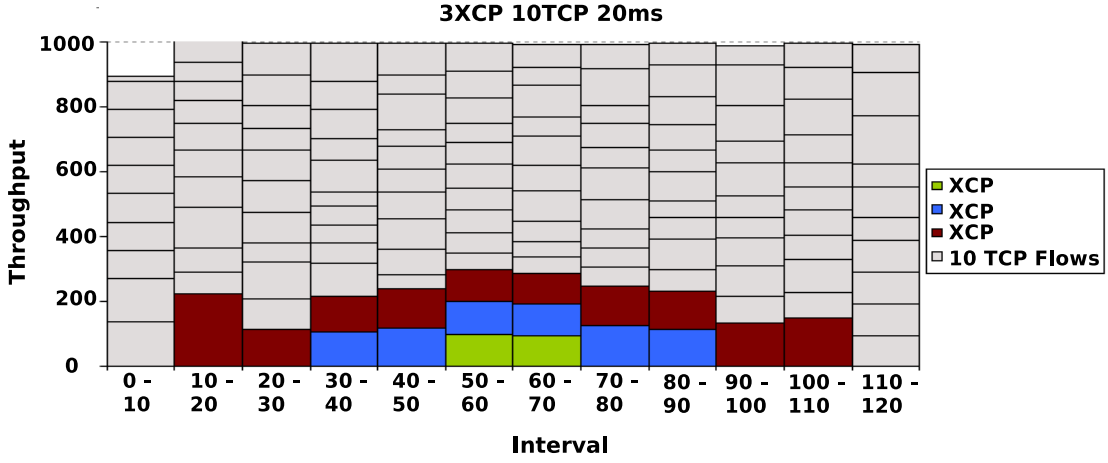
As we can see in Figure 6.4(b) ($RTT \approx 100ms$), every time a new XCP flow comes in, our friendliness mechanism gives some bandwidth to this new flow. However, since the XCP flows are also stopped gradually (one XCP flow is stopped every 20 seconds after second 70) before TCP flows are stopped, the amount of bandwidth assigned to XCP after second 70 is larger than needed. Moreover, in this case, TCP is able to grab the lost bandwidth (approximately 150Mbps between seconds 80 and 100) faster than the case shown in Figure 6.3(b) (less than 20Mbps between seconds 80 and 100). The explanation of this phenomenon is as follows: let imagine a scenario with M TCP flows. Since in steady state every flow increases approximately by 1 packet per RTT, then during a period of time $intv$, the total TCP rate increase will be $M * intv/RTT$ packets. Therefore, if we increase the number of TCP active flow by a factor α, then, with the same RTT value, the total TCP rate increase will be $\alpha * M * intv/RTT$.

However, it is important to remark that even if in the case shown in Figure 6.4(b), TCP grabs bandwidth faster than in Figure 6.3(b), and therefore the friendliness between XCP and TCP is improved, XCP has still more bandwidth than needed. The lowest fairness level is found between second 20 and 30, where one XCP flow gets up to approximately 400Mbps while the remaining 10 TCP flows have to share approximately 600Mbps, even though the fairness point is found at $BW_{XCP} \approx 93Mbps$ and $BW_{TCP} \approx 931Mbps$. The best bandwidth repartition is found between seconds 50 and 60, where 3 XCP flows share around 350 Mbps while 10 TCP flows share almost 650 Mbps ($BW_{XCP} \approx 236.30Mbps$ and $BW_{TCP} \approx 787.70Mbps$).

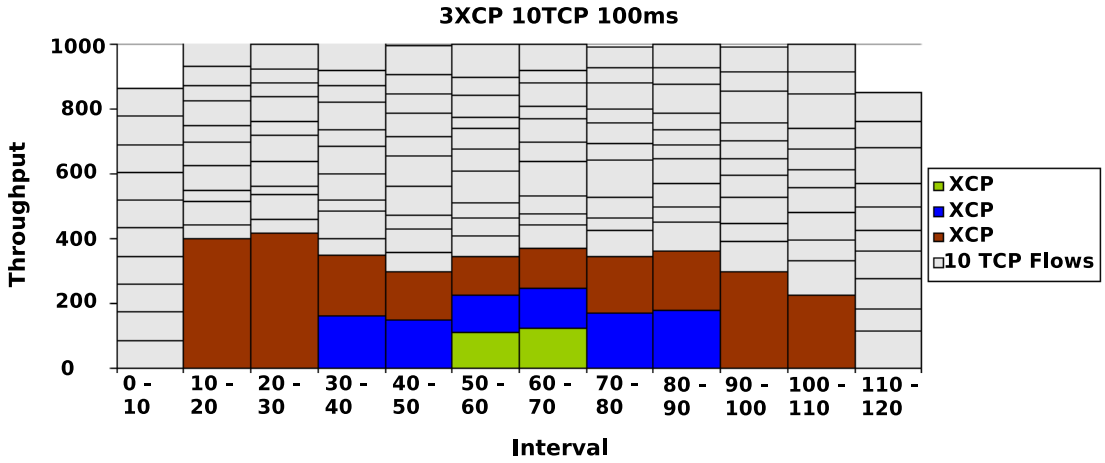
By increasing the number of TCP flows and/or by decreasing the RTT, TCP should become more aggressive. This aggressiveness should be limited by our XCP-TCP mechanism. In order to prove the robustness of our algorithm, we decided to execute the same experiment (3 XCP appearing gradually among 10 TCP flows), but this time using an $RTT \approx 20ms$ (Figure 6.4(a)), that represents an increase of the TCP aggressiveness by a factor $\alpha = 5$, in comparison to the experiment shown in Figure 6.4(b). As we can observe in Figure 6.4(a), the increase in the aggressiveness is well managed by our friendliness mechanism, adapting quickly the p_{drop} value to the new TCP aggressiveness. The lowest fairness level is found between seconds 20 and 30, where one XCP flows gets approximately 200Mbps while the remaining 10 TCP flows have to share almost 800Mbps ($BW_{XCP} \approx 93Mbps$ and $BW_{TCP} \approx 931Mbps$). The best bandwidth repartition is found between seconds 60 and 70, where 3 XCP flows share close to 260Mbps, while 10 TCP flows share approximately 740Mbps ($BW_{XCP} \approx 236.30Mbps$ and $BW_{TCP} \approx 787.70Mbps$).

6.4.3 10 XCP flows & 3 TCP competing flows

We already explored the case where the number of TCP flows is greater than the number of XCP flows, in order to test the robustness of our mechanism when TCP increases its aggressiveness. Now, in this subsection we explore the case where most of the flows are XCP, in order to know whether our mechanism lets TCP compete adequately against XCP flows. In this new set of experiments (the results are shown in Figure 6.5) we gradually incorporated 3 TCP flows after second 10 (each flows is incorporated with a 20-seconds delay), among 10 XCP flows (all XCP flows are started at second 0), with an $RTT \approx 20ms$ (case 1) and an $RTT \approx 100ms$ (case 2).

