Joint network and system performance for cloud computing

Sonia Belhareth

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION

ET DE LA COMMUNICATION

THÈSE

Pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention : Informatique

Présentée et soutenue par

Sonia BELHARETH

Performances Réseaux et Système pour le Cloud Computing

Directeur de thèse : **Guillaume URVOY-KELLER**

soutenue le 18 Décembre 2014

Jury :

Directeur de thèse	M. Guillaume URVOY-KELLER	Professeur, UNS, France
Co-directeur	M. Denis COLLANGE	Orange Labs, France
	M. Dino LOPEZ-PACHECO	UNS,France
Rapporteur et Président	M. André-Luc BEYLOT	ENSEEIHT, France
Rapporteur	M. Martin HEUSSE	ENSIMAG, France
Examinateur	M. Matti SIEKKINEN	Aalto University, Finland

Abstract

Cloud computing enables a flexible access to computation and storage services. This requires, for the cloud service provider, mastering network and system issues. During this PhD thesis, we focused on the performance of TCP Cubic, which is the default version of TCP in Linux and thus widely used in today's data centers.

Cloud environments feature low bandwidth-delay products (BPD) in the case of intra data center communications and high BDP in the case of inter data center communications. We have developed analytical models to study the performance of a TCP Cubic connection in isolation or a set of competing TCP Cubic connections. Our models turn out to be precise in the low BDP case but fail at capturing the synchronization of losses that NS-2 simulations reveal in the high BDP case.

We have complemented our simulations studies with tests in real environments: (i) an experimental network at I3S and (ii) a cloud solution available internally at Orange: Cube.

Studies performed in Cube have highlighted the high correlation that might exist between network and system performance and the complexity to analyze the performance of applications in a cloud context.

Studies in the controlled environment of I3S have confirmed the existence of synchronization and enabled us to identify its condition of appearance. We further investigated two types of solution to combat synchronization: client level solutions that entail modifications of TCP and network level solutions based on queue management solutions, in particular PIE and Codel.

Résumé

Le cloud computing permet d'offrir un accès à la demande à des ressources de calcul et de stockage. Le succès du cloud computing nécessite la matrise d'aspects système et réseau. Dans cette thèse, nous nous sommes intéressés aux performances du protocole TCP Cubic, qui est la version par défaut de TCP sous Linux et donc présent dans de nombreux serveurs opérationnels dans les data centers actuels.

Afin de comprendre les performances d'un environnement cloud, qui offre un faible produit bande passante-délai pour le cas intra-data center, et un fort produit dans le cas inter-data center, nous avons développé des modèles analytiques pour les cas d'une ou plusieurs connexions TCP Cubic. Nos modèles se sont révélés précis dans le cas intra-datacenter, mais ne capturaient pas la synchronisation des pertes indiquée par les simulations NS-2 dans le cas inter-datacenter.

Nous avons complété les simulations par des tests en environnements réels avec (i) un réseau expérimental à l'I3S ; et (ii) une solution cloud interne à Orange : Cube.

Les études dans Cube nous ont démontré la forte corrélation qui pouvait exister entre performances réseau et système, et la complexité d'analyser les performances des applications dans des contextes cloud.

Les études dans l'environnement I3S ont confirmé la forte synchronisation qui peut exister entre connexions TCP Cubic et nous ont permis de définir les conditions d'apparition de cette synchronisation. Nous avons étudié deux types de solution pour lutter contre la synchronisation: des solutions niveau client, avec des modifications de TCP Cubic, et des solutions réseau avec l'utilisation de politiques de gestion de tampon, notamment PIE et Codel.

Acknowledgments

I express my gratitude to Prof. Guillaume Urvoy-Keller, for the time he spent with me, for particularly enriching scientific exchanges, to the great cultural discussions, for his support and sympathy. I also thank Dr. Dino Lopez-Pacheco with whom I took a real pleasure to work with.

My thanks to Dr. Denis Collange for our daily trading during these three years, his involvement, his wise and decisive in times of doubt advices.

A special thank to Dr. Lucile Sassatelli for her advices, help and excellent knowledge on Mean-field theory.

I want to thank the members of my jury for agreeing to evaluate this thesis, including M. André-Luc Beylot, M. Martin Heusse, and M. Matti Siekkinen. Their skills and generosity have contributed to improving the manuscript.

I do not forget to thank my friends: Mariem, Hajer, Ameni, Olfa, Youssef, Neetya, Imen, Wafa, Bryan, with whom I spent unforgettable moments.

A big thanks, I address to my dear family who trusted me, and always encouraged me, and this has been my unwavering support during these three years.

Finally, I extend heartfelt thanks to my Love Zouhir, for his continued and unfailing love, support and understanding that makes the completion of this thesis possible.

Contents

List of Figures

List of Tables

Acronyms

ACK	**A**cknowledgment
ADSL	**A**symmetric **D**igital **S**ubscriber **L**ine
AIMD	**A**dditive **I**ncrease **M**ultiplicative **D**ecrease
API	**A**pplication **P**rogramming **I**nterface
AQM	**A**ctive **Q**ueue **M**anagement
ARED	**A**daptive **R**andom **E**arly **D**iscard
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
BDP	**B**andwidth **D**elay **P**roduct
CDF	**C**umulative **D**istribution **F**unction
CPU	**C**entral **P**rocessing **U**nit
CoDel	**C**ontrolled **D**elay
CoV	**C**oefficient of **V**ariation
DC	**D**ata **C**enter
EBCDIC	**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode
EC2	**E**lastic **C**ompute **C**loud
ECN	**E**xplicit **C**ongestion **N**otification
FC	**F**ast **C**onvergence
FIFO	**F**irst **I**n **F**irst **O**ut
FT	**F**rance **T**elecom
GIN	**G**roup **I**ntranet **N**etwork
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
I3S	**I**nformatique, **S**ignaux et **S**ystémes
IaaS	**I**nfrastructure as a **S**ervice
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IP	**I**nternet **P**rotocol
I/O	**I**nput/**O**uput

MSS	**M**aximum **S**egment **S**ize
NS-2	**N**etwork **S**imulator version **2**
ODE	**O**rdinary **D**ifferential **E**quation
OLNC	**O**range **L**abs **N**etworks and **C**arriers
OS	**O**perating **S**ystem
PDV	**P**acket **D**elay **V**ariation
PIE	**P**roportional **I**ntegral controller **E**nhanced
QUIC	**Q**uick **U**DP **I**nternet **C**onnections
RAM	**R**andom **A**ccsess **M**emory
RDP	**R**emote **D**esktop **P**rotocol
RED	**R**andom **E**arly **D**iscard
RTT	**R**ound-**T**rip **T**ime
SSH	**S**ecure **S**hell
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
US	**U**nited **S**tates
VM	**V**irtual **M**achine

I dedicate this THESIS to my family and my Love,
for their encouragement, their support,
their patience and their love.

Introduction

Context and Motivation

Transmission Control Protocol (TCP) [CK05] is the main protocol for exchange of data in Internet Protocol (IP) network. TCP is involved especially in Web page download, HTTP streaming and peer to peer traffic, which together account for a large volume of transfers on the Internet. According to [JD04], more than 90% of the Internet traffic is handled by the TCP protocol. Therefore, if one wants to assess the Quality of Service (QoS) perceived by the cloud services customers, it is mandatory to study the performance, stability and robustness of TCP in Data Center networks.

To meet the changing requirements of Internet networks, various strategies for congestion control in TCP have been designed. Among them we can mention, Bic-TCP [XHR04a], TCP Cubic [HRX08] and Compound TCP [TSZS06]. Bic-TCP and TCP Cubic are designed specifically for high bandwidth delay products (BDP) links.

TCP Cubic is implemented and used by default in the Linux kernel since version 2.6.19, and it is the most widely used version of TCP nowadays [YLX+11]. It is characterized by a cubic window growth function. The purpose of TCP Cubic is to achieve fair bandwidth allocation among flows with different RTTs using a window growth rate independent of the RTT. Instead, it uses a function of the time elapsed since the last loss event. When it does not operate in a high-bandwidth product environments, TCP Cubic features a TCP mode that mimics legacy TCP (TCP New Reno [PFTK00]) but is not identical to it.

The kind of networks for which TCP Cubic (and other high-speed TCP flavors) has been designed calls for analytical models to predict performance in general cases and study the impact of various parameters. Indeed, using experimental testbeds does not usually provide enough flexibility to explore a wide range of parameter values.

Massive data transfers are common in typical cloud scenarios, either within the data center itself or between the data center and the customer premise. In such a scenario, the transport layer, namely TCP, is put under pressure and might suffer performance

problem, e.g., the TCP incast problem which is observed when a large number of storage devices simultaneously send data chunks to a single machine leading to congestion at the switch servicing the machine [VPS+09a].

Cloud environments are characterized by plenty of bandwidth. Modern versions of TCP such as TCP Cubic, are designed to work efficiently in such situations as they are able to probe for available bandwidth in a non linear fashion, unlike TCP New Reno, which inflates its windows by one MSS per RTT in stationary regime. However, there is a price to pay for being more aggressive: the fairness offered by TCP Cubic and other high speed versions of TCP is not as high as legacy TCP versions [LLS07]. Several studies also pointed out the appearance of synchronization among competing TCP Cubic flows [HR08]. The latter means that TCP Cubic flows, when competing for a bottleneck, tend to loose packets at the same time instant and the resulting aggregated throughput time series exhibit a clear Cubic behavior as if a single flow was active.

Some studies have linked the synchronization problem with the buffer sizing issue. Buffer sizing is a key aspect for both router design and network performance. After 20 years of intense activity in the study of "buffer sizing", no proposal seems completely satisfactory. The empirical rule of the Bandwidth Delay Product (BDP) is still widely used as it optimizes the load of the output link of a router. However, it does not guarantee neither the loss rate nor the delay experienced by packets. Under certain conditions of traffic, a buffer size equal to the BDP can lead to performance problems essentially a very large time (i.e., the phenomenon of "bufferbloat"). Therefore, the scientific community has focused on the study of active mechanisms in the buffer in order to answer the original question in the context of buffer sizing. Among these, there are proposals of adaptive buffer and more frequently today Active Queue Management (AQM) mechanisms in order to ensure a quality of service to individual flows. Random early detection RED [FJ93a] is one of the early AQM disciplines. Given that it needs careful tuning of its parameters for various network conditions, most network operators do not turn RED on [Ada13]. The adaptive RED or active RED (ARED) algorithm [FGS01] infers whether to make RED more or less aggressive based on the observation of the average queue length. Recently, two new AQMs Codel [NJ12b] [NJ12a] and PIE [RPC+13] have been proposed to control the latency in order to address bufferbloat problem.

In general, our work aim to study the performance of TCP Cubic, widely used in today's data centers, in cloud environment.

Outline and Contributions

In Chapter 1, we review the basic concepts and motivations of works related to this thesis. First, we present research efforts to analyze performance and failures in a data center networks. Several research works have been done to: (i) explore the nature of traffic in data centers; (ii) identify and resolve performance problems; and (iii) propose comparator of the performance and cost of cloud providers. Then, we review in details TCP, the dominant transport protocol in cloud networks, and studies related to congestion control, buffer sizing and active queue management (AQM) mechanisms. Furthermore, we review studies that established a performance evaluation of TCP Cubic, through simulations and experiments. Finally, we present analytical models for TCP.

In Chapter 2, we first present a detailed overview of TCP Cubic algorithm, then we describe the analytical model that we have developed to study the performance of an isolated long-lived TCP Cubic flow. Through this model, we reveal the differences between the published TCP Cubic specifications and its implementation in NS-2. The model was validated by comparison with NS-2 simulations.

In Chapter 3, we aim at developing an analytical model for TCP Cubic to analyze its performance in typical cloud scenarios where a large number of long-lived TCP connections, e.g., HTTP streaming or back-up flows, share a bottleneck link. Specifically, we consider three scenarios: (i) an intra data-center (DC) scenario with a lot of ongoing traffic between physical servers (intra-DC scenario); (ii) an inter data-center scenario where highly provisioned links are used to synchronize or back up data (inter DC scenario); and (iii) a content distribution scenario where a large number of high speed clients, e.g., FTTH clients, simultaneously download content from the data center (FTTH scenario). Our contributions are twofold:

- Based on a mean-field approximation, we derive a fluid model for TCP Cubic, that allows to predict performance in terms of several metrics. We additionally prove a scaling property of the fluid model, that allows to use it in a variety of networking scenarios without prohibitive computational cost. We carefully validate this analytical model against NS-2 simulations for our cloud scenarios.

- We provide an extensive comparison of TCP Cubic and New Reno for cloud scenarios, assessing their efficiency/fairness trade-off as well as the impact of the buffer size on their performance. In particular, we demonstrate that TCP Cubic outperforms TCP New Reno, even for cases where the BDP is low, which are often encountered in cloud scenarios. This is interesting as TCP Cubic is mostly known

for its behavior on high BDP paths ; not on low BDP paths.

In Chapter 4, we study the performance of TCP in real networks using two testbeds. The first testbed, called "Cube", is an experimental network used by Orange Lab in order to test new services provided by the France Telecom (FT) company. The second testbed consists of a few Linux machines in I3S Lab that can be booted either under a native CentOS operating system or under VMware or Xen. This chapter comprises two separate parts. In the first part, we present results obtained with benchmarking tools for disk read and write operations. We use both the dd [Ubua] and hdparm [Ubub] tools.

In the second part of Chapter 4, we report on the network measurements performed from Cube machines. We generate traffic from or to one Cube machine to one host located in Orange Labs in Sophia. Next, we report the measurements in the Sophia to Cube and then on the Cube to Sophia direction. When increasing the number of parallel flows from Cube to Sophia, we encountered some factors limiting the performance of these transfers: (i) the disk access; (ii) the virtualization layer; and (iii) a shaper. The engineering issue in this chapter was an opportunity for us as it allowed us to recognize the interaction between the system and network issues in a typical cloud solution.

In Chapter 5, we investigate the issue of synchronization among TCP Cubic sources in detail. We study the extent and the root causes of synchronization using an experimental approach with a testbed hosted in I3S Lab combined with simulations. The former enables to experiment with actual protocol implementation in a controlled environment, while the latter permits to explore a wider set of network scenarios.

Our contribution to the study of the synchronization of TCP Cubic are as follows:

- We experimentally establish the relation between the existence and extent of synchronization with key parameters like RTT and buffer size. We demonstrate the resilience of synchronization to background traffic, and how the Fast Convergence option, which aims at making TCP Cubic more fair to fresh connections, catalyzes synchronization. For this point and the subsequent ones, we use New Reno as a reference point.