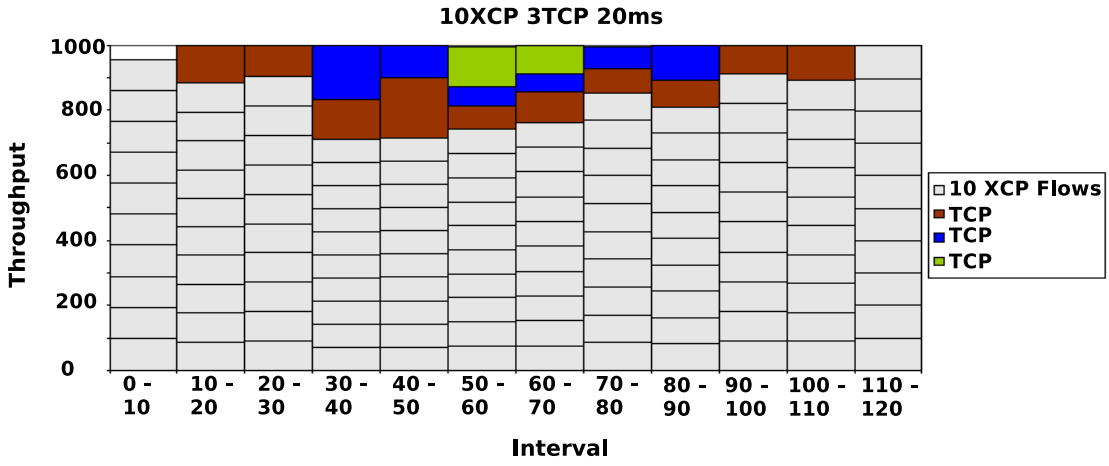


(a) $RTT \approx 20ms$

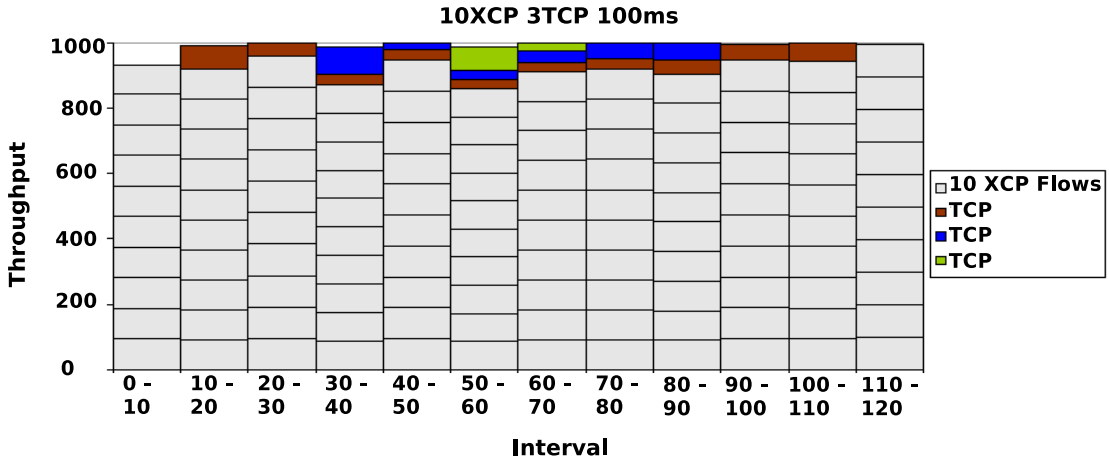


(b) $RTT \approx 100ms$

Figure 6.4: 3 XCP flows appear among 10 TCP flows



(a) $RTT \approx 20ms$



(b) $RTT \approx 100ms$

Figure 6.5: 3 TCP flows appear among 10 XCP flows

The simulation results of the first case are shown in Figure 6.5(a) ($RTT \approx 20ms$), where we can see that every time a new TCP flow starts, it gets more bandwidth than needed (seconds 10, 30, 50). However, after Slow Start finishes, TCP throughput is successfully limited by our XCP-TCP friendliness solution, as we can observe between seconds 20 and 30 and between seconds 50 and 110. Figure 6.5(a) proves also that our mechanism ensures friendliness even though every TCP flow comes in/out asynchronously to the network.

In the case where the $RTT \approx 20ms$, the lowest fairness level is found maybe between seconds 30 and 50, where 2 TCP flows share close to 300Mbps while the remaining 10 XCP flows share around 700Mbps ($BW_{TCP} \approx 170.66Mbps$ and $BW_{XCP} \approx 853.34Mbps$). On the other hand, one of higher fairness level is found between second 60 and 70, where 3 TCP flows share approximately 250Mbps while the remaining 10 XCP flows share around 750Mbps ($BW_{TCP} \approx 236.30Mbps$ and $BW_{XCP} \approx 787.70Mbps$).

When the $RTT \approx 100ms$, at the beginning every TCP flow takes more resources than needed, producing the execution of the XCP-TCP friendliness mechanism. After finishing the Slow-Start phase, due to our friendliness mechanism, the amount of bandwidth taken by TCP is smaller than the maximum allowed bandwidth. Similarly to the case shown in Figure 6.3(b), even if our mechanism does not penalize TCP flows when the TCP throughput is too low, since $p_{drop} \rightarrow 0$, the time needed by TCP to get enough resources is very large (see seconds 60 to 110 in Figure 6.5(b)). We believe therefore that our XCP-TCP friendliness solution benefits mainly long-life flows, which have the time needed to recover the lost bandwidth. As we said in Chapter 1, the solutions proposed in this thesis concern mainly networks with large BDP and where long-life flows are necessary and frequent (our solutions cannot be applied as such in networks like Internet).

In the case where the $RTT \approx 100ms$, the lowest fairness level is found maybe between seconds 60 and 70, where 3 TCP flows share close to 100Mbps while the remaining 10 XCP flows share around 900Mbps ($BW_{TCP} \approx 236.30Mbps$ and $BW_{XCP} \approx 787.70$). On the other hand, one of the higher fairness level is found between second 100 and 110, where 1 TCP flow gets approximately 100Mbps while the remaining 10 XCP flows share around 900Mbps ($BW_{TCP} \approx 93Mbps$ and $BW_{XCP} \approx 931Mbps$).

6.5 Limitations and optimizations

In the XCP-TCP friendliness mechanism presented above, we have used the zombie estimator to get an idea about the number of active flows, as proposed in [71]. This mechanism has the benefit to be lightweight in terms of CPU utilization, since the number of operations is minimum and most of these operations can be executed in parallel. However, the operations of the zombie estimator need to be executed once for every incoming packet. Thus, both the estimation operations and the XCP routers operations could increase significantly the processing time of packets.

For this reason, we have proposed to compute the number of XCP and TCP flows, taking as base only a percent of the total incoming packets. This modification may have a very important impact in the routers. For instance, in a router processing 833,333 packets/s (10 Gbps whether every packet is composed by 1500B), to estimate the number of flows such a router would approximately generate 1,666,666 random numbers; access the zombie table 833,333 times; execute 833,333 comparisons; between others operations like writing the ID flow when a *mis* is declared, etc. Taking only into account the generated random numbers,

the accesses to the zombie table and the comparisons, our 10Gbps routers should execute 3,333,332 operations. Thus, a reduction of 30% in the number of inspected packets should represent a decrease of approximately 1,000,000 operations in the router.

In order to estimate the active flow number without checking every incoming packet, a few changes in the *hit* probability equation are necessary. As shown in Equation 2.28, from Chapter 2, the *hit* probability equation is given by:

$$P(t) = (1 - \alpha)P(t - 1) + \alpha.Hit(t)$$

where α reflects the probability to get a packet from the zombie table with the same ID as the incoming packet. This way, α depends on the probability to insert an ID in the zombie table (25%) and the probability to choose this ID from the zombie table storing 1000 IDs (1/1000). Therefore $\alpha = 0.00025$.

When we do not check every incoming packet, α must be modified according to the new conditions. Since in this thesis, we propose to check only 50% of the total incoming packets, α will depend on the probability to check an incoming packet (50%), on the probability to insert an ID in the zombie table (25%) and on the probability to choose this ID from the zombie table storing 1000 IDs (1/1000). Therefore $\alpha = 0.000125$. However, verifying only 50% of the total incoming packets and updating the zombie table with a low probability (25%) increases significantly the probability of missing flows in our zombie table. On the other hand, to increase the updating probability of the zombie table should decrease the problem of missing flows. For these reasons, we have decided to use a probability of 50% to update the zombie table in case of *mis*.

With the modification introduced in our friendliness mechanism (that allows routers to monitor only 50% of incoming packets to estimate the number of flows), we reexecuted the experiment shown in the Subsection 6.4.3, where 3 TCP flows are incorporated gradually among 10 active XCP flows. Figure 6.6 shows the results of our new set of experiments.

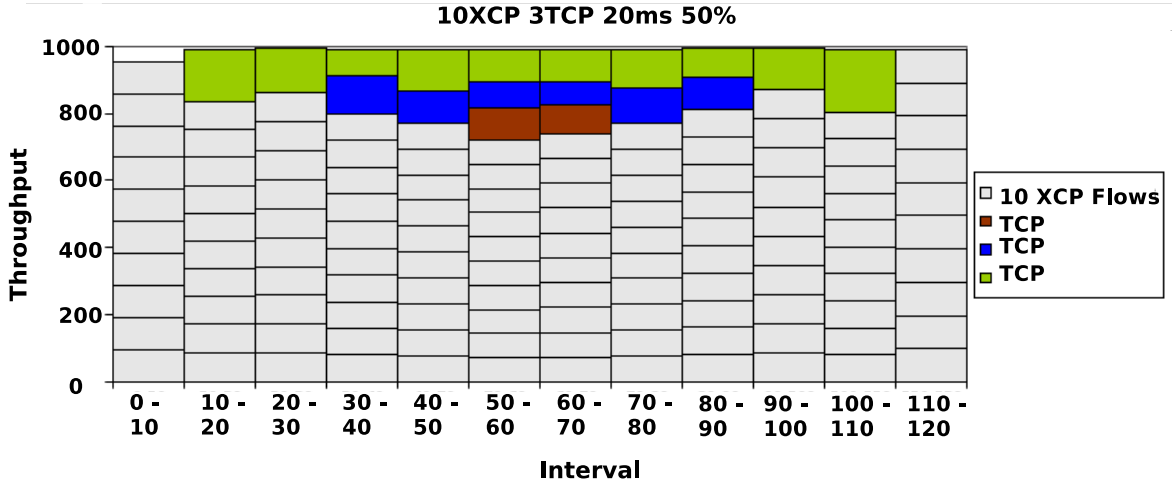
As we can see in Figure 6.6, our XCP-TCP friendliness mechanism successfully provides a good bandwidth repartition by inspecting only 50% of the total incoming packets. In addition, if we compare (*i*) Figure 6.6(a) ($RTT \approx 20ms$ and 50% of inspected packets) with Figure 6.5(a) ($RTT \approx 20ms$ and 100% of inspected packets), and (*ii*) Figure 6.6(b) ($RTT \approx 100ms$ and 50% of inspected packets) with Figure 6.5(b) ($RTT \approx 20ms$ and 100% of inspected packets); we can see that the results are very similar. Thus, we show that we can reduce significantly the operations executed by our XCP-TCP friendliness solution, while keeping the similar fairness levels shown in the simulations of Section 6.4 (where we inspected 100% of packets).

6.6 Conclusion

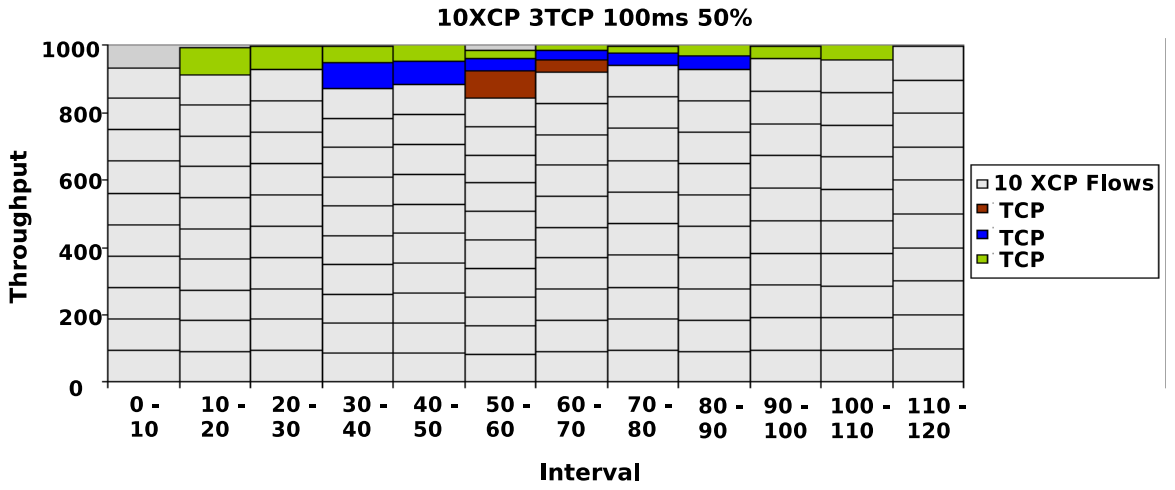
In the current chapter we have discussed the problems suffered by XCP when it shares the available resources with TCP. This problem is caused mainly by two factors:

1. To avoid congestion, XCP only assigns the available bandwidth to the XCP senders, where the available bandwidth is computed as the difference between the output link capacity and the input traffic rate.