- We demonstrate that several factors contribute to the appearance of synchronization in TCP Cubic: (i) the rate of growth of the congestion window when a TCP Cubic source reaches the capacity of the network and its relation to the RTT of the connection; (ii) the way the congestion algorithm of TCP Cubic tracks the ideal cubic curve in the kernel; and (iii) the competition among the TCP Cubic sources and the aggressiveness of the sources that did not experience losses during the last loss episode.

- We propose and evaluate two approaches to reduce the level of synchronization and hence the loss rate of TCP Cubic transfers. Perhaps more importantly, we provide hints that synchronization is the price to pay to have a high-speed TCP version that needs to explore the available bandwidth in the network in a super-linear manner. It is probable that we can alleviate synchronization, as our modifications of TCP Cubic do, but eliminating it out completely will be a complex task.

In Chapter 6, we evaluate the potential impact of the queue management algorithms (i.e., CoDel [NJ12b] [NJ12a], PIE [RPC+13], and ARED [FGS01]) on synchronization. Also, we explore how the two approaches that we proposed in Chapter 5 can be combined with advanced queuing mechanisms to further reduce synchronization. We show that through the use of active queue management (AQM) mechanisms we may have smaller delays and reduce synchronization between flows, but there is a price to pay: a higher packet loss and goodput degradation. Compared to our approaches, the use of AQMs for reducing synchronization was more efficient.

Finally, the contents of this thesis are summarized and perspectives are provided in Chapter 7.

Chapter 1

State of the art

1.1 Introduction

In this chapter, we present the research efforts related to the different parts of the thesis. We revisit the most important related works and we highlight problems faced when we revisited TCP performance in a cloud environment. We also provide a high level overview of the challenges we address in this work.

1.2 Performance of data centers

In the context of this thesis, we aim at investigating some of the performance problems raised by cloud computing architectures. Those problems are related to several elements:

- Computational/storage facility that generally consists of a bunch of high end servers that are heavily virtualized.
- Interconnection network within the datacenter that connect the physical servers to the Internet gateways (of the datacenter) with whom clients interfer.
- The paths between the gateway and the end users that is not under the control of the entity which runs the cloud computing service, but is the major source of delay and can constitute the bottleneck of the overall system.

Through a set of techniques and methods researchers can extract useful information concerning the path taken by a transfer and the components involved, then by applying data analysis they can overcome limitations of each variable (i.e., device, link), and draw conclusions that lead to actions.

1.2.1 Data Analysis

[GJN11] presents the first large-scale analysis of network failure events in several of Microsoft's data centers, that was conducted by a team from the University of Toronto

and Microsoft Research. In this study, the authors focus on characterizing failures of network links and devices, estimating their impact, and analyzing the effectiveness of network redundancy in masking failures. They developed a methodology that correlates network traffic logs with logs of actionable events, to filter a large volume of non-impacting failures due to spurious notifications and errors in logging software. The authors want to leverage lessons learned from this study to guide the design of future data center networks.

Nowadays, data centers house a large number of inter-connected servers. It is thus necessary to explore how to better design and manage the data center networks. Existing proposals collect the flow traces by deploying additional modules on either switches or servers in small scale data center networks. In [KSG+09], the authors investigate the nature of data center network traffic on a single MapReduce data center and investigate whether traffic matrices can be inferred from link counters by tomographic methods. They capture several characteristics of traffic flows within data center networks (i.e., the real traffic characteristics): which server is talking to which other servers, when and for what reasons, the flow durations, the inter-arrival times.

1.2.2 Scalable Network-Application Profiler (SNAP)

In [YGM+11], the authors present SNAP, a scalable network-application profiler that guides developers to identify and resolve performance problems. They aim to reduce the demand for developer time by automatically identifying performance issues and narrowing them down to specific times and places (e.g., send buffer, delayed ACK, or network congestion).

The SNAP's features are :

- It has full knowledge of the network topology, and the mapping of applications to servers. This allows it to identify applications with frequent problems, as well as congested resources that affect multiple applications.
- It can instrument the network stack to observe the evolution of TCP connections directly, rather than trying to infer TCP behavior from packet traces.
- It can collect finer-grain information, compared to conventional SNMP statistics, without resorting to packet monitoring.

SNAP collects TCP statistics and socket-level logs in real time, classifies and correlates the data to pinpoint performance problems. The profiler quickly identifies the right location (end host, link, or switch), the right layer (application, network stack, or network), at the right time.

To validate the design of SNAP, the authors take two approaches: they inject a few known problems in the production data center and check if SNAP correctly catches these problems (SNAP correctly pinpointed all the labeled problems), or they evaluate the accuracy of identifying delayed ACK in SNAP by comparing its results with the packet trace.

1.2.3 Comparing Public Cloud Providers (CloudCmp)

In [LYKZ10], the authors present CloudCmp, a comparator of the performance and cost of cloud providers. They apply CloudCmp to the four most known cloud providers today, and they show that it can guide customers in selecting the best-performing provider for their applications.

CloudCmp measures the elastic computing, persistent storage, and networking services offered by a cloud along metrics that directly reflect their impact on the performance of customer applications.

The authors use three indicators to compare the performance of the compute clusters: benchmark finishing time, cost per benchmark, and scaling latency. These quantities reflect how quickly an instance can run, how profitable it is, and how quickly it can scale.

CloudCmp compares the performance and cost of cloud providers along dimensions that matter to customers. The authors consider three types of metrics :

- Computation metrics : The authors modified a set of Java-based benchmark tasks, a standard benchmark suite for Java virtual machines.
- Storage metrics: The authors wrote their Java-based client based on the reference implementations from the providers.
- Network metrics: standard tools (Iperf and ping) .

From the comparison results, the authors find that the performance and price of the four providers vary significantly with no one provider standing out.

1.2.4 The Incast Collapse problem

With the incast problem application throughput decreases when multiple senders communicate with a single receiver in high bandwidth, low delay networks using TCP. In [VPS+09b], the authors present a solution to eliminate TCP incast collapse in datacenter environment. They propose the use of high-resolution timers to enable microsecond-granularity TCP timeouts.

The authors present and evaluate a set of system extensions to enable microsecond-granularity retransmission timeouts (RTO). They proceeded by modifying the Linux

TCP implementation to use high-resolution kernel timers. They show that for both simulation and real cluster, and for all configurations, goodput drops with increasing RTOmin above 1ms. Through a wide-area evaluation, the authors showed that these modifications remain safe for use in the wide-area, providing a general and effective improvement for TCP-based cluster communication.

1.2.5 Virtualization in cloud environments

Virtualization is the new trend of modern private and public cloud solutions. Several virtualization solutions exist on the market and rely on different principles: (1) para-virtualization; (2) full virtualization; and (3) container-based solutions. A key point for those solutions in a datacenter context, is the technique used to share the physical network interfaces among the virtual machines (VMs) [TIIN10].

Whatever the solution is, one ends up in a situation where a lot of VMs share the physical resources of the physical host. Virtualization is being used by an increasing number of organizations to: (i) reduce power consumption and air conditioning needs; (ii) reduce space savings resulting from decreasing the number of servers as one physical server can host many virtual machines; and (iii) create a more efficient system administration through a centralized and flexible management.

Virtualization also provides high availability for critical applications, streamlines application deployment and migrations. By simplifying operations, it allows IT organizations to respond faster to quick changes in the business demands. Two examples are Amazon EC2 [1] and Windows Azure [2].

Several studies have investigated the impact of virtualization on performance in cloud environments. In [APU12], the authors created a testbed (hosted in the I3S lab France) and used it to evaluate three operating systems (OS) CentOS Vanilla, VMware ESX and Xen in terms of User Datagram Protocol (UDP) performance with a brief glance at TCP. They considered the Packet Delay Variation (PDV) as a metric to assess the performance of these 3 OSs. Under UDP, the authors noticed better performance of native CentOS than the virtualized machines, while the TCP performance showed that the virtualized machines behave better than the native OS in terms of fairness between the virtual machines.

In [WN10], the authors present a measurement study to characterize the impact of virtualization on the networking performance of the Amazon Elastic Cloud Computing (EC2) data center. They measure a set of properties in their experiments: the processor sharing, packet delay, throughput and packet loss among Amazon EC2 virtual machines.

[1] http://aws.amazon.com/ec2/
[2] http://www.windowsazure.com/en-us/

They consider mainly Amazon EC2 small instances and high CPU medium instances. Small instances provide a small amount of CPU resources, while medium instances offer a balanced set of CPU resources, memory, and network [3]. The authors set up 2 types of experiments: (i) a spatial experiment to evaluate how the network performance varies on instances at different network locations; and (ii) a temporal experiment to evaluate how the network performance varies on a given instance over a long time period.

Experiments show an unstable network characteristics that are caused by virtualization and processor sharing on server hosts. First, Amazon EC2 small instance virtual machines typically receive only a 40% to 50% share of the processor. Also, processor sharing can cause very unstable throughput among Amazon EC2 small instances. Even though the data center network is not heavily congested, there are abnormally large packet delay variations among Amazon EC2 instances. And, the delay variations can be a hundred times larger than the propagation delay between two end hosts.

1.3 TCP and congestion control

1.3.1 A Protocol for Packet Network Intercommunication

The proposal of TCP laid the basis for the development of Internet. The TCP congestion control mechanism enables the sender to adjust the transmission rate and the congestion window size according to the network conditions. This protocol shows a great improvement in the utilization of the network.

In [CK05], the authors provide one of many important studies on the Internet development. In May 1974, when the paper was published, there existed many kinds of networks, that implemented packet switching in different ways. Hence the necessity of a common protocol to share resources among these networks (inter-networking protocol).

Cerf and Kahn [CK05] are sometimes referred to as the "fathers of the Internet" for implementing the common protocol TCP/IP. They present a protocol design and philosophy that supports the sharing of resources existing in different packet switching networks.

Some fundamental tasks of the TCP protocol were described in this paper:

- *GATEWAY*, a network node that acts as an interface between networks.
- Mutliplexing and demultiplexing of segments among processes.
- Sequencing used for the reconstruction of messages at the TCP receiver.
- Timeout and positive acknowledgment mechanism.

[3] https://aws.amazon.com/ec2/instance-types/

- Flow control mechanism used to manage the rate of data transmission between two nodes, and to prevent a fast sender to overwhelm a slow receiver.

1.3.2 Congestion problem

The huge growth of computer networks has produced major congestion problems. The authors in [Jac88] speculate that much of the cause lies in transport protocol implementations (not in the protocols themselves). In October 1986, the Internet had the first of what became a series of "congestion collapses" that were caused by congestion control in the implementation of TCP. In control system theory, this is expected since the system in that time reaches capacity exponentially but does not reduce the input by a comparable rate.

One key to prevent congestion collapse is to ensure that senders do not over-utilize links at the beginning of the transmission. Thus, to solve this problem, the authors looked for pieces of a TCP implementation that violate the conservation principle. The packet conservation principle states that a packet should only be injected into the network after one has been removed . If the network accomplishes this principle, it should be robust in face of congestion.

They reported three ways to violate the principle : either (1) the connection does not get to equilibrium; or (2) a sender injects a new packet before and old packet left; or (3) the equilibrium can not be reached because of resource limits along the path. To achieve the required equilibrium, the authors developed an algorithm that increases the packets that a sender sends with several rules. They add a congestion window (*cwnd*) for each connection at the sender, when starting or restarting after a loss, set *cwnd* to one packet. On each acknowledgment the *cwnd* is increased by one packet, and when sending, send the minimum of the receiver's advertised window and *cwnd*.

To resolve the second problem of "Conservation at equilibrium", the authors set a RTT estimator.

Finally, to solve the resource limits problem, the authors developed a congestion avoidance algorithm that decreases the *cwnd* to half its current value (this is the multiplicative decrease), and for each ACK for new data, increases *cwnd* by 1 (this is the additive increase).

The principle of congestion avoidance (Additive Increase Multiplicative Decrease AIMD) is illustrated in Figure 1.1. Paths $x_0x_1, \ldots, x_2nx_{2n+1}$ represent the additive increase part where both flows have the same increase rate of their congestion windows. While paths $x_1x_2, \ldots, x_{2n+1}x_{2n+2}$ represent the multiplicative decrease for which a flow with the larger congestion window decreases more than a flow with the smaller congestion window.

In the flow-control world if the offered load in an uncontrolled distributed sharing system (e.g., road traffic) exceeds the total system capacity, the effective load will go to zero (collapses) as load increases, see Figure 1.2.

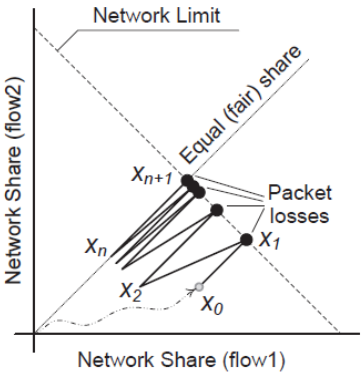

FIGURE 1.1: Congestion avoidance (AIMD) [ATRK10]

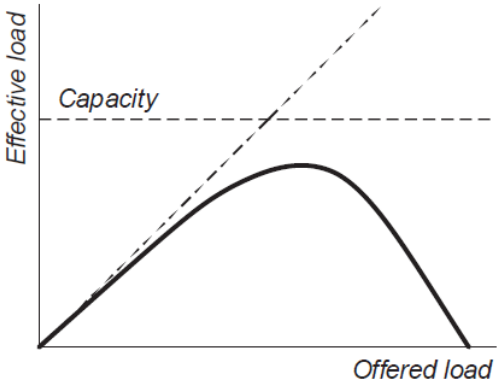

FIGURE 1.2: Effective TCP load versus offered load from TCP senders [ATRK10]

1.3.3 Congestion control for TCP

In the last few years, through the development and evolution of the Internet, the focus of research has changed from the fundamental problem of removing the congestion collapse phenomenon to problems of the efficient use of network resources in variety of environments.

In [ATRK10], the authors presented a study of different approaches to TCP congestion control that do not rely on any explicit notification from the network. They focus on a variety of problems that TCP tried to optimize.

They devoted the first part of their study to classify and to discuss proposals that build a foundation for host-to-host congestion control principles. The first proposal is Tahoe [Jac88], the first version of TCP with congestion control, proposed by Jacobson in 1988. Tahoe introduces the basic technique of probing progressively network resources and relying on packet loss to detect that the network limit has been reached. Although this technique resolves the problem of congestion collapse, it creates a great deal of inefficient use of the network by straining the network with high-amplitude periodic phases. This behavior induces periodic changes in sending rate, round-trip time, and network buffer utilization, leading to variability in packet losses. To resolve the efficiency problem, the authors propose algorithms that refine the core congestion control principle by making more optimistic assumptions about the network and using prediction parameters of the network congestion state (e.g., three duplicate acknowledgment (ACK), timeout).