2. The lack of mechanisms for controlling flows using any E2E protocol.



(a) $RTT \approx 20ms$



(b) $RTT \approx 100ms$

Figure 6.6: 3 TCP flows appear among 10 XCP flows - inspecting 50% of packets

As a result of these factors, TCP flows grab as many resources as needed to the detriment of XCP flows. It is very important to note that most of ERN protocols are also affected by both factors mentioned above, and that they are therefore also unable to compete adequately against E2E protocols.

For this reason, in the current chapter we presented a solution that provides friendliness between XCP and TCP, which is independent of the XCP protocol and hence easy to apply to any ERN protocol. Our XCP-TCP friendliness method lies on the execution of two main mechanism. The first mechanism estimates the resources needed by the XCP and non-XCP flows. The second mechanism limits the TCP throughput, on the basis of the information given by the resources estimation mechanism. While the second mechanism is based on a BLUE-like probabilistic mechanism to drop packets, which is light in terms of CPU utilization (one multiplication every XCP control interval to update the drop probability and one comparison for every incoming packet), the first mechanism can introduce an important number of operations at 1-10Gbps rates.

Thus, we have proposed a solution that estimates the resources needed by XCP and TCP inspecting only 50% of the total incoming packets, instead of 100% of the incoming packets (as originally proposed in [71]). As a result, our friendliness mechanisms can reduce significantly the number of operations executed during the estimation process in a 1-10Gbps router.

The experiments made in this chapter, to test our XCP-TCP friendliness solution, proved that our algorithms successfully "allocate" and "deallocate" bandwidth dynamically to XCP, in order to ensure the friendliness between XCP and TCP flows. However, the level of fairness in the bandwidth repartition between XCP and TCP does not fully depend on our friendliness mechanism. In fact, the bandwidth repartition between XCP and TCP is strongly influenced by some parameters that are beyond the control of our XCP-TCP friendliness method, like the RTT of the TCP flows.

Thus, as the RTT of TCP decreases, the fairness between TCP and XCP increases, as shown in our experiments (Section 6.4). The origin of this phenomenon is found in the fact that, when TCP flows take more bandwidth than needed and our friendliness mechanism drops one of their packets, TCP senders react halving their rate. This way, the resulting total rate of TCP flows can be too low in comparison to the maximum allowed throughput. The capacity of TCP to regrab the lost bandwidth will depend on the RTT of TCP flows and the number of TCP flows.

In order to decrease the unfairness between XCP and TCP due to the RTT, some mechanisms can be added or changed in our XCP-TCP friendliness solutions. For instance, we could “evaluate” the TCP aggressiveness when the TCP throughput increases, in order to update the drop probability in a more intelligent way.

Finally, we want to remark that our XCP-TCP friendliness solution represents the last of a set of solutions that we have proposed to solve the inter-operability problems of ERN protocols with mechanisms and protocols currently used in heterogeneous networks. Thus, we aimed at stimulating the deployment of the ERN protocols in heterogeneous large BDP networks and/or the creation of ERN protocols that adapt our propositions, to make them inter-operable with current network mechanisms and protocols.

Chapter 7

Conclusions

7.1 Context and problems studied in this thesis

The work presented along this thesis concerns mainly:

- IP wired networks with large delays and large bandwidth, which are produced by the evolution in network technologies as well as large scale interconnection (Chapter 1).
- Long life flows (movements of bulk data over IP networks) which are part of modern computational science applications, where instrument raw data, large computational data sets and visualization data must be exchanged between institutions.

This thesis aimed at improving the performance of Long-life flows in LDHP networks, since currently the movement of bulk data transfer is difficult. This problem lies on the fact that :

1. TCP, which is the more deployed and the best known of congestion control protocols, has poor performance in LDHP networks.
2. TCP-like protocols for LDHP networks introduce inter and intra protocol unfairness between flows due to their aggressiveness (Chapter 2).
3. the responsiveness of TCP-like protocols for LDHP networks is slow when the resources in the network change over time (Chapter 3).

The problems of unfairness and responsiveness of TCP-like protocols for LDHP networks are based on the fact that they are unable to correctly estimate the state of the network, since their mechanisms are only implemented in the end host (E2E protocols). AQM mechanisms, to be implemented in routers, have then been proposed to better manage the network resources. However, the benefits of the AQM mechanisms are limited (Chapter 2).

7.2 Weaknesses and strengths of ERN protocols

7.2.1 Benefits of any ERN protocol in LDHP Networks

As we have seen in Chapter 2, most of ERN protocols are able to adapt their sending rate to the maximum allowed bandwidth in only one RTT [3, 4]. This adaptation depends only on

the time needed by the routers to compute how the resources must be allocated between the responsive flows in order to get a max-min fairness criterion [1] .

Thus, ERN senders are able to grab and give back bandwidth, following perfectly the evolution of the available bandwidth, while suffering fewer losses than E2E protocols. Additionally, ERN flows remains stable in the intersection point between the fairness and the efficiency line [1] (Chapters 2, 4 and 5). Thus, ERN protocols behave better than any E2E protocol.

Having congestion control protocols which implement ERN can provide huge benefits to the throughput of long-life flows in LDHP networks, while keeping high fairness level between flows. Finally, as a result of a more accurate usage of network resources, congestion problems should decrease.

7.2.2 Why are ERN protocols not used in current networks?

In spite of the benefits that ERN protocols could provide to bulk data transfers in LDHP networks, we have shown that ERN protocols, such as they are proposed, cannot be used in current IP networks due to their heterogeneous nature.

The problems of the ERN protocols, that prevent their deployment in IP networks, lies in the absence of mechanisms enabling the inter-operability of the ERN protocols with different hardware devices (like DropTail routers) and protocols (like E2E protocols), which are currently widely used in IP networks.

Thus, this thesis presented our solutions concerning the inter-operability problems of ERN protocols in heterogeneous LDHP networks. Our researches have covered three main issues which had not been explored before :

1. the inter-operability of ERN protocols with non-ERN routers;
2. the inter-operability of ERN protocols with E2E protocols;
3. the improvement of the robustness of ERN protocols.

7.3 Propositions for a robust and inter-operable ERN protocol

7.3.1 Enabling the inter-operability with non-ERN routers

When the bottleneck of a flow using any ERN protocol is composed by non-ERN routers, then the throughput of such a flow can become very unstable.

Therefore, we proposed a solution which enables the inter-operability of ERN protocols with non-ERN-capable routers, by extending the capabilities of ERN routers. Our solution detects non-ERN routers (non-ERN clouds), estimates the available bandwidth between two ERN routers surrounding a non-ERN cloud and creates ERN virtual routers, which replace every non-ERN cloud by an ERN router.

In our simulations (made by implementing our propositions into the *ns* XCP modules provided by Dina Katabi), our solutions proved to be able to correctly maximize the utilization and to fairly share the resources between ERN flows in a wide range of topologies covering different scenarios of gradual deployment of ERN protocols.

[1] Except Quickstart, which does not provide fairness.

Our mechanism to detect non-ERN routers, which is based in a dual-TTL strategy, does not keep any states in the ERN routers. In Chapter 4, we have explained why our algorithm to detect non-ERN routers should work even in the presence of MPLS domains or IPsec tunnels. Thus, while we have extensively analyzed and proposed a solution to the problem of non-ERN layer-3 devices, the problems of non-ERN layer-2 devices remains open.

Layer-2 devices, like Ethernet switches, could also present bottlenecks. These devices cannot be detected by our non-ERN cloud detection algorithm, since our algorithm requires a change in the TTL IP header, which remains unchanged when packets are forwarded by Ethernet and SONET switches. Thus, detecting switches between routers or end hosts and routers can be a difficult task. Some studies explore the communications between layer-2 devices and end hosts [121]. These approaches could allow interaction between layer-2 devices and ERN protocols.

Concerning the estimation bandwidth of non-ERN clouds, we provided a brief discussion about the bandwidth estimation speed and accuracy offered by some of the available bandwidth estimation algorithms, showing why using such bandwidth estimation techniques is a viable option. Thus, if available bandwidth estimation tools are efficient, why can we not just use such tools in combination with TCP in senders? The answer is simple: since TCP senders do not have idea about the number of flows incoming and leaving the network, TCP senders are not able to fairly share the estimated available bandwidth with others TCP senders. This problem does not happen in our solution, since we create an ERN virtual router, which reuses the code already available in the ERN real routers, that maximizes and fairly shares the estimated available bandwidth between active flows. In addition, since we estimate the available bandwidth only between two ERN routers and not between end hosts, the accuracy of the bandwidth estimation algorithm increases.

In order to create virtual routers, an ERN router must keep a hash table with the sources address of the neighbor ERN routers and the estimated bandwidth between its neighbor and itself. ERN routers do not keep states per flow.

We wish to remark finally that the use of the mechanisms added to the ERN routers, to interact correctly with non-ERN routers, can offer limited advantages to the performance of short-life flows, like those generated by HTTP or SSH. However, the use of our mechanisms can have great benefits for long-life flows, which transfer large amount of data.

7.3.2 Enabling the inter-operability with E2E protocols

Enabling the inter-operability of ERN protocols with existing E2E protocols (such as TCP) can strongly stimulate the deployment of ERN protocols in IP networks. However, most of ERN protocols are unable to cohabit adequately with E2E protocols: ERN protocols take or give all the resources to E2E protocols.

In this thesis, we have proposed a solution that provides friendliness between ERN and E2E protocols. Our friendliness solution is based on the estimation of the resources needed by the ERN and E2E flows, as well as a random drop mechanism to limit the rate of E2E protocols. The mechanisms implemented in our friendliness mechanisms can be executed in parallel, minimizing the additional delay in the treatment of packets, is light in terms of CPU utilization and does not keep any state per flow.

In our simulations, we have observed that due to the TCP algorithm that forces flows to halve their rate when losses occur, every time the TCP throughput was limited by our friendliness mechanism, the final rate of TCP was lower than the maximum allowed rate.

Later, the time needed by TCP to get its maximum allowed rate depended strongly from the RTT value. As a result, our friendliness solution could give XCP more resources than TCP in average. However, replacing TCP by a high speed version of TCP (e.g. HSTCP), this "problem" of wrong bandwidth repartition should be reduced.

On the other hand, using HSTCP could require increasing the drop probability faster than in the case of TCP. In addition, in IP networks, we could have any combination of standard TCP and HSTCP flows. Thus, in order to improve our friendliness solution, we believe that the drop probability should be increased in an elastic way, according to the acceleration of the input traffic of E2E flows (a new module to estimate the aggressiveness of E2E should be implemented). Thus, more research concerning the friendliness between ERN protocols and E2E protocols are needed.

7.3.3 Improving the robustness of ERN protocols

Due to the burstiness nature of sender in current IP networks, the losses of packets are frequent. On the other hand, the aggregation/disaggregation of high-priority flow can produce VBE. Thus, for ERN protocols, IP networks can represent hostile lossy VBEs.

For this reason, we have proposed a new architecture that improves robustness of ERN protocols in presence of ACK (feedback) losses. This architecture, that we have called the r architecture, is based on the migration of some parts of the code from the sender to the receiver. Our r architecture keeps the ERN routers code untouched. It just needs to add a flag in the ERN packet header to synchronize the sender and the receiver when losses of data packet occur.

For validating our r architecture, we have implemented our proposition into the *ns* XCP modules provided by Dina Katabi. In the simulations results, we found that in case of feedback (ACK) losses in VBE, the r architecture lets ERN senders obtain a more accurate information of the network state, improving the stability, the performance and the fairness of flows.

7.4 Generalization of our propositions

The solutions presented in this thesis were tested on the XCP protocol. However, our solutions can be applied to most ERN protocols.

For instance, in the case of the inter-operability between ERN protocols and non-ERN routers, both the strategy for detecting the non-XCP clouds and the available bandwidth estimations are totally independent of the mechanisms of the ERN protocols, and therefore they can be implemented in any ERN router. Concerning the ERN virtual router, the heart of such virtual router is the estimated available bandwidth, as in most of ERN protocols (e.g. JetMax [4], Quickstart [79], TCP MaxNet [78], etc.). Therefore, once having the estimated available bandwidth, we could compute a feedback adapted to any ERN protocol by only reusing the available code in its routers.

The set of mechanisms proposed in our friendliness algorithm are also independent of the mechanisms of the ERN protocols. Therefore, such mechanisms can be implemented in any ERN router. However, our friendliness solution has been created to provide friendliness when the performance of ERN flows are degraded by E2E flows, like XCP [3], JetMax [4], and some others. When the performance of E2E flows are degraded by the aggressiveness of ERN protocols, like CADPC/PTP [77], our friendliness solution does not offer any advantage.

Finally, our proposed r architecture for ERN protocols can be used in any ERN protocol. Since all the ERN protocols provide information to the senders by mean of the ACK to avoid introducing unnecessary overhead, and that losses in IP heterogeneous networks are frequent, we believe that the r architecture must be implemented in every proposed ERN protocol.