Two proposals, BIC TCP [XHR04b] and TCP Cubic [HRX08], use packet loss to establish an approximated network resource limit, which is used as a secondary criterion to estimate the current network state. TCP Cubic uses time elapsed since last congestion to update its window size via a cubic function. Another proposal is Compound TCP [TSZS06] that relies on the estimation of queueing delay as a measure of congestion. If the queueing delay is low, Compound TCP supposes that there is no congestion and increases the rate. Compound TCP is deployed in the Windows-world, and the Linux-based OSs use TCP Cubic.

1.4 Other congestion control strategies

1.4.1 Buffer sizing

In recent years, several studies addressed the problem of how to adequately dimension router buffers for large bandwidth delay product (BDP) networks. So, to compute the "right" amount of buffering, several rules derived from these studies which are conflicting with each other.

Router buffers are sized today based on a rule-of-thumb commonly attributed to a 1994 paper by Villamizar and Song [VS94]. The authors used experimental measurements of at most eight TCP flows on a 40Mb/s link. Their results show that a router needs an amount of buffering equal to the average RTT per flow that passes through the router, multiplied by the capacity of the router's network interfaces C, see Equation 1.1 .

$$B = C \times \overline{RTT} \tag{1.1}$$

But, it is not easy to build routers with larger capacity. That is why other studies have derived other rules for determining an adequate buffer size B.

In [AKM04], the authors argue that with the rule-of-thumb queuing delays can be long, have high variance, may destabilize the congestion control algorithms, that make it inappropriate for backbone routers. So, they have proposed a new rule 1.2 that can be applied for both N long-lived and short-lived TCP flows. To validate this rule, the authors used two validation methods: simulation using NS-2, and a network of real Internet routers.

$$B = C * \overline{RTT}/\sqrt{N} \tag{1.2}$$

In [WM05], the authors recommend using buffer sizes equal to Equation 1.3

$$B = 0.63 * C * \overline{RTT}/\sqrt{N} \tag{1.3}$$

By means of NS-2 simulations, the authors in [HR09] and [CB07] assert that an adequately dimensioning router buffers for links should be in the order of 10%BDP and B=20%BDP, respectively.

Other papers [Rai05] [EGG+05] claim the use of buffers of the order of a few dozen of packets.

According to what we have read above there is no consensus yet on what is a "good" dimensioning rule that allows to always achieve high utilization.

1.4.2 Active Queue Management mechanisms

Active Queue Management (AQM) mechanism is a congestion control mechanism at a router for controlling the number of packets in the router's buffer by actively discarding some arriving packets. It can shorten the average delays in the router's buffer and can also achieve higher throughput.

Random Early Detection (RED) [FJ93b] is one of the AQM disciplines. It controls the queue length which would affect delay implicitly. RED requires careful tuning of their parameters in order to provide good performance.

Several modern and self-tuning AQM disciplines have been proposed, and can be run with their default parameters in most circumstances. The adaptive RED or active RED (ARED) [FGS01] algorithm infers whether to make RED more or less aggressive based on the observation of the average queue length. It changes the drop probability according to how aggressively it senses it has been discarding traffic.

More recently, some researchers designed and implemented new AQMs: (i) Controlled Delay (CoDel) developed by Van Jacobson [NJ12b]; and (ii) Proportional Integral controller Enhanced (PIE) [RPC+13]. These AQMs control the latency to address bufferbloat problem.

There have been some comparisons of these AQMs: in [RPC+13] the authors show that compared to Codel and ARED, PIE can achieve better latency and higher link utilization.

In [NK13], the authors conducted a comparison of Adaptive RED, Codel and PIE through experiments in both wired and wireless networks. They show that ARED achieves better queuing delay for all RTTs, but it loses in terms of goodput for RTTs larger than 100ms.

In [HR09], the authors present an evaluation of the potential impact of the Random Early Detection queue management algorithm on loss synchronization. Their results show that for large buffers, RED strongly reduces the synchronization rate, compared

to DropTail. With medium to small buffers, the loss synchronization is roughly similar with both types of queue management strategies.

More details about active queue mechanisms will be given in chapter 6.

1.4.3 Multipath TCP

Multipath TCP protocol is a replacement for TCP. It allows creating simultaneous multiple sub-flows amongst two end hosts through multiple network interfaces where each sub-flow maintains sending data packets over a path. The coordinated congestion control moves more traffic on the less-congested paths as a load balancing mechanism.

In [WRGH11], the authors describe and evaluate the design of a multipath congestion control algorithm for multihomed servers, data centers and mobile clients. They show that their algorithm works across a wide range of scenarios and that it can be used as a drop-in replacement for TCP.

1.4.4 Transport protocol at the application layer

Google has proposed 2 new protocols to replace HTTP, SPDY [MBT12] and then QUIC (Quick UDP Internet Connections) [qui13]. These protocols require to be deployed both at the sender and the server sides. No other cloud operator than Google has claimed to use these new protocols. Facebook has proposed to launch the working group "httpbis" within the IETF to standardize the next version of HTTP, the HTTP2.0 [MB13]. HTTP2.0 aims to improve the performance of HTTP transfer using more efficiently TCP.

1.4.4.1 SPDY

In [MBT12], the authors describe SPDY (pronounced "SPeeDY") a protocol conceived for the transport of contents with low latency on the World Wide Web. The most frequent browsers (Internet Explorer, Firefox, Chrome, Opera and Safari) already implement SPDY.

SPDY adds a session layer atop SSL that allows multiple concurrent, interleaved streams over a single TCP connection. Amongst the objectives of SPDY we mention:

- Target a 50% reduction in page load time.
- Minimize deployment complexity as it requires no changes to existing networking infrastructure.
- Avoid the need for any changes to content by website authors. The only changes required to support SPDY are in the client user agent and web server applications.

It should be noticed that some papers express doubts about the actual efficiency of SPDY. Microsoft have proposed a new HTTP, "Speed+Mobility" [Pao12].

1.4.4.2 QUIC

QUIC [qui13] is a network protocol that supports a set multiplexed connections over UDP, and was designed to provide security protection equivalent to TLS/SSL, along with reduced connection and transport latency. It combines a carefully selected collection of techniques to reduce the number of round trips needed to surf the Internet.

QUIC employs bandwidth estimation in each direction for congestion avoidance, and then pace packet transmissions evenly to reduce packet loss. It also use packet-level error correction codes to reduce the need to retransmit lost packet data. QUIC is currently only implemented in Chrome.

1.4.4.3 HTTP2.0

HTTP2.0 [MB13] is an optimized expression of the syntax of the Hypertext Transfer Protocol (HTTP). The HTTP2.0 encapsulation enables more efficient use of network resources and reduced perception of latency by allowing header field compression and multiple concurrent messages on the same connection.

1.4.4.4 Aspera FASP

In [Comb], the authors present Aspera FASP a data transport technology built to supply an optimal alternative to traditional technologies of transport based on TCP to transfer files over public and private IP networks. It represents a service proposed by Amazon [4].

Aspera is implemented at the application layer, as an endpoint application protocol, avoiding change of network standard. It is designed to provide an efficient data transport over an IP network independent of network delay and packet loss.

Aspera has several major improvements:

- Bandwidth discovery : Automatically discovers the path MTU size, which may avoid packet fragmentation and improves performance.
- Optimization for small files : Introduce a file streamlining technique which eliminates the performance bottleneck when transferring several small files.
- Improved adaptive transfer rate : Enable users to utilize all available bandwidth along the transfer path, while fairly sharing the network capacity with other traffic.

[4] http://cloud.asperasoft.com/aspera-on-demand/aspera-on-demand-for-amazon-web-services/

- Parallel transfer : Introduces parallel transfer for server clusters or multi-core machines, enabling full leverage of computing power.
- Aggregate bandwidth management : Include the management of the aggregate bandwidth to keep the total transfer rate of all transfers below a pre-configured bandwidth.

1.5 TCP Cubic evaluation : models, simulations and experiments

The focus of this thesis was on TCP Cubic [HRX08], implemented and used by default in Linux kernels since version 2.6.19. This protocol achieves some improvements on BIC TCP [XHR04b]. It performs well in wired networks with large bandwidth-delay product. TCP Cubic simplifies the BIC TCP window control and improves TCP-friendliness.

We made a thorough study about models as well as experimentations that have been elaborated in the literature about TCP Cubic protocol. We report in the next section several important works related to TCP Cubic.

TCP Cubic is a TCP-friendly high-speed variant, in which the window size is a cubic function of time since the last loss event. More details about the window evolution will be given in the next chapter.

Several studies have evaluated the performance of TCP Cubic through simulations and experiments. Analytical models for TCP Cubic with a large number of flows sharing a common bottleneck link are few. We classified these works in 2 categories: models and experimentations.

1.5.1 Evaluation through analytical models and simulations

In [BWL10], the authors propose an analytical model to analyze the performance of a single TCP Cubic connection in wireless networks. This model aims to determine the steady state throughput of TCP Cubic. It considers both congestion loss and random packet loss. Congestion loss occurs when the transmission rate reaches the maximum capacity C of the bottleneck link. However random packet loss is caused by fading, or interference on the wireless link (random poisson process with rate λ).

In order to validate the accuracy of the proposed analytical model, the authors develop a discrete-event simulator. They consider the root mean square (RMS) error as a performance metric. The RMS error reflects the gap between the analytical results and the simulation results. Results show that (1) as the simulation time is increased the RMS error decreases. (2) The random packet loss reduces the normalized average throughput more for end-to-end flow with large bandwidth-delay product.

In [PS11], the authors build an analytical model for TCP Cubic with random packet drops and constant RTT. They investigate two cases: (i) three TCP Cubic connections; and (ii) two TCP connections, one TCP Cubic and the other one TCP New Reno. The authors consider the case of random packet losses, each packet of connection i gets lost in transmission with probability $p_i \geq 0$ independently of each other. (This can corresponds to wireless links). They use the M/GI/1 approximation to evaluate the sojourn time in the queue at the router. All connections sharing the same link are subject to the same loss rate and same propagation delay $\triangle$. Through comparison with NS-2 simulations, the authors show that the model captures the dynamics of TCP Cubic fairly well. The results using the model closely match the results obtained through NS-2 simulations.

The authors in [BAC09] compare the performance of TCP Cubic, Compound TCP, HighSpeed TCP and Reno under a simple loss model, where each packet is dropped with probability p. They model the evolution of the congestion window with a Markov chain, and use efficient numerical algorithms to compute the average window size, the Coefficient of Variation (CoV) of the window and the average throughput. They find that, for smaller bandwidth delay products (i.e., 150 packets), TCP Cubic can have a similar throughput to Reno while for larger values (i.e., between 5000 and 7000 packets) the throughput of all new versions is similar and larger than Reno.

Also in [ARFK10], the authors compare the performance of TCP New Reno, TCP Cubic and Compound using NS-2 simulations. They consider 3 scenarios: (1) a single connection; (2) a single bottleneck with N sources sharing a 10Gbps link; and (3) wireless link. They investigate the goodput, intra and inter-protocol fairness of these two protocols. Results show that TCP Cubic outperforms the other two variants in terms of goodput and intra-protocol fairness in high speed wired networks. In wireless networks all protocols are unable to reach a high goodput in the presence of reverse traffic. Also, the intra-protocol fairness is almost equal to 1 for all of them.

In [LGBP10], the authors proposed a new method to predict TCP throughput's variations. This method is based on the theorem of large-deviations that has been applied to a Markov-chain model of the evolution of TCP Reno's congestion window. They used the loss probabilities imposed by the network conditions for calculating a theoretical prediction of the so-called large-deviations spectrum which contains detailed information on TCP performance. The resulting prediction remains perfectly accurate in complex environments and also on real Internet. This method was applied on other TCP variants: BIC TCP, HighSpeed and TCP Cubic, and the obtained results show that it permits fine performance characterization of these TCP variants.

In [LHHL12], the authors introduce an extended version of TCP Cubic, TCP multiple paths (MPCubic). MPCubic moves traffic away from congested paths to uncongested paths, and fairly share the capacity with standard TCP at common bottleneck. This protocol can achieve stability, throughput improvement, fairness, and load-balancing. Through simulation results, the authors found that the proposed protocol can outperform MPTCP, improve throughput performance, and can improve its traffic away from congested link, while it can preserve fairness with single-path TCP Cubic, regular TCP and MPTCP (with short RTTs). Moreover, MPCubic can quickly recover its data rate after restoration of failed links.

The goal in [CEH+09] is to compare the protocol growth functions, especially in terms of the second or higher-order stochastic behaviors of the protocols that employ these functions. They indicate that protocols having a concave-convex window growth function and using the maximum window size in the last congestion event as an inflection point, have most of the time a concave window growth profile in the steady state. These results were confirmed for both BIC TCP and TCP Cubic protocols that have this property, through NS-2 simulation and experimentation.

1.5.2 Evaluation through experiments

In [JR11], the authors present an experimental evaluation of TCP Cubic in a small buffer regime (i.e., buffers of the order of a few tens of packets). The experiments are carried out using the NetFPGA platform [5].

They focus on the interaction of long-lived flows, over a variety of bandwidth-delay product environments. This work shows the following effects: (1) for small link capacities, there is a distinctive impact on utilization as the number of users, and the round trip times vary; and (2) for large link capacities, small buffers can induce synchronization effects.

The authors conducted experiments with two sets of capacities: 123Mbps and 946Mbps. In each of these cases, they used two sets of buffer sizes: 16, and 128 packets. The number of flows is equal to 10, 100 and 200 flows.

For scenario with 123Mbps, a buffer size of 16 packets, and for a fixed population of users, as the RTT gets larger, the utilization drops. Also, for a fixed RTT, as the number of users increase, there is a visible increase in the utilization. While, with a buffer size of 128 packets, varying the number of users or the RTT did not produce a visible reduction in the utilization. For scenario with 946Mbps and a buffer size equal 128 packets, authors noted the appearance of synchronization between flows, which means that all flows loose packets simultaneously.

[5] http://netfpga.org/main.html

1.6 Analytical models: Mean-field approach

Analytical models are mathematical models which have a closed form solution. The solution of the equations used to describe the changes in a system can be expressed as a mathematical analytical function.

The initial motivation of mean-field models was to analyze the behavior of computer networks. It can be used to analyze systems with large number of objects; where each object is defined by a stochastic model while the global model can be approximated by a fluid limit.

The authors in [BtLB08] have given a results for a model of N interacting objects. They consider that the ordinary differential equation (ODE) is a good approximation of the occupancy measure of the stochastic system. The paper comprises two separate parts: a first part is devoted to the analysis of the mean field limits of a general system of interacting objects that are also interacting with a random environment. In the second part, the authors demonstrate that this approach can be applied to understand the behavior of computer networks with a large number of objects sharing resources.