Today, there are not yet any "winner" ERN protocol. For this reason, all our propositions have been specially designed with a flexible architecture to be applicable to most or all ERN protocols. This way, we propose a set of solutions providing fundamental functionalities to stimulate the deployment of ERN protocols in heterogeneous large BDP networks for the benefit of the general performance of long-life flows.

7.5 Perspectives

Our propositions to enable the inter-operability of ERN protocols in heterogeneous large BDP networks have been extensively analyzed theoretically and validated by simulation. As we have seen in Section 7.3, they can be enriched by taking into account others factors.

As future works, our propositions should be tested on real networks after an implementation process and engineering step. Thus, some perspectives could consist on implementing our propositions in Linux-based systems by extending for instance, the capabilities of the XCP [3] or JetMax [4] protocols. In a first step, we could then test our propositions in a managed testbed (which emulates large BDP networks).

Once validated the implementation of inter-operability solutions, the last step should be to test our propositions in a Wan Area Network (e.g. Grid'5000 [36]). This deployment should require to analyze the benefits of our mechanisms by comparing the average throughput, the throughput evolution, packet losses, and fairness of several E2E protocols and our chosen ERN protocol in bulk data transfers scenarios.

Bibliography

[1] Tom Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, 2003.

[2] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *INFOCOM*, 2004.

[3] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *ACM SIGCOMM*, 2002.

[4] Derek Leonard Yueping Zhang and Dmitri Loguinov. JetMax: Scalable Max-Min Congestion Control for High-Speed Heterogeneous Networks. In *INFOCOM*, April 2006.

[5] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[6] The GEANT project: http://www.geant.net/.

[7] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

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

[9] Larry Dunn. Three sides of a coin: Why it's easy/impossible/tricky to get your ideas into routers. Protocols for Fast Long-Distance Networks (PFLDnet) 2007. http://wil.cs.caltech.edu/pfldnet2007/slides/dunn3sidesofacoin.pdf.

[10] ns2. The Network Simulator. In *http://www.isi.edu/nsnam/ns/index.html*, 2007.

[11] H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1988.

[12] Amos E. Joel. *Asynchronous Transfer Mode Switching.* IEEE, December 1993.

[13] Cisco. Frame Relay: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/frame.htm.

[14] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.

[15] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[16] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[18] Eric He, Jason Leigh, Oliver T. Yu, and Thomas A. DeFanti. Reliable blast udp: Predictable high performance bulk data transfer. In *CLUSTER*, pages 317–324, 2002.

[19] S. Wallace. Tsunami file transfer protocol. In *Proc. of First Int. Workshop on Protocols for Fast Long-Distance Networks*, CERN, Geneva, Switzerland, February 2003.

[20] Nageswara S. V. Rao and Qishi Wu and Steven M. Carter and William R. Wing. High-speed dedicated channels and experimental results with hurricane protocol. *Annals of Telecommunications*, 61(1–2):21–45, 2006.

[21] D. Velten, R.M. Hinden, and J. Sax. Reliable Data Protocol. RFC 908 (Experimental), July 1984. Updated by RFC 1151.

[22] C. Partridge and R.M. Hinden. Version 2 of the Reliable Data Protocol (RDP). RFC 1151 (Experimental), April 1990.

[23] Van Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[24] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), November 2000.

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

[26] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582 (Experimental), April 1999. Obsoleted by RFC 3782.

[27] D.-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Comput. Netw. ISDN Syst.*, 17(1):1–14, 1989.

[28] A. Akella, S. Seshan, S. Shenker, and I. Stoica. Exploring Congestion Control, 2002.

[29] Narayanan Venkitaraman, Tae eun Kim, and Kang-Won Lee. Design and Evaluation of Congestion Control Algorithms in the Future Internet. In *Measurement and Modeling of Computer Systems*, pages 212–213, 1999.

[30] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[31] Z. Wang and J. Crowcroft. A New Congestion Control Scheme: Slow Start and Search (Tri-S). *ACM Computer Communication Review, SIGCOMM*, 21(1):32–43, 1991.

[32] R. Jain. A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks. *ACM Computer Communication Review.*, 19(5):56–71, 1989.

[33] O. Ait-Hellal and E. Altman. Analysis of TCP Vegas and TCP Reno, 1997.

[34] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean C. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *INFOCOM (3)*, pages 1556–1563, 1999.

[35] Chelsio. 10 Gigabit Ethernet iSCSI in Action. White paper.

[36] Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: a Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *Grid'2005 Workshop*, Seattle, USA, November 2005. IEEE/ACM.

[37] IEEE-USA Committee on Communications and Information Policy. Providing Ubiquitous Gigabit Networks in the United States. White paper.

[38] W.D. Lvancic. Architecture Study of Space-Based Satellite Networks for NASA Missions. In *Aerospace Congerence IEEE*, volume 3, pages 1179–1186, March 2003.

[39] Bharat K. Bhargava, Xiaoxin Wu, Yi Lu, and Weichao Wang. Integrating Heterogeneous Wireless Technologies: A Cellular Aided Mobile Ad Hoc Network (CAMA). *MONET*, 9(4):393–408, 2004.

[40] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[41] Michael Weltz, Eric He, Pascale Vicat-Blanc Primet, and M. Goutelle. Survey of Protocols other than TCP. Technical report, Global Grid Forum, April 2005. GFD 57.

[42] Yunhong Gu and Robert L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Comput. Netw.*, 51(7):1777–1799, 2007.

[43] Yunhong Gu and Robert L. Grossman. SABUL: A Transport Protocol for Grid Computing. *Journal of Grid Computing,*, 1:377–386, 2004.

[44] Vern Paxson. End-to-end internet packet dynamics. *IEEE/ACM Trans. Netw.*, 7(3):277–292, 1999.

[45] Qishi Wu and Nageswara S. V. Rao. A class of reliable UDP-based transport protocols based on stochastic approximation. In *INFOCOM*, pages 1013–1024, 2005.

[46] Václav Dupač. A Dynamic Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 36(6):1695–1702, December 1965.

[47] Kazumi Kumazoe, Katsushi Kouyama, Masato Tsuru, and Yuji Oie. Transport Protocols for Fast Long-Distance Networks: Evaluation of their Penetration and Robustness on JGNII. In *Proceeding of Third International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet) 2005*, February 2005.

[48] S. Floyd. Limited Slow-Start for TCP with Large Congestion Windows. RFC 3742 (Experimental), March 2004.

[49] The Linux Kernel Archives: http://www.kernel.org/.

[50] Injong Rhee and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. In *International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[51] D.J. Leith, R. N. Shorten, and G. McCullagh. Experimental Evaluation of Cubic-TCP. In *Proceedings of Fifth International Workshop on Protocols for Fast Long-Distance Networks*, Los Angeles, California, USA, February 2007.

[52] D. H. Choe and S. H. Low. Stabilized Vegas. In *39th Annual Allerton Conference on Communication, Control, and Computing*, October 2002.

[53] David X. Wei Cheng Jin and Steven H. Low. FAST TCP: Motivation, Architecture, Algortihms, Performance. In *INFOCOM*. IEEE, March 2004.

[54] Kun Tan, Jingmin Song, Qian Zhang, and Murari Shridaran. Compound TCP: An Scalable and TCP-Friendly Congestion Control for High-Speed Networks. In *IEEE Infocom*, Barcelona, Spain, April 2006.

[55] Windows Vista: http://www.microsoft.com/windows/products/windowsvista/.

[56] Shao Liu, Tamer Başar, and R. Srikant. TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks. In *Valuetools '06: Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, New York, NY, USA, 2006. ACM Press.

[57] R. N. Shorten and D. J. Leith. H-TCP: TCP for High-Speed and Long-Distance Networks. In *Proceedings of the Second PFLDNet Workshop*, Argonne, Illinois, USA, 2004.

[58] Saad Biaz and Nitin H. Vaidya. Is the Round-Trip Time Correlated With the Number of Packets in Flight? In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 273–278, New York, NY, USA, 2003. ACM Press.

[59] Jim Martin, Arne Nilsson, and Injong Rhee. Delay-Based Congestion Avoidance for TCP. *IEEE/ACM Transactions on Networking*, 11(3):356–369, 2003.

[60] R. S. Prasad, M. Jain, and C. Dovrolis. On the Effectiveness of Delay-Based Congestion Avoidance. In *Proceedings of Second International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[61] S. Mascolo et al. TCP Westwood: Congestion Window Control Using Bandwidth Estimation. In *IEEE Globecom 2001*, volume 3, pages 1698–1702, November 2001.

[62] A. Dell'Aera, L. A. Grieco, and S. Mascolo. Linux 2.4 Implementation of Westwood+ TCP with Rate-Halving: A Performance Evaluation over the Internet. In *IEEE International Conference on Communication*, June 2004.

[63] S. Mascolo and G. Racanelli. Testing TCP Westwood+ over Transatlantic Links at 10 Gigabit/second rate. In *Third International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[64] D. Tennenhouse, S. Garland, L. Shrira, and M. Kaashoek. From Internet to ActiveNet, 1996.

[65] David J. Wetherall, John Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE Openarch*, April 1998.

[66] Laurent Lefèvre and Jean-Patrick Gelas. *Programmable Networks for IP Service Deployment*, chapter 14 on "High Performance Execution Environments", pages 291–321. Artech House Books, UK, isbn 1-58053-745-6; edition, May 2004.

[67] The FAIN Project: http://www.ist-fain.org/.

[68] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[69] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *INFOCOM (2)*, pages 942–951, 2000.

[70] W. Feng, K. Shin, D. Kandlur, and D. Saha. The BLUE Active Queue Management Algorithms, August 2002.

[71] Teunis J. Ott, T. V. Lakshman, and Larry H. Wong. SRED: Stabilized RED. In *INFOCOM*, pages 1346–1355, 1999.

[72] M. Kwon and S. Fahmy. A Comparison of Load-Based and Queue-Based Active Queue Management Algorithms, August 2002.

[73] Zheng Youquan, Lu Mingquan, and Feng Zhenming. Performance Evaluation of Adaptive AQM Algorithms in a Variable Bandwidth Network. *IEICE Transactions on Communications*, 86(6):2060–2067, 2003.

[74] Victor Firoiu and Marty Borden. A Study of Active Queue Management for Congestion Control. In *INFOCOM (3)*, pages 1435–1444, 2000.

[75] Yong Xia, Lakshminarayanan Subramanian, Ion Stoica, and Shivkumar Kalyanaraman. One More Bit is Enough. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 37–48, New York, NY, USA, 2005. ACM Press.

[76] ERICA: Explicit Rate Indication for Congestion Avoidance in ATM Networks. United States Patent 5805577. http://www.freepatentsonline.com/5805577.html.

[77] Michael Welzl. *Scalable Performance Signalling and Congestion Avoidance*. PhD thesis, University of Innsbruck, November 2002.

[78] Bartek Wydrowskilete Martin Suchara, Ryan Witt. TCP MaxNet: Implementation and Experiments on the WAN in Lab. In *IEEE International Conference on Networks*, November 2005.

[79] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-Start for TCP and IP. RFC 4782 (Experimental), January 2007.

[80] Dino M. Lopez Pacheco, Cong-Duc Pham, and Laurent Lefevre. XCP-i : eXplicit Control Protocol for Heterogeneous Inter-Networking of High-Speed Networks. In *Globecom 2006*, San Francisco, California, USA, November 2006.

[81] Dino M. Lopez Pacheco, Laurent Lefevre, and Cong-Duc Pham. Fairness Issues When Transferring Large Volume of Data on High Speed Networks With Router-Assisted Transport Protocols. In *High Speed Networks Workshop 2007, in conjunction with IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.

[82] Dino M Lopez-Pacheco and Congduc Pham. Robust Transport Protocol for Dynamic High-Speed Networks: Enhancing the XCP Approach. In *Proceedings of IEEE International Conference on Networks*, volume 1, pages 404–409, Kuala Lumpur, Malaysia, November 2005.

[83] S. H. Low, Lachlan L. H. Andrew, and Bartek P. Wydrowski. Understanding XCP: Equilibrium and Fairness. In *Proceedings of IEEE INFOCOM*, pages 1025–1036, Miami, FL, March 2005.

[84] Y. Zhang and T. R. Henderson. An Implementation and Experimental Study of the eXplicit Control Protocol (XCP). In *INFOCOM*, pages 1037–1048, 2005.

[85] Steven H. Low, Fernando Paganini, Jiantao Wang, and John C. Doyle. Linear Stability of TCP/RED and a Scalable Control. *Computer Networks*, 43(5):633–647, 2003.

[86] C. Barakat. TCP/IP Modeling and Validation. *IEEE Network*, 15(3):38–47, 2001.

[87] P. Gevros, J. Crowcroft, P. Kirstein, and S. Bhatti. Congestion Control Mechanisms and the Best Effort Service Model. *IEEE Network*, 15(3):16–25, 2001.

[88] Matthew Mathis, Jeffrey Semke, and Jamshid Mahdavi. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[89] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP Throughput: A Simple Model and its Empirical Validation. *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 303–314, 1998.

[90] T. Dunigan, M. Mathis, and B. Tierney. A TCP Tuning Daemon, 2002.

[91] Antony Antony, Johan Blom, Cees de Laat, Jason Lee, and Wim Sjouw. Microscopic Examination of TCP Flows over Transatlantic Links. *Future Generation Computer Systems*, 19(6):1017–1029, 2003.

[92] C. Barakat, E. Altman, and W. Dabbous. On TCP Performance In a Heterogeneous Network: A Survey. *Communications Magazine IEEE*, 38(Communications Magazine):40–46, 2000.