In [BMM07], the authors present a generic result that allows a reduction of a large Markov chain model of interacting objects to a dynamical system whose state is the occupancy measure (i.e. the distribution of states of all objects), or, more generally, a function of the history of the occupancy measure. The resulting dynamical system is deterministic, but it can be used to study a stochastic system (with considerably smaller dimension) that reflects the state of one or several tagged objects.

There have been several mean-field models of TCP in the literature such as [BMR02]. In this paper, the authors introduced a mean-field model for a large number N of TCP Reno connections. All connections share a bottleneck queue in a router implementing RED (Random Early Discard) active queue management mechanism. The model converges, when the number of connections N tends to infinity, to a deterministic transport equation.

1.7 Conclusion

In this chapter, we presented an overview of the main research works in correlation with our scopes of TCP Cubic performance analysis. We have reported brief history of some TCP protocols. We also present a survey of some congestion control algorithms. We further report relevant works in (i) Mean-field theory; (ii) buffer sizing; and (iii) virtualization. And finally, we briefly review studies on AQMs mechanisms.

In the next Chapter, we focus on TCP Cubic protocol as it is used by default since 2004 in the Linux Kernel, and we summarized its main characteristics. Then we present our analytical model for a single long-lived connection.

Chapter 2

Analyzing a Single TCP Cubic flow

2.1 Introduction

In this chapter, we first present a detailed overview of TCP Cubic algorithm, then we describe the analytical model that we have developed to study the performance of an isolated long-lived TCP Cubic flow. We highlight the differences between the published specifications of TCP Cubic and its implementation in NS-2. The model is validated by comparison with NS-2 simulations.

2.2 Background on TCP Cubic

2.2.1 Window variation in TCP Cubic

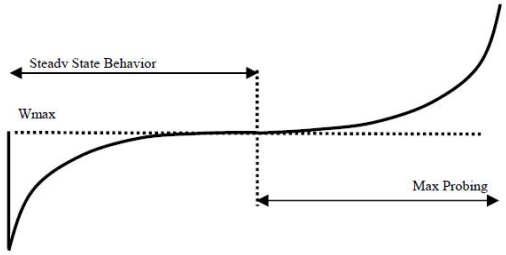

FIGURE 2.1: TCP Cubic window growth function [HRX08]

TCP Cubic [HRX08] is the next version of BIC TCP [XHR04b]. It modifies the linear window growth function of existing TCP Standards with a cubic function in order to improve the scalability of TCP over fast and long distance networks.

Another major difference between TCP Cubic and previous TCP versions is that the congestion window increases is not correlated to the RTT. Indeed the amount of packets by which the congestion window must be increased depends only on the time elapsed since the last congestion event. In contrast, with standard TCP, flows with very short RTTs increase their congestion windows faster than flows with longer RTTs.

We report in Figure 2.1 the growth function of TCP Cubic with the origin at W_{max}, which contains both a concave and a convex part.

At a loss event, TCP Cubic registers W_{max} to be the window size where the last loss event occurred and performed a multiplicative decrease of congestion window by a factor of β (Typically $\beta = 0.2$). After it enters into congestion avoidance from fast recovery, the window starts to increase using the concave profile of the cubic function. The cubic function is set to have its plateau at W_{max} so the concave growth continues until the window size becomes W_{max}.

As the window size gets closer to W_{max} the growth rate slows down and the cubic function turns into a convex profile and the convex window growth begins. This style of window adjustment (concave and then convex) aims at improving protocol and network stability while maintaining high network utilization. This is because the window size remains almost constant, forming a plateau around W_{max} which is likely to be the available bandwidth of the network. Under steady state, most window size samples of TCP Cubic are close to W_{max}, thus enabling high network utilization and protocol stability.

When in congestion avoidance, TCP Cubic features two modes of operations, the so-called TCP and Cubic modes. The TCP mode is to be used in low bandwidth delay products (BDPs), while the Cubic mode is triggered for high BDPs. Each mode corresponds to a specific way of increasing the window size and is determined by the following pair of equations:

$$w_c(t) = C_{cubic}(t - V_{cubic})^3 + w_{max} \tag{2.1}$$

$$w_{tcp}(t) = w_{max}(1 - \beta) + \frac{3\beta}{(2 - \beta)} \frac{t}{R(t)} \tag{2.2}$$

where w_{max} is the congestion window prior to the last loss event[1], $R(t)$ is the estimated RTT of the connection, β and C_{cubic} are constant values usually set to 0.2 and 0.4, respectively, and $V_{cubic} = \sqrt[3]{\frac{\beta w_{max}}{C_{cubic}}}$. The quantity t in Equation 2.1 is the elapsed time since the last loss event.

The congestion window size $cwnd(t)$ is set to $max(w_c(t), w_{tcp}(t))$ upon each ACK reception. TCP Cubic is thus said to operate in Cubic mode (resp. TCP mode) if the maximum is $w_c(t)$ (resp. $w_{tcp(t)}$).

[1]Note that w_{max} is varying over time but is constant between two loss events. This is also the case for V_{cubic}.

The equation of $w_c(t)$ is designed in such a way that when a TCP Cubic connection is operating in the Cubic mode, it converges quickly to w_{max}. Then it plateaus for a while, before increasing again to probe the link to sense whether more bandwidth is available in the path (see Figure 2.1).

Concerning the TCP mode, we can note that $w_{tcp}(t)$ depends both on the RTT of the connection and the time elapsed since the last loss event. Thus, in practice, when the RTT is low, $w_{tcp}(t)$ ensures that the window increase of TCP Cubic is not slower than the one of New Reno.

TCP Cubic possesses an optional mechanism called Fast Convergence (FC). This mechanism is designed to make TCP Cubic fairer. When a new flow joins the network, existing flows in the network must give up their share of bandwidth to allow the new flow to grab some bandwidth.

Upon detection of a loss, w_{max} is set to the last congestion window $cwnd(t)$, before the congestion window be reduced by 20%. In case the last w_{max} was larger than the congestion window when the loss is detected, and if the Fast Convergence mechanism is applied, w_{max} is set to $(1 - \frac{\beta}{2}) * cwnd(t)$ (more details in the Section 2.4.1.1).

2.2.2 TCP Cubic mode of operation

From Equations (2.1) and (2.2), it appears that for fixed network set-up (the physical path between the sender and the receiver) and stationary load conditions, TCP Cubic operates (in stationary regime) in either Cubic or TCP modes but not both.

The Cubic mode depends on the bandwidth delay product of the path, through the value of $w_{\max}$, while the TCP mode depends on the RTT.

We can observe an alternation of modes if $RTT_{\min}$ is below the threshold that triggers TCP Cubic while it is above when the buffer starts filling up and the RTT increases. At 100Mb/s, the latency of the path that ensures that TCP is always in Cubic mode is 39ms, while at 1Gb/s, it is 18ms.

To find those values, one needs to consider the difference $D(t, RTT, w_{max}) = w_c(t) - w_{tcp}(t)$:

$$\begin{aligned} D(t, RTT, w_{max}) &= w_c(t) - w_{tcp}(t) \\ &= C_{cubic}(t - V_{cubic})^3 + \beta w_{max} - \frac{3\beta}{(2-\beta)} \frac{t}{R(t)} \end{aligned}$$

We can see in Figure 2.2 that this difference first increases with t, then decreases and increases again. Finding the first value $t_0(RTT, w_{max}) > 0$ for which the derivative of $D(t, RTT, w_{max})$ with respect to t is zero allows to express the minimum value of the difference. This value $D(t_0, RTT, w_{max})$ is obtained for:

$$t_0(RTT, w_{max}) = \left(\frac{\beta}{C_{cubic}(2-\beta)RTT}\right)^{\frac{1}{2}} + V_{cubic} \ .$$

Then it can be seen that $D(t_0, RTT, w_{max})$ is increasing in RTT and in w_{max}. In the steady state, the value of w_{max} is $BDP + B$, where B denotes the buffer size available for a TCP connection, and the RTT is lower-bounded by the end-to-end path latency denoted by $baseRTT$. Hence, for a given network setting of $baseRTT$, B and BDP, the sign of $F(baseRTT, B, BDP) = D(t_0(baseRTT, B + BDP), baseRTT, B + BDP)$ allows to determine the mode of operation of TCP Cubic.

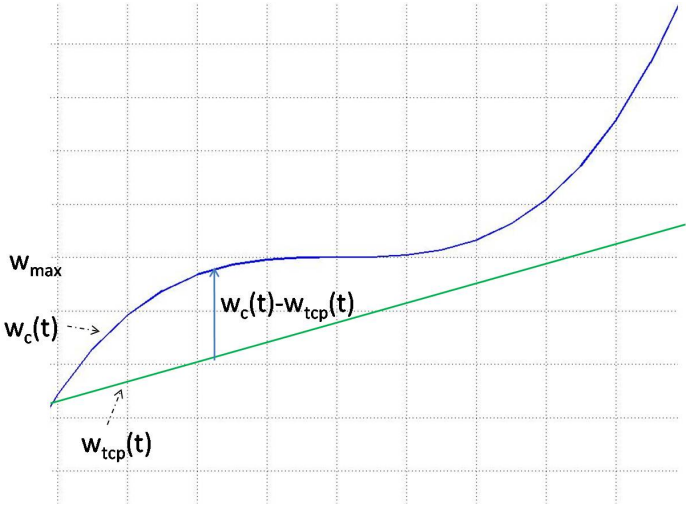

FIGURE 2.2: Congestion window growth of TCP Cubic in Cubic and TCP modes.

2.3 TCP Cubic model

TCP Cubic uses the cubic function (Equation 2.1) of the elapsed time since the last congestion event which usually contains concave and convex parts (the window growth is independent of RTTs). Compared to Standard TCP (New Reno), TCP Cubic aims to improve the scalability of TCP over fast and long distance networks. The state of a TCP Cubic connection is described by the congestion window w_c and w_{max}. We assume that time is slotted and events (e.g. losses) can be detected at those discrete time slots only.

We present a simplified analytical model of TCP Cubic, to analyze in details its behavior for an isolated long-lived flow. Consider the model shown in Figure 2.3. It consists of two nodes (the source and the destination) that are connected through a finite capacity link, with rate C and buffer size B (i.e., FIFO queue). For simplicity, we

Notation	Description
β	The window reduction factor after a loss, equal to 0.2
C_{cubic}	Cubic parameter, equal to 0.4
τ	The propagation delay
C	The capacity of the link in packets per second
m	The packet size (constant)
b	The number of received packets after which the TCP destination sends an acknowledgment (The default value is 2)
B	The buffer capacity in packets
t_x	The starting time of phase x
t_{xy}	The time since the start of the cycle until the completion of stage x and the beginning of phase y
W(t)	The congestion window at time t
S(t)	The sequence number sent at time t
Q(t)	The buffer occupancy for packets sent at time t
R(t)	The round trip time for packets sent at time t
A(t)	The last received acknowledgment at time t
w_{max}	The maximum window size at the last loss event
V_{cubic}	Time period required to increase $w_c(t)$ to w_{max} when there is loss

TABLE 2.1: Notations for the system model

assume that the sender has an unlimited amount of data to send. Table 2.1 lists the parameters of the system.

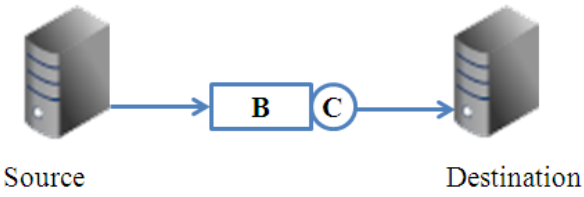

FIGURE 2.3: Model for a single TCP connection

The Round Trip Time RTT and the queue are given by :

$$R(t) = \tau + \frac{Q(t)}{C} \tag{2.3}$$

$$Q(t) = \begin{cases} 0 & ,if \quad W(t) < C\tau \\ W(t) - C\tau & ,if \quad W(t) - C\tau < B \\ B & ,otherwise \end{cases} \tag{2.4}$$

In the literature, few analytical models have been proposed to analyze the performance of TCP Cubic. The authors of [CEH+07], [BWL10], [BAC09] consider a single long-lived flow, while in [PS11] the authors consider 3 TCP Cubic flows.
In [CEH+07], the authors present a stochastic tool, called convex ordering, that provides

an ordering of any convex function of transmission rates of two protocols and valuable insights into high order behaviors of protocols. With their tool, the authors show that a protocol with a growth function composed of a concave function that switches to a convex function around the maximum window size in the previous loss epoch, gives the smallest rate variation under a variety of network conditions. This tool was tested with BIC and TCP Cubic that have this window growth function, with experimentations and simulation results.

In [BWL10], authors propose a Markovian model to determine the steady state throughput of one single long-lived TCP Cubic flow in a wireless environment. The proposed analytical model is validated via simulations. The authors show that random packet loss reduces the normalized average throughput more for end-to-end flow with large bandwidth-delay product.

In [BAC09], Blanc et al. compare the performance of TCP Cubic, Compound TCP, HighSpeed TCP and New Reno under a simple loss model, where each packet can be randomly dropped with probability p. They model the evolution of the congestion window with a Markov chain to compute the average window size, its coefficient of variation (CoV) and the average throughput. They find that, for small bandwidth delay products, TCP Cubic can have a similar throughput to New Reno while for larger values the throughput of all new TCP versions behave similarly and outperform New Reno.

In [PS11], Poojary and Sharma investigate the cases of three TCP Cubic connections as well as the competition between a TCP Cubic and New Reno connection.

As our interest was to study single TCP Cubic connection, we were motivated to develop our own model. We proceeded by detailing some TCP Cubic metrics, then we distinguish the different phases of a TCP Cubic connection from the beginning of the cycle until the detection of a loss. Finally, we model the sending window which is the state variable TCP Cubic and we validate the numerical results of our model with NS-2 simulations.

2.3.1 Dynamical Analysis

2.3.1.1 Different Phases of TCP Cubic

We considered different states corresponding to different phases in the evolution of the window size. TCP Cubic window goes through five states according to the window size $W(t)$ and the buffer occupancy Q(t):

- State A: TCP mode with empty buffer

No	State transition	Description
1	$\langle A \to B \rangle$, $\langle E \to B \rangle$	The queue is empty ($Q(t) = 0$), $W_c(t) > W_{tcp}(t)$
2	$\langle A \to C \rangle$, $\langle D \to C \rangle$, $\langle E \to C \rangle$	The queue is non-empty ($Q(t) > 0$), $W_c(t) < W_{tcp}(t)$
3	$\langle B \to A \rangle$, $\langle E \to A \rangle$	The queue is empty ($Q(t) = 0$), $W_c(t) < W_{tcp}(t)$
4	$\langle B \to D \rangle$, $\langle C \to D \rangle$, $\langle E \to D \rangle$	The queue is non-empty ($Q(t) > 0$), $W_c(t) > W_{tcp}(t)$
5	$\langle C \to E \rangle$, $\langle D \to E \rangle$	A loss occurs

TABLE 2.2: State transitions and corresponding events

- State B: Cubic mode with empty buffer
- State C: TCP mode with a non empty buffer
- State D: Cubic mode with a non empty buffer
- State E: Loss recovery

We report in Figure 2.4, the transitions that can occur between them (i.e., twelve possible transitions).