[93] G. Hasegawa and M. Murata. Survey on Fairness Issues in TCP Congestion Control Mechanisms, 2001.

[94] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209 (Proposed Standard), December 2001. Updated by RFCs 3936, 4420, 4874.

[95] Christer Bohm, Markus Hidell, Per Lindgren, Lars H. Ramfelt, and Peter Sjödin. Fast Circuit Switching for the Next Generation of High Performance Networks. *IEEE Journal on Selected Areas in Communications*, 14(2):298–305, 1996.

[96] Lars Gauffin, Lars Håkansson, and Björn Pehrson. Multi-Gigabit Networking Based on DTM: A TDM Medium Access Technique With Dynamic Bandwidth-Allocation. *Computer Networks and ISDN Systems*, 24(2):119–130, 1992.

[97] Nageswara S.V. Rao and Jianbo Gao and Leon O. Chua. On dynamics of transport protocols over wide-area Internet connections. In *Complex Dynamics in Communication Networks*, pages 69–101, 2005.

[98] Yang Richard Yang and Min Sik Kim and Simon S. Lam. Transient Behaviors of TCP-friendly Congestion Control Protocols. In *INFOCOM*, pages 1716–1725, 2001.

[99] C. Zhang and V. Tsaoussidis. The interrelation of tcp responsiveness and smoothness in heterogeneous networks, 2002.

[100] D. Dutta and Y. Zhang. An Active Proxy Based Architecture for TCP In Heterogeneous Variable Bandwidth Networks. In *GLOBECOM*, San Antonio, Texas, USA, November 2001. IEEE.

[101] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard), January 2001.

[102] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta. MPLS Label Stack Encoding. RFC 3032 (Proposed Standard), January 2001. Updated by RFCs 3443, 4182.

[103] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas. LDP Specification. RFC 3036 (Proposed Standard), January 2001.

[104] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen. Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. RFC 3270 (Proposed Standard), May 2002.

[105] P. Agarwal and B. Akyol. Time To Live (TTL) Processing in Multi-Protocol Label Switching (MPLS) Networks. RFC 3443 (Proposed Standard), January 2003.

[106] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005.

[107] Alok Shriram et al. Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. In *PAM*, 2005.

[108] V. Ribeiro. PathChirp: Efficient Available Bandwidth Estimation for Network Path. In *PAM*, 2003.

[109] M. Jain and C. Dovrolis. Pathload: An Available Bandwidth Estimation Tool. In *PAM*, 2002.

[110] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. In *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, volume 21(6), August 2003.

[111] George Kola and Mary K. Vernon. QuickProbe: Available Bandwidth Estimation In Two RoundTrips. *SIGMETRICS Performance Evaluation Review*, 34(1):359–360, 2006.

[112] Cisco Line Cards: http://www.cisco.com/en/us/products/hw/modules/ps2710/products_data_sheets_list.html.

[113] Curtis Villamizar and Cheng Song. High Performance TCP In ANSNET. *SIGCOMM Computer Communication Review*, 24(5):45–60, 1994.

[114] T. Gyires. Using Active Networking for Congestion Control In High-Speed Networks With Self-Similar Traffic. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 405–410, October 2000.

[115] Will E. Leland, Walter Willinger, Murad S. Taqqu, and Daniel V. Wilson. On the Self-Similar Nature of Ethernet Traffic. *SIGCOMM Computer Communication Review*, 25(1):202–213, 1995.

[116] M. Taqqu, V. Teverovsky, and W. Willinger. Estimators for Long-Range Dependence: An Empirical Study, 1995.

[117] The ns Manual: 21.2.2 XCP router. http://www.isi.edu/nsnam/ns/doc/node241.html.

[118] Ed. S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166 (Informational), March 2008.

[119] Burton H. Bloom. Space/Time Trade-Offs In Hash Coding With Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[120] Jung-Shian Li and Yong-Shun Su. Random Early Detection With Flow Number Estimation and Queue Length Feedback Control. *Journal of Systems Architecture*, 52(6):359–372, 2006.

[121] Jinjing Jiang and Raj Jain. Analysis of Backward Congestion Notification (BCN) for Ethernet In Datacenter Applications. In *IEEE INFOCOM 2007 Minisymposium*, Anchorage, AK, USA, May 2007.

Acronyms

ACK	Acknowledgment.
ADIADD	ADaptive Increase ADaptive Decrease.
AIAD	Additive Increase Additive Decrease.
AIMD	Additive Increase Multiplicative Decrease.
AQM	Active Queue Management.
ATM	Asynchronous Transfer Mode.
BDP	*bandwidth-delay* product.
BIC	Binary Increase TCP.
CADPC/PTP	Congestion Avoidance with Distributed Proportional Control / Performance Transparency Protocol.
CARD	Congestion Avoidance using RTT Delay.
CTCP	Compound TCP.
DBECD	Delay-Based Early Congestion Detection.
DKWSA	Dynamic Kiefer-Wolfowitz Stochastic Approximation.
DUPACK	Duplicated ACK.
E2E	End-to-End.
EC	Efficiency Controller.
ECN	Explicit Congestion Notification.
ERICA	Explicit Rate Indication for Congestion Avoidance.
ERN	Explicit Rate Notification.
FC	Fairness Controller.
FIFO	First-In First-Out.
FTP	File Transfer Protocol.
HSTCP	High Speed TCP.
HTTP	HyperText Transfer Protocol.
IETF	Internet Engineering Task Force.

IGI	Initial Gap Increasing.
IP	Internet Protocol.
ISP	Internet Service Provider.
LDHP	Long-Distance High Performance.
LFN	Long Fat Network.
MIMD	Multiplicative Increase Multiplicative Decrease.
MPLS	Multiprotocol Label Switching.
MSS	Maximum Segment Size.
NACK	Negative ACK.
OS	Operating System.
PTR	Packet Transmission Rate.
QoS	Quality of Service.
RBPP	Receiver based Packet Pair.
RBUDP	Reliable Blast UDP.
RED	Random Early Detection.
RSVP-TE	Resources Reservation Protocol - Traffic Engineering.
RTO	Retransmission TimeOut.
RTP	Real-Time Transport Protocol.
RTT	Round Trip Time.
RUDP	Reliable UDP.
RUNAT	Reliable UDP-based Network Adaptive Transport.
SABUL	Simple Available Bandwidth Utilization Library.
SRED	Stabilized RED.
SSH	Secure SHell.
STCP	Scalable TCP.
TCP	Transmission Control Protocol.
TTL	Time To Live.
UDP	User Datagram Protocol.
UDT	UDP-based Data Transfer.
VBE	Variable Bandwidth Environment.
VCP	Variable-structure congestion Control Protocol.

XCP eXplicit Control Protocol.