For each state X, we compute several metrics: (i) the expression of the congestion window; (ii) the number of packets sent during this phase n_x; (iii) the sojourn time since the beginning of the cycle until completion of this state t_x; and (iv) the different transitions to other states.

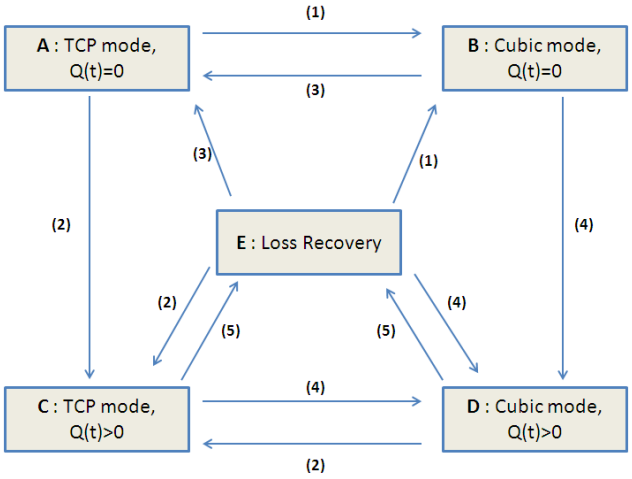

FIGURE 2.4: Transition diagram

Table 2.2 lists the permitted state transitions and the description of the events that trigger the state change. In each of these states the evolution of the window will be different (i.e. the TCP Cubic mode, the buffer occupancy). Thus, we report the window size expression for the four states {A, B, C, D} in Table 2.3.

We also present the expression of window sizes for all transitions between states in Table 2.4. W_{ij} represent the window size when a flow move from State i to State j.

States	Window size W(t)
State A	$w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)}\frac{t}{\tau}$ (Equation 2.5)
State B	$C(t-V_{cubic})^3+wmax$ (Equation 2.1)
State C	$\frac{(w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)})}{2}+\frac{1}{2}\times((\frac{3\beta}{b(2-\beta)}+w_{max}(1-\beta))^2+4(\frac{3\beta Ct}{b(2-\beta)}-\frac{3\beta w_{max}(1-\beta)}{b(2-\beta)}))^{\frac{1}{2}}$ (Equation 2.11)
State D	$C(t-V_{cubic})^3+w_{max}$ (Equation 2.1)

TABLE 2.3: Window size at time t W(t)

Transition	Window size	According to
$<A\rightarrow B>$	$W_{AB}=w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)}\frac{t_{AB}}{\tau}$	Equation 2.5
$<A\rightarrow C>$	$W_{AC}=w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)}\frac{t_{AC}}{\tau}$	Equation 2.5
$<B\rightarrow D>$	$W_{BD}=C(t_{BD}-V_{cubic})^3+w_{max}$	Equation 2.1
$<B\rightarrow A>$	$W_{BA}=C(t_{BA}-V_{cubic})^3+w_{max}$	Equation 2.1
$<C\rightarrow D>$	$W_{CD}=\frac{(w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)})}{2}+\frac{1}{2}\times((\frac{3\beta}{b(2-\beta)}+w_{max}(1-\beta))^2$ $+4(\frac{3\beta Ct_{CD}}{b(2-\beta)}-\frac{3\beta w_{max}(1-\beta)}{b(2-\beta)}))^{\frac{1}{2}}$	Equation 2.11
$<C\rightarrow E>$	$W_{CE}=C\tau+B+1$	X
$<D\rightarrow C>$	$W_{DC}=C(t_{DC}-V_{cubic})^3+w_{max}$	Equation 2.1
$<D\rightarrow E>$	$W_{DE}=C\tau+B+1$	X

TABLE 2.4: Window size when moving between states

The sojourn time in one state can be used in several criteria calculation, such as the calculating the length of a cycle. This criterion gives an indication of the stability of the behavior of each connection. We report in Table 2.5 the sojourn time since the beginning of the cycle until completion of each state. For more details regarding the calculation of these times, see Appendix A.

Transition	Residence time
$<A\rightarrow B>$	$t_{AB}=\frac{3}{2}V_{cubic}+/-\sqrt{\sigma}$
$<A\rightarrow C>$	$t_{AC}=\frac{(2-\beta)b\tau}{3\beta}(\tau C-(1-\beta)w_{max})$
$<B\rightarrow D>$	$t_{BD}=\sqrt[3]{\frac{(C\tau-w_{max})}{C_{cubic}}}+V_{cubic}$
$<B<\rightarrow A>$	$t_{BA}=\frac{3}{2}V_{cubic}+/-\sqrt{\sigma}$
$<C\rightarrow D>$	$t_{CD}=\frac{3}{2}V_{cubic}+/-\sqrt{\sigma}$
$<C\rightarrow E>$	$t_{CE}=\frac{b(2-\beta)}{3\beta C}(\frac{1}{4}(\frac{b(2-\beta C)}{3\beta C}\times 4(C\tau+B+1-\frac{(w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)})^2}{2})^2$ $-((w_{max}(1-\beta)+\frac{3\beta}{3\beta C})^2)+\frac{3\beta w_{max}(1-\beta)}{b(2-\beta)})$
$<D\rightarrow B>$	$t_{DB}=$
$<D\rightarrow E>$	$t_{DE}=\sqrt[3]{\frac{C\tau+B+1-w_{max}}{C}}+V_{cubic}$

TABLE 2.5: The sojourn time since the beginning of the cycle until completion of the state

There is another important criteria which is the number of packets sent in one cycle, see Table 2.6. To find this quantity of packets between two epochs t_1 and t_2 we integrate the window size W(t) between these two times: $n_{t_1 t_2} = \int_{t_1}^{t_2} W(t)dt$.

Transition	Number of packets
$< A \rightarrow B >$	$n_{AB} = w_{max}(1-\beta)t_{AB} + \frac{3\beta}{2b(2-\beta)}\frac{t_{AB}^2}{\tau}$
$< A \rightarrow C >$	$n_{AC} = w_{max}(1-\beta)t_{AC} + \frac{3\beta}{2b(2-\beta)}\frac{t_{AC}^2}{\tau}$
$< B \rightarrow D >$	$n_{BD} = \frac{C}{4}(t_{BD} - V_{cubic})^4 + w_{max}t_{BD}$
$< B \rightarrow A >$	$n_{BA} = \frac{C}{4}(t_{BA} - V_{cubic})^4 + w_{max}t_{BA}$
$< C \rightarrow D >$	$n_{CD} = C(t_{CD} - t_{AC})$
$< C \rightarrow E >$	$n_{CE} = C(t_{CE} - t_{AC})$
$< D \rightarrow C >$	$n_{DC} = C(t_{DC} - t_{AD})$
$< D \rightarrow E >$	$n_{DE} = C(t_{DE} - t_{AD})$

TABLE 2.6: The number of packet sent during a transition

2.3.1.2 Combining Multiple Phases

As we said before that the evolution of the TCP Cubic window is characterized by five different states, it is possible to identify 11 different ways of combining these states. Each particular realization will take a different path, depending on the specific values of all the parameters involved (i.e., the BDP and on the buffer size B).

- Case 1 : States (A)-(B)-(D)-(E)
- Case 2 : States (A)-(C)-(E)
- Case 3 : States (A)-(C)-(D)-(E)
- Case 4 : States (C)-(D)-(E)
- Case 5 : States (C)-(E)
- Case 6 : States (B)-(D)-(E)
- Case 7 : States (B)-(D)-(C)-(E)
- Case 8 : States (B)-(A)-(C)-(E)
- Case 9 : States (B)-(A)-(C)-(D)-(E)
- Case 10 : States (D)-(C)-(E)
- Case 11 : States (D)-(E)

2.3.2 RTT Analysis

According to Equation 2.3, the Round Trip Time depends on three parameters: the capacity C; the delay τ; and the buffer Q(t) which depends on the state variable. We identify two cases, depending on the buffer occupancy:

- Empty buffer, Q(t)=0

If Q(t)=0, then the window size $W(t)$ at time t is smaller than $C\tau$, so in this case $R(t) = \tau$ (Equation 2.3). According to Equation 2.2, the expression of the congestion window in TCP mode will be as follows:

$$W_{tcp}(t) = w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)}\frac{t}{\tau} \tag{2.5}$$

- Non empty buffer, Q(t)>0

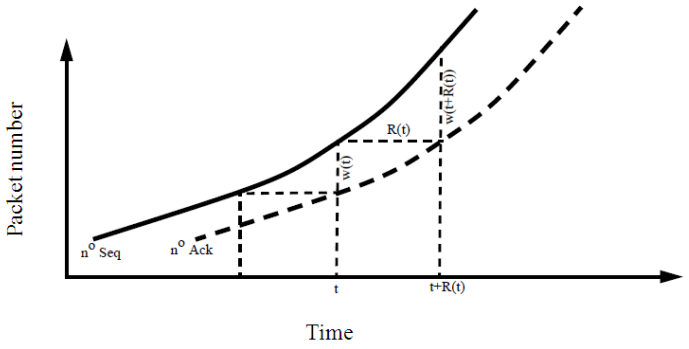

FIGURE 2.5: Trace of the sequence and acknowledgment numbers when there is no loss [Col98]

Figure 2.5 provides a graphical description of the window size as function of the sequence and acknowledgment numbers. The window size at time t W(t) is a function of the value at the previous time (t-R(t)). According to Equations (2.3) and (2.4), where W(t) $\geq C\tau$

$$R(t) = \frac{W(t - R(t))}{C} \tag{2.6}$$

To simplify the calculation, we derive both a lower and an upper bound.

2.3.2.1 The upper bound for R(t)

The upper bound for R(t) according to Equation 2.6 and without considering delay (i.e., lower bound for W(t)) is as follows :

$$R(t) = \frac{W(t)}{C}$$

Where R(t) is expressed in seconds, W(t) in packets and C is expressed in packets per second. Consider the TCP mode of TCP Cubic, where $W(t) = W_{tcp}$. If we change RTT by its value in Equation 2.2, we obtain :

$$W(t) = w_{max}(1-\beta) + \frac{3\beta C}{b(2-\beta)}\frac{t}{W(t)} \tag{2.7}$$

Thus, we have:

$$W^2(t) - w_{max}(1-\beta)W(t) - \frac{3\beta C}{b(2-\beta)}t = 0$$

The solutions to this quadratic equation are:

$$\triangle = (w_{max}(1-\beta))^2 + 4\frac{3\beta C}{b(2-\beta)}t$$

The roots are given by:

$$W(t) = \frac{(w_{max}(1-\beta)) \pm \sqrt{\triangle}}{2} \tag{2.8}$$

The positive solution is the only possible one, compatible with W(t) that increases, so W(t) is given by:

$$W_{tcp}(t) = \frac{(w_{max}(1-\beta))}{2} + \frac{1}{2} \times ((w_{max}(1-\beta)^2 + \frac{12\beta C}{b(2-\beta)} \times t)^{\frac{1}{2}} \tag{2.9}$$

2.3.2.2 The lower bound for R(t)

The lower bound for R(t) according to Equation 2.6 and considering delay (upper bound for W(t)) is as follows :

$$W(t) = w_{max}(1-\beta) + \frac{3\beta C}{b(2-\beta)}\frac{t}{(W(t) - \frac{3\beta}{b(2-\beta)})} \tag{2.10}$$

When applying the same calculation as in the previous section, we obtain:

$$W_{tcp}(t) = \frac{(w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)})}{2} + \frac{1}{2} \times ((\frac{3\beta}{b(2-\beta)} + w_{max}(1-\beta))^2 + 4(\frac{3\beta C t}{b(2-\beta)} - \frac{3\beta w_{max}(1-\beta)}{b(2-\beta)}))^{\frac{1}{2}} \tag{2.11}$$

When building our analytical model, we have a problem to estimate the R(t), so we considered the lower bound of R(t), Equation 2.11.

2.4 Numerical results

2.4.1 Algorithm versus simulation

Through the implementation of the model for a TCP Cubic, we identified slight differences between the specifications mentioned in the Cubic paper and its implementation in TCP Linux from NS-2 which is supposed to be the same as the one in some Linux kernels. Some of these details were not well explained in the TCP Cubic paper [HRX08].

2.4.1.1 Fast Convergence

Fast Convergence is an optional mechanism that allows the flow to accommodate the change in the available bandwidth, its principle is as follows : when a loss occurs, the value of the congestion window is saved in a parameter $last_{max}$. If the current cwnd is smaller than $last_{max}$, this indicates that the saturation point experienced by this flow is getting reduced. So, we allow this flow to free more bandwidth by further reducing w_{max} and then set $last_{max}$ to $cwnd * (1 - \frac{\beta}{2})$. The slow start threshold *ssthresh*, which is called when the given TCP detects a loss, is always set to $cwnd(1 - \beta)$.

An important parameter is V_{cubic}, which represents the time period required by an isolated TCP Cubic connection to increase $w_c(t)$ from $(1 - \beta)w_{max}$ to w_{max} when there is no further loss event. This quantity is calculated by using the following equation:

$$V_{cubic} = \sqrt[3]{\frac{\beta w_{max}}{C_{cubic}}}$$

But when we look to the specification of the algorithm in [HRX08], we found that $V_{cubic} = \sqrt[3]{\frac{last_{max} - W(t)}{C}}$, where W(t) is the current window size. And as the $last_{max}$ value depends on the Fast Convergence mechanism, this quantity will have 2 different expressions.

Assume that the convergence window at a loss event is equal to w_{max} and window size after a loss is $W(t)$.

- **Case 1** : There is no Fast Convergence

For this scenario $W(t) = (1 - \beta)w_{max}$ and $last_{max} = w_{max}$ so :

$$V_{cubic} = \sqrt[3]{\frac{\beta w_{max}}{C_{cubic}}} \tag{2.12}$$

- **Case 2** : There is Fast Convergence

The window size $w(t)$ will be equal to $(1-\beta)w_{max}$ and $last_{max}$=$(1-\frac{\beta}{2})w_{max}$. If we consider $V_{cubic} = \sqrt[3]{\frac{last_{max}-w(t)}{C_{cubic}}}$, we get:

$$V_{cubic} = \sqrt[3]{\frac{\beta W_{max}}{2C_{cubic}}} \tag{2.13}$$

To verify the accuracy of V_{cubic} expressions we build our analytical model using C++, we thus performed simulations with both the model and NS-2 simulator.
Consider the scenario shown in Figure 2.3 in which a client is connected to a server by a 100Mbps link with a 50ms propagation delay and a buffer of 50 packets and a BDP is equal to 416 packets.

We report in Figure 2.6 the time series for the window size for the model and NS-2 simulations. The blue curve (blue triangles) presents the model behavior when considering only the first expression of V_{cubic} in Equation 2.12 , while the red curve (red stars) represents the NS-2 results.

We observe that TCP Cubic operates in the Cubic mode. This is consistent with the calculation that was done in the Section 2.2.2, were the RTT_{min} ensuring that TCP Cubic is in Cubic mode for 100Mbps is equal to 39ms.

We note that without the Fast Convergence the loss is not always detected at the same level. And the matching between the model and the simulation is less good. We conclude from these results that it is more correct to consider both Fast Convergence expressions.

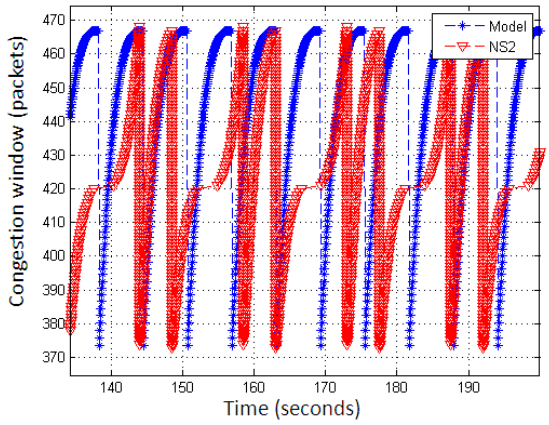

FIGURE 2.6: Fast Convergence phenomenon, $C = 100Mbps, \tau = 50ms$, B=50 packets

2.4.1.2 Delayed Acknowledgments

To avoid overloading the network with acknowledgments (ACKs), the destination does not return an acknowledgment for each packet received, several ACK responses

may be combined together into a single response.

Consider the TCP window in Equation 2.2, the proper expression should be as follow:

$$W_{tcp}(t) = w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)}\frac{t}{R(t)} \tag{2.14}$$

Where b is the number of received packets after which the TCP destination sends an acknowledgment. We consider b=1 (the recommended value is 2). It is usually more in high speed networks with recent OS (typically 6). The parameter b is not marked in the original paper of TCP Cubic, but we realized that it is taken into account by NS-2.

We considered a scenario where TCP Cubic operates in the TCP mode, the streaming online, with a link capacity equal to 200Mbps, a latency equal to 10ms and a buffer size equal 50 packets.

We report in Figure 2.7 the time series for the window size for the model as well as the NS-2 simulation for b=2. We notice that the model and the simulation slopes are different. So, simulation results show that the delayed ACK misbehaves with TCP Cubic, so we disable it for the next simulations (i.e., b=1).

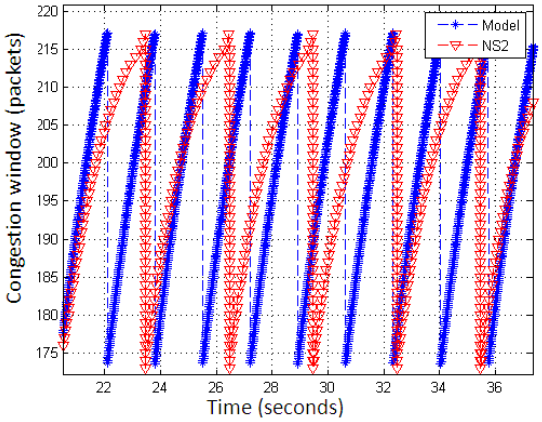

FIGURE 2.7: Delayed Acknowledgments phenomenon, $C = 200Mbps, \tau = 10ms$, B=50 packets

2.4.1.3 Entire packet

We considered the same scenario as in the Section 2.4.1.2. We report in Figure 2.8 the time series for the window sizes. We notice that the loss is not always detected at the same level (a difference in fractions of packet) which triggers the Fast Convergence phenomenon. With simulations, we do not see the same behavior since they consider whole packets. So we set also this quantity to be an integer for the remaining tests.

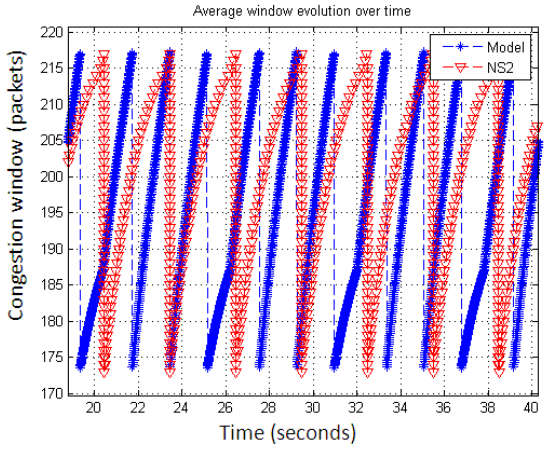

FIGURE 2.8: Entire packet phenomenon, $C = 200Mbps, \tau = 10ms$, B=50 packets

2.4.2 Validation

In this section, we present validation results based on a set of scenarios. Our approach for validation is to compare the analytical model results against NS-2 simulations, that last 500 seconds. The TCP packet size is set to 1500 bytes.

We considered one metric to perform this comparison, which is the time series evolution of the congestion window.

2.4.2.1 ADSL scenario

As a first scenario, we consider $C = 10Mbps, \tau = 50ms$ and the queue B=10 packets. The BDP is equal to 42 packets. We report in Figure 2.9 the time series of window size for model and simulations. We note that the model behavior is close to the NS-2 results. But, NS-2 results are sometimes faster compared to the model.

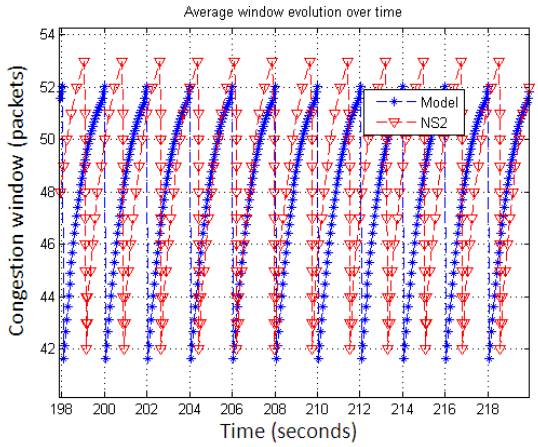

FIGURE 2.9: Time series of the window, ADSL, $C = 10Mbps$

2.4.2.2 FTTH scenarios

We consider 2 FTTH scenarios with a capacity equal to $C = 100Mbps$ (resp. $C = 200Mbps$), a latency equal to $\tau = 50ms$ (resp. $\tau = 10ms$), and a same buffer size of B=50 packets. So a BDP=417 packets. TCP Cubic operates in the Cubic mode for the first scenario, while it is in the TCP mode for the second one.

We present in Figures 2.10 and 2.11 the time series of window sizes for the two scenarios. We observe a very good temporal match both in terms of the amplitude variations and in the frequency of oscillations of the this metric.

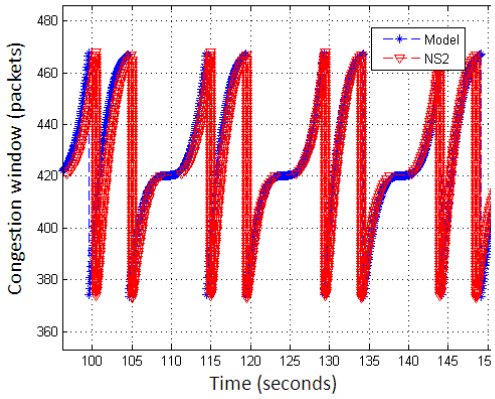

FIGURE 2.10: Time series of the window, FTTH, $C = 100Mbps$

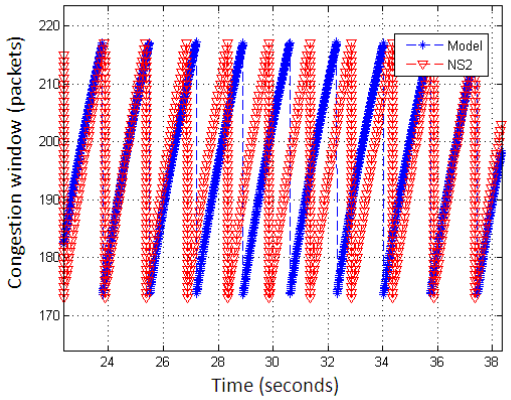

FIGURE 2.11: Time series of the window, FTTH, $C = 200Mbps$

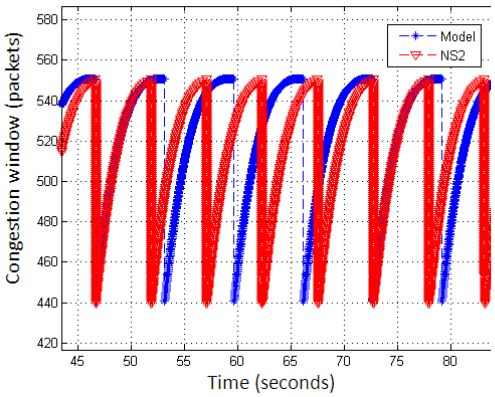

FIGURE 2.12: Time series of the window, FTTH, $C = 300Mbps$

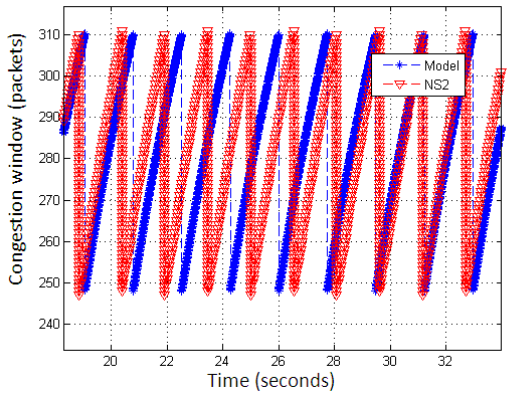

FIGURE 2.13: Time series of the window, FTTH, $C = 400Mbps$

We considered 2 other FTTH scenarios with 300Mbps and 400Mbps. A latency of 20ms (resp. 8ms) and a buffer size equal to 50 packets (resp. 43 packets). The model behavior is similar to NS-2, the difference is that the model takes a little more time to reach the loss value.

From the above results, we can see that while the model does not always match the simulations with great precision, it does accurately predict the overall evolution of the window and it allowed us to identify the different states of TCP Cubic.

2.5 Conclusion

The focus of this chapter was on analytical models for a single-long lived TCP connection. We have highlighted some details about the algorithm of TCP Cubic that were not well explained in TCP Cubic paper. Then, we proposed our own model for one TCP Cubic connection. Using this model, we have analyzed the window size, showing how this metric depends on the different modes of TCP Cubic and the buffer size.

While this is a simple model, it allows us to identify the different TCP Cubic behaviors. It is actually used by one of the Orange Labs team for troubleshooting TCP connections.

Recently, there has been a growing interest in large-scale modeling of TCP flows. Especially in cloud environments, where the number of flows sharing a bottleneck link is huge. So, in the next chapter, we introduce an analytical model which allows us to analyze the scaling behavior of large number of TCP Cubic.

Chapter 3

Understanding TCP Cubic Performance in the Cloud: a Mean-field Approach

3.1 Introduction

In this chapter, we aim at developing an analytical model for TCP Cubic to analyze its performance in typical cloud scenarios where a large number of long-lived TCP connections, e.g., HTTP streaming or back-up flows, share a bottleneck link.

We rely on a Mean-field approach leading to a fluid model to analyze the performance of TCP Cubic. After a careful validation of the model through comparisons with NS-2, we evaluate the efficiency and fairness of TCP Cubic as compared to that of New Reno for a set of typical cloud networking scenarios.

3.2 Background

3.2.1 TCP Cubic

Several analytical models have been proposed in the literature to analyze the performance of legacy TCP versions, but there are fewer for TCP Cubic. The authors of [CEH+07], [BWL10], [BAC09] consider a single long-lived flow. In [PS11], Poojary and Sharma investigate the cases of three TCP Cubic connections as well as the competition between a TCP Cubic and New Reno connection. In [BAC09], Blanc et al. compare the performance of TCP Cubic, Compound TCP, HighSpeed TCP and New Reno under a simple loss model, where each packet can be randomly dropped with probability p. They model the evolution of the congestion window with a Markov chain to compute the average window size, its coefficient of variation (CoV) and the average throughput. They find that, for small bandwidth delay products, TCP Cubic can have a similar

throughput to New Reno while for larger values the throughput of all new TCP versions behave similarly and outperform New Reno.

A few studies have investigated TCP Cubic in vivo using experimental testbeds. In [JR11], Jain and Raina observed that at smaller link capacities, there is a detrimental impact on utilization as the number of users gets smaller or the round trip times gets higher. At larger link capacities, small buffers can readily induce synchronization effects. In [LSM07], Leith et al. observed in their testbed that at higher speeds, for buffer sizes below 30% of the BDP, the link utilization achieved by TCP Cubic collapses to around 50% and is significantly lower than the link utilization achieved by standard TCP. We revisit this question in Section 3.5.

3.2.2 Mean-field models

Mean field approximations, or equivalently mean-field limits, date back to the seventies [Kur70] and are used to analyze the limit behavior of systems made of N objects, as N tends to infinity. As the limit process is the solution of a deterministic ordinary differential equation (ODE), it is referred to as a fluid limit, or fluid model, as well. Depending on the underlying interaction process between the objects, articles such as [BLB08, BW09, BMP10] have refined the results of [Kur70] and exemplified with various applications in computer science.

Baccelli et al. introduced a mean-field model for a set of N TCP Reno connections in [BMR02]. In this paper, the authors consider a bottleneck router implementing the RED (Random Early Discard) active queue management policy, and they assume that the connections have reached equilibrium, i.e., TCP operates in congestion avoidance mode and does not experience time-outs. The model is derived, but the focus is then put on the fixed points of the mean field equations. We build on this work to obtain a mean field model of TCP Cubic. The model for TCP Cubic is an extension of [BMR02] in that it is more complex: two parameters instead of one now define the state of an object, and the time of the last loss is an additional quantity that must be approximated. Furthermore, as our purpose is to investigate the performance in cloud networking scenarios, we prove a scaling property of the model that allows us to run the model equations for any network scenario, without increasing the computational cost that may arise from state explosion. Finally, we validate extensively our model on cloud networking scenarios, trying to assess its domain of validity by identifying simulation behavior that it cannot capture. The model is used to investigate the performance of both TCP Cubic and TCP New Reno.

3.3 Performance analysis of TCP Cubic

3.3.1 A fluid model for TCP Cubic

We build on [BMR02] and consider N TCP Cubic connections routed through a bottleneck link whose aggregate capacity is $N \times L$ packets per second. The queue size at the sending buffer of the bottleneck link router is denoted by $Q^{(N)}(t) = Nq^{(N)}(t)$, the buffer size being $N \times B$. Let $S_n^{(N)}(t) =< w^{(n)}(t), w_{max}^{(n)}(t) >$ be the state of connection n, for $n = 1, \ldots, N$ at time t. All connections have the same latency (i.e., the same $baseRTT$). We assume without loss of generality, that the RTT measured at the sender side is the same for all connections, and is denoted by $R^{(N)}(t)$. Hence $R^{(N)}(t)$ can be expressed recursively by

$$R^{(N)}(t) = baseRTT + \frac{q^{(N)}\left(t - R^{(N)}(t)\right)}{L} \tag{3.1}$$

In order to express all quantities governing the connection states in terms of an absolute time variable, t is changed to $t - s_{loss}^{(n)}(t)$ in Equations (2.1) and (2.2), where $s_{loss}^{(n)}(t)$ denotes the elapsed time since the last loss seen by the n-th TCP sender.

Our goal is to predict the performance of the system of N TCP Cubic connections thanks to a fluid model, stemming from a mean-field approximation. Let's consider the limit behavior of the system when N tends to infinity, so as to get fluid model of the performance. Considering $\mathbf{Y}^{(N)}(t) = (S_1^{(N)}(t), \ldots, S_N^{(N)}(t))$ as the state of the system of interest with N connections. $\mathbf{Y}^{(N)}(t)$ is an homogeneous Markov chain that can be shown to be a mean-field interaction model with N objects, as defined in [BLB08]. We define the occupancy measure as the fraction of connections in each state at each time t, and denote it by $p^{(N)}(t, w, w_{max})$ for time t and state $< w, w_{max} >$. Theorem 3.1 of [Kur70] ensures that, as $N \to \infty$, for any $t > 0$ and $< w, w_{max} > \in E$, $p^{(N)}(t, w, w_{max})$ converges uniformly almost surely to the solution $p(t, w, w_{max})$ of the coupled Ordinary Differential Equations (ODE) below, with initial condition $p(0, (1-\beta)x, x) = 1$ (x is set to 5 in the experiments of Section 3.4). Additionally, the other quantities of interest, namely $q^{(N)}(t)$, $R^{(N)}(t)$ and $s_{loss}^{(n)}(t)$ can be expressed thanks to their deterministic fluid limits $q(t)$, $r(t)$ and $s_{loss}(t)$, respectively. In particular, $q^{(N)}(t)$ is approximated by $q(t)$ given in Equation (3.3). It is worth noting that the convergence to the fluid limit holds in the presence of $q^{(N)}(t)$ that can be considered as a resource, as defined and proven in [BLB08]. The equations below are ODE expressed in the case w and w_{max} take integer values. These equations can be easily adapted to the continuous case such as in [BMR02]. Note however that the time remains continuous.

$$\frac{dp(t, w, w_{max})}{dt} =$$

$$
\begin{aligned}
& \Big\{ \frac{w}{(1-\beta)} \frac{1}{r(t)} \delta \left(w_{max}, \frac{w}{(1-\beta)} \right) \sum_{v=1}^{W} p \left(t, \frac{w}{(1-\beta)}, v \right) \\
& - \frac{w}{r(t)} p\left(t, w, w_{max}\right) \Big\} k(t - r(t)) \\
+ \quad & \Big\{ - A p(t, w, w_{max}) \\
& + A \frac{(w-1)}{r(t)} p\left(t, (w-1), w_{max}\right) \Big\} (1 - k(t - r(t)) \qquad (3.2)
\end{aligned}
$$

$$
\frac{dq(t)}{dt} = \begin{cases} (1 - k(t)) \sum\limits_{v,w=1}^{W} \frac{w}{r(t)} p(t, w, v) - L & \text{, if } q(t) > 0 \\ 0 & \text{, otherwise} \end{cases} \qquad (3.3)
$$

$$
r(t) = baseRTT + \frac{q(t)}{L} \qquad (3.4)
$$

All the equations describing the system involve per-connection quantities. In the above equations, W denotes the maximum possible value of w and w_{max}. Note that the term $\sum\limits_{v,w=1}^{W} \frac{w}{r(t)} p(t, w, v)$ corresponds to the mean rate injected into the network per connection. The probability that a packet be dropped by the bottleneck buffer at time t is denoted by $k(t)$. As we consider the droptail policy as buffer management, we have:

$$
k(t) = \begin{cases} 0, \text{ if } q(t) \leq B \text{ or } & \sum\limits_{v,w=1}^{W} \frac{w}{r(t)} p(t, w, v) < L \\ 1 - \frac{L}{\sum\limits_{v,w=1}^{W} \frac{w}{r(t)} p(t,w,v)} & \text{, otherwise} \end{cases} \qquad (3.5)
$$

The parameter $s_{loss}(t)$ denotes the average absolute time of the last loss before time t. It is estimated thanks to the intensity $i(t)$ of the loss process, assumed to be Poisson as in [BMR02]:

$$
\frac{di(t)}{dt} = k(t - r(t)) \sum_{v,w=1}^{W} \frac{w}{r(t)} p(t, w, v) \; .
$$

Then we take

$$
s_{loss}(t) = \begin{cases} 0 & \text{, if } i(t) < 1 \\ t - \frac{t}{i(t)} & \text{, otherwise} \end{cases} \qquad (3.6)
$$

We assume losses to be uniformly distributed over time, see Figure 3.1.

Parameter A in Equation (3.2) denotes the increase of the congestion window $w(t)$ between t and $t + dt$. Depending on the mode of operation (either Cubic or TCP), A is hence the time derivative of $w_c(t)$ or $w_{tcp}(t)$ given in Equations (2.1) and (2.2): $A = 3C(t - s_{loss}(t) - V_{cubic})^2$ or $A = \frac{3\beta}{2-\beta} \frac{1}{r(t)}$. Note also that taking $A = \frac{1}{r(t)}$ readily gives back the results of [BMR02] for TCP New Reno. Each of the four terms in Equation (3.2) corresponds, in order of appearance, to connections arriving in and leaving state $< w, w_{max} >$ in the time interval $[t, t + dt[$ because a loss is detected, and connections

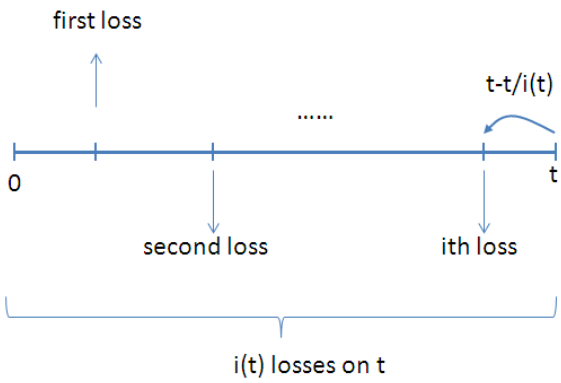

FIGURE 3.1: The elapsed time until the last loss $s_{loss}(t)$

leaving and arriving in state $< w, w_{max} >$ because of window increase in case no loss is detected. Figure 3.2 show the 4 possible transitions from end to a state w_c, w_{max}.

Eventually, obtaining the above quantities (the distribution of the connection states, the queue size, the RTT and the loss probability) allows us to predict all performance metrics of interest, such as throughput, goodput, buffer occupancy, mean window size and the distribution of windows. These performance metrics are discussed in Sections 3.4 and 3.5.

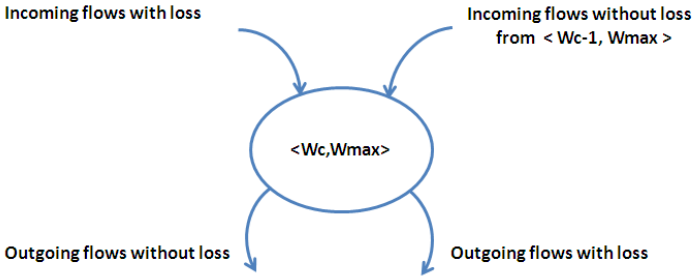

FIGURE 3.2: Connections arriving in and leaving state $< w, w_{max} >$ in the time interval $[t, t + dt[$

3.3.2 Decreasing the computational intensity of the model

Our model builds on the work of Baccelli et al. in [BMR02], but is more computationally complex as the window size evolution in TCP Cubic is governed by two parameters ($w(t)$ and $w_{max}(t)$). Therefore, the number of states for a connection is W^2. When implementing the model, e.g., in matlab as explained in the next section, the size of the occupancy vector is hence the square of that with TCP New Reno. This leads to problems when we intend to use the model in high BDP scenarios, which are common in cloud networking, that is when a high value of W is required. In order to overcome

this issue, we exploit a scaling property of the model, that allows to run the model for a down-scaled scenario and extrapolate the results to an up-scaled scenario.

Let us consider a scenario, that we call Scenario 2, characterized by L_2, B_2 and $baseRTT$. The quantities $p_2(t_2, w_2, w_{max_2})$, $q_2(t_2)$, $r_2(t_2)$ and $k_2(t_2)$ can be deduced from $p_1(t_1, w_1, w_{max_1})$, $q_1(t_1)$, $r_1(t_1)$ and $k_1(t_1)$, the quantities of Scenario 1 with features $L_1 = 1/\alpha L_2$, $B_1 = 1/\alpha B_2$ and $baseRTT$, with $\alpha > 1$. Let us sketch how this can be achieved and why. We look for a function $f(.,.,.)$ such that $(t_2, w_2, w_{max_2}) = f(t_1, w_1, w_{max_1})$. As the BDP is scaled by α, we set $w_2 = \alpha w_1$ and $w_{max2} = \alpha w_{max1}$. We then look for t_2 as a function of t_1 such that the following condition be fulfilled:

$$p_2(t_2, w_2, w_{max_2}) = \frac{1}{\alpha} p_1(t_1, w_1, w_{max_1}) \ . \tag{3.7}$$

This condition comes from the mass conservation (i.e., the distribution function must sum up to 1). We re-write condition 3.7 as

$$\frac{\partial p_2(t_2, w_2, w_{max_2})}{\partial t_2} = \frac{1}{\alpha} \frac{\partial p_1(t_1, w_1, w_{max_1})}{\partial t_2}$$

and unfold the right-hand side by making appear a partial derivative with respect to t_1:

$$\frac{1}{\alpha} \frac{\partial p_1(t_1, w_1, w_{max_1})}{\partial t_2} = \frac{1}{\alpha} \frac{\partial p_1(t_1, w_1, w_{max_1})}{\partial t_1} \frac{dt_1}{dt_2} \ .$$

Substituting the middle term with Equation (3.2) allows to get back the left-hand side of condition 3.7 assuming that:

- $t_2 = \alpha t_1$ in TCP mode, $t_2 = \alpha^{1/3} t_1$ in Cubic mode (as it depends on the value of A),
- $k_2(t_2) = k_1(t_1)$ and $r_2(t_2) = r_1(t_1)$,
- $q_2(t_2) = \alpha q_1(t_1)$,
- the probability of loss ($k_1(t_1)$) is low. This last assumption comes from the fact that the term in front of $k(t - r(t))$ in Equation (3.2) needs to be neglected so as to retrieve condition 3.7 with the above mentioned scaling in t. It is important to note that such assumption makes sense because we consider the steady phase (congestion avoidance) of a TCP connection, where by definition the TCP sender strives to avoid loss events. Furthermore, the relevance of this assumption is verified in the simulations below which are obtained using the scaling property.

It is also worth noting that the scaling for the TCP mode also applies to the model for New Reno. We report in Figure 3.3 a simple example of the scaling factor α for New Reno. After a loss event the window decreases by 50%. Suppose we have the scenario1,

When we apply the scaling factor α to the it, the maximum window D_1 and the time T_1 for the first scenario are multiplied by a factor α, and we got $D_2 = \alpha D_1$ and $T_2 = \alpha T_1$.

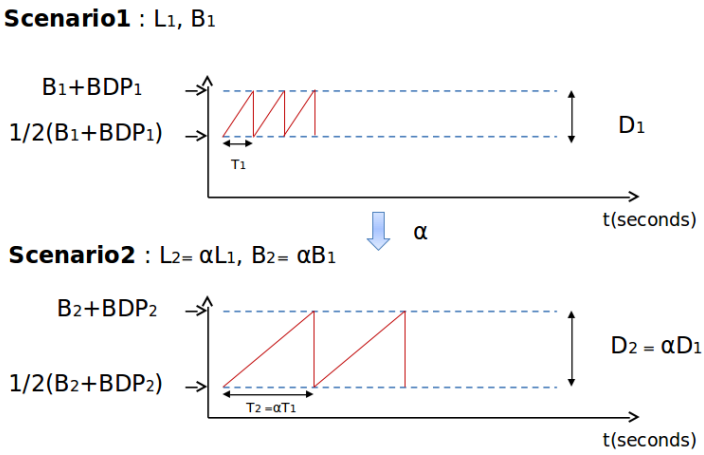

FIGURE 3.3: Scaling factor α for New Reno

Regarding the mode of operation of TCP Cubic, the model allows to choose it on the fly, as it is done in real-world implementations. However, in order to be able to use the scaling property and reduce the computational intensity of the model for high BDP scenarios, we need to force the mode of operation of the down-scaled scenario (here above named Scenario 1) so as to make it similar to that of the up-scaled scenario of interest. The latter is determined based on L_2, B_2 and $baseRTT$ thanks to the condition presented in Section 2.2.2. Forcing the mode is done by imposing the right expression to A, and then the time scaling is chosen in accordance to the preset mode.

3.4 Numerical validation

In this section, we present validation results based on the three scenarios. Our approach for validation is to compare the fluid model results against NS-2 simulations. The former are obtained using a numerical ODE solver of matlab. The TCP packet size is set to 1480 bytes. We set the number of connections in NS-2 to $N = 10$. We consider several metrics to perform this comparison:

(i) The time-series of average window size and instantaneous queue size. The former is obtained by computing the average window over all connections at each given time instant, while the former is obtained from Equation (3.3).

(ii) The marginal distribution of the window size. It is obtained by considering a large time interval where the model has apparently reached equilibrium, gathering the samples of window sizes at each time instant to form the corresponding cumulative distribution function (CDF).

(iii) The time-series of the goodput. The goodput is simply the throughput multiplied by the packet loss probability and thus relates to the traffic that the server could effectively service over a given time period.

We present results by dividing them into three parts. First, the intra-DC and FTTH scenarios for TCP Cubic. Then, the inter-DC scenario. Eventually, we present results for New Reno in the FTTH scenario.

3.4.1 Network scenarios

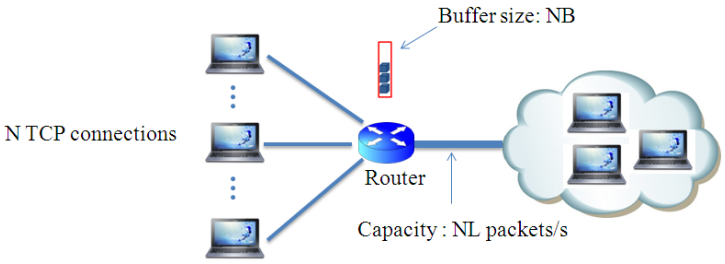

FIGURE 3.4: Network scenario indicating the N flows, buffer and servicing link

In this chapter, we assume the network is in steady state, i.e., the TCP Cubic sender has reached equilibrium and operates in the congestion avoidance mode. Thus, we neglect, in line with the approach in [BMR02], the slow start phase of TCP. However, it is worth noting that our model could be extended to encompass the transient behaviors. We further assume that losses are recovered using fast-retransmit and not time-out

We consider a classical dumbbell topology with N TCP senders, N TCP receivers and a shared bottleneck with fixed capacity $N \times L$ and fixed buffer size $N \times B$, shown in Figure 3.4. The latency of the path between each pair of sender and receiver is fixed and equal to $baseRTT$. L and B can thus be seen as the allocated server and buffer capacities per flow. We assume FIFO/droptail as server/queue management policy at the bottleneck, as it is the prevalent policy in todays network, including data-centers. The three scenarios we focus on correspond to the following choices of L, $baseRTT$ and B:

Scenario A - FTTH-client: this scenario models the case of high-speed clients, with FTTH access, that are simultaneously downloading from a DC. We thus consider $L = 100Mb/s$ and $baseRTT = 20ms$, and take the buffer size $BS = 50$ packets. The value of $baseRTT$ corresponds to typical RTTs observed for FTTH clients [HUKC+11], especially when they access well-provisioned servers. This is in contrast with DSL access where the latency on the last mile typical represents around 50 ms of the total RTT.

Scenario B - Intra-DC: we consider $B = 1Gb/s$ and $baseRTT = 1ms$, as servers in a typical DC are equipped with 1 Gb/s NICs and the end-to-end delay observed in DC are in the order of a ms [BAM10]. We also take $B = 50$ packets

Scenario C - Inter-DC: we consider a dedicated link connecting two DCs that are far apart. Hence, we take $B = 1Gb/s$ (remember that it corresponds to the average bandwidth per flow), $baseRTT = 50ms$ and $B = 500$ packets.

3.4.2 FTTH and intra-DC scenarios

We have grouped the FTTH and intra-DC scenarios together as TCP Cubic operates in the same TCP mode in both cases. We present in Figures 3.5 and 3.6 the time series of average window size and queue sizes for the two scenarios. Once the simulation and the model have reached equilibrium (which takes a longer time for the simulation as we do not account for the slow-start phase of TCP in the fluid model), we observe a very good temporal match both in terms of the variation of amplitudes and in the frequency of oscillations of the two metrics. This is further confirmed by the distributions of window sizes in Figures 3.7 and 3.8.

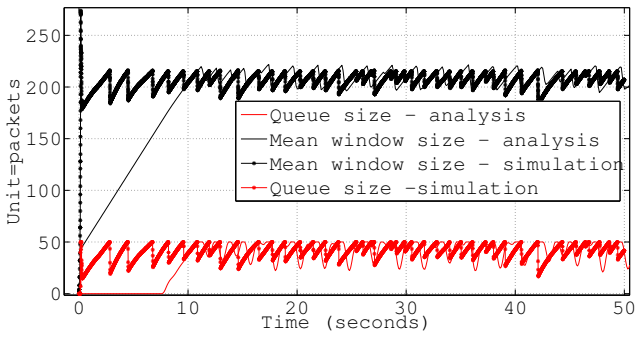

FIGURE 3.5: Time series of queue size and average window size - FTTH scenario - TCP Cubic

3.4.3 Inter-DC scenario

In the Inter-DC scenario, owing to the large bandwidth delay product of the path and the high RTT, TCP Cubic operates in the Cubic mode. The matching between the model and the simulation is less good in this scenario, as it can be observed from Figure 3.9. Our fluid model tends to over-estimate the average window as compared to the simulation. This is confirmed by Figure 3.10, which presents the marginal distribution of window sizes. What our model does not capture is in fact the loss synchronization effect among the sources that occurs in the simulation. Indeed, the shape of the average window time series in Figure 3.9 indicates that almost all connections experience loss simultaneously

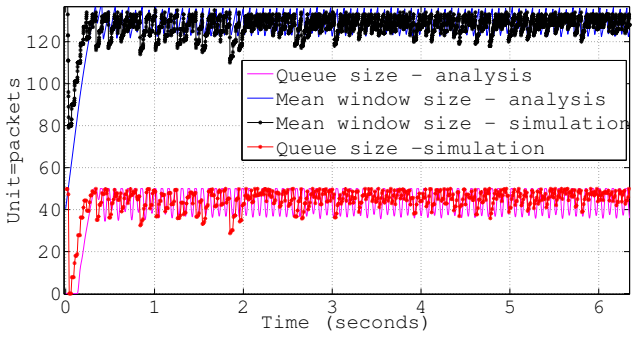

FIGURE 3.6: Time series of queue size and average window size - Intra-DC scenario - TCP Cubic

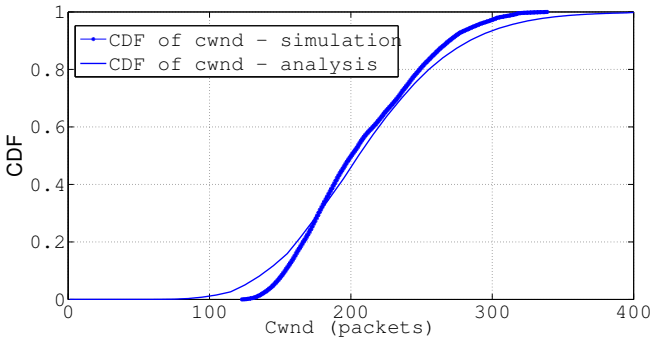

FIGURE 3.7: Congestion window - FTTH scenario - TCP Cubic

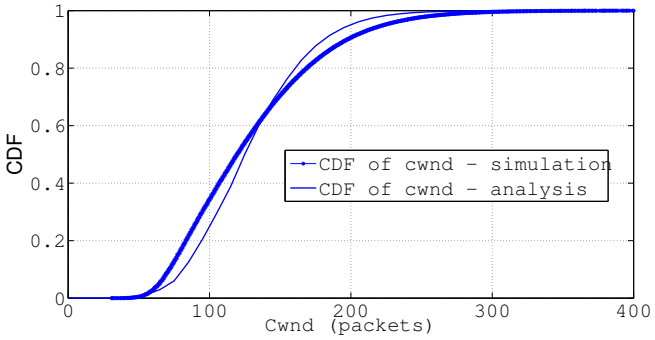

FIGURE 3.8: Congestion window - Intra-DC scenario - TCP Cubic

and repeatedly. Indeed, the pattern of Figure 2.1 that corresponds to the case of a single connection is observed on the mean value here. This is possible only if all connections experience losses simultaneously. A further confirmation of the synchronization of the TCP sources is that the average window decreases by a factor $(1 - \beta)$ after a loss, which is possible only if all sources loose packet simultaneously. We have checked in the

NS-2 simulations that all the 10 connections have indeed synchronized loss events. We tried to work around this issue by applying various techniques to avoid synchronization of sources. Especially, we tried to increase the buffer size or the level of multiplexing (number of active connections). However, the synchronization pattern of Figure 2.1 appears at each attempt. It seems to be a fundamental feature of TCP Cubic to exhibit this loss event synchronization as already observed by Hassayoun and Ros in [HR09]. In this paper, the authors studied several high speed version of TCP and observed, through simulation, the existence of synchronization among sources even when using several counter-measures like RED policy, traffic on the backward path or time-varying RTT. They also observed that while a lot of sources experience losses simultaneously, the utilization of the link remain close to the maximum. This is confirmed by our simulations and somewhat captured by our model. A last remark regarding Figure 3.9 relates to the frequency of oscillations that is higher with our model than in the simulations. In the fluid model, only a fraction of the sources experience losses when the buffer gets full. The other sources therefore keep on increasing their window, leading to losses at a higher rate than for simulations where the pressure on the buffer decreases for a long period of time when all sources simultaneously lose packets. A deeper analysis will be performed in the next chapter on the origin of synchronization in TCP Cubic.

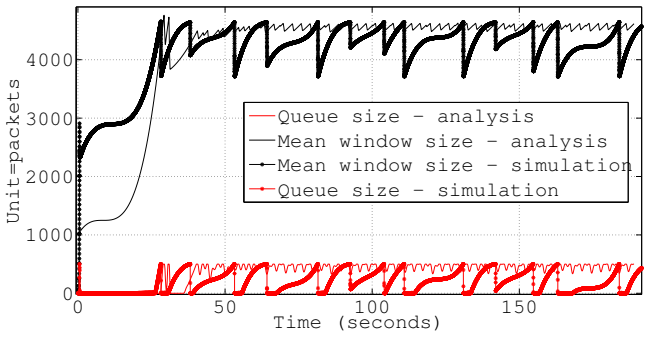

FIGURE 3.9: Time series of queue size and average window size - Inter-DC scenario - TCP Cubic

3.4.4 FTTH scenario with New Reno

For the case of New Reno, we obtained a good match between the simulations and the model. To present a metric different from the ones presented in the previous scenarios, we consider here the time series of goodput. As it can be seen from Figure 3.11, the frequency as well as the amplitude of the utilization time series match between the simulation and the model, once the stationary regime is reached.

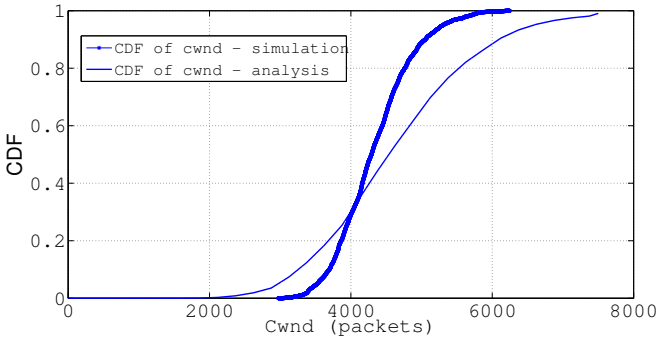

FIGURE 3.10: Congestion window - Inter-DC scenario - TCP Cubic

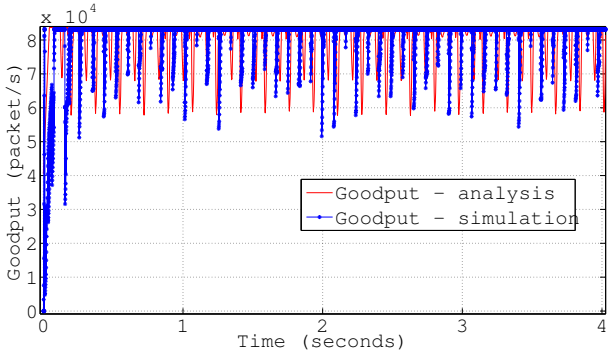

FIGURE 3.11: Time series of utilization - FTTH scenario - New Reno

3.5 Study of fairness and the impact of the buffer size

In this section, we use our fluid models to investigate two key problems. The first one is the fairness of TCP Cubic as compared to that of TCP New Reno. While TCP Cubic is able to take advantage of paths with larger BDPs, one can question its ability to share the bandwidth evenly between flows. We use New Reno as a reference, as it is known to achieve a good level of fairness when the flows share the same path.

The second issue that we investigate is the impact of the buffer size on the efficiency of TCP Cubic (and also New Reno). The question of buffer sizing has received a lot of attention, e.g., [WM05, HR09, CB07]. In [HR09] (respectively [CB07]), they advocate using a buffer size equal to 10% of BDP (respectively 20% of BDP). In [WM05], authors have a more complex rule of a buffer size equal to $0.63 \times C \times \overline{RTT}\sqrt{N}$. In our study, we investigate buffer sizes whose range is between 10% and 60% of the bandwidth delay product of the path. However, some measurements studies focusing specifically on TCP Cubic, lead to observing a detrimental effect of small buffer [LSM07]. However, the authors in [LSM07] pinpointed that the jury was still out concerning the root cause of

the inefficiency that they observed as it could be an intrinsic feature of TCP Cubic or an artifact of their testbed.

We restrict ourselves to the intra-DC and FTTH scenarios, as we obtained good match with simulations for those cases. While TCP Cubic operates in TCP mode in these scenarios, note that the algorithms that govern TCP Cubic in TCP mode and TCP New Reno are not the same.

3.5.1 Fairness analysis

Fairness relates to the ability of a mechanism to share the available resources among a set of competing tasks. However, fairness needs to be studied jointly with an efficiency metric. Indeed, consider the scheduler of a 1 Gb/s access link with two competing flows. If the scheduler attributes the full link capacity to one flow and nothing to the other, it is efficient (the link is fully utilized) but unfair. On the contrary, if each flow receives 1 Mb/s, the scheduler is fair but inefficient. We obviously want to have efficiency and fairness simultaneously.

For the case of TCP Cubic and New Reno, we assess the fairness of the protocol by two metrics. First, the distribution of the congestion window, which is computed by considering a large time interval when the model has reached equilibrium, gathering all samples and reporting its cdf. However, the marginal distribution of congestion window is not sufficient as one loses the notion of time when computing this metric. To explain that, let us consider the following toy example with two TCP flows and a 1 Gb/s link: at time t_1, each flow received 0.5 Gb/s, and time $t_1 + \delta$, each flow has 0.1 Gb/s. In this scenario, we have fairness, even though we are not efficient. Now consider the alternative case where at times t_1 and $t_1 + \delta$, one flow receives 0.5 Gb/s while the other receives 0.1 Gb/s. The marginal distribution of rates (and thus windows) is the same in both scenarios. However, the level of fairness is not the same. To capture the time variation of the distribution of congestion windows, we compute, at each time instant the coefficient of variation[1] (CoV) of the window size distribution and we report its cdf over a large time period.

We report in Figures 3.12 and 3.13 the marginal distribution of congestion window (we normalized the results of the second scenario by the mean window of the first one to ease presentation) and of the CoV over time of the congestion window distribution. It is clear that TCP Cubic achieves a better level of fairness than TCP New Reno over the two scenarios of interest, as (i) the marginal cdfs span over a smaller set of values for TCP Cubic and (ii) the CoVs for TCP Cubic are both smaller and span also over a smaller set of values.

[1]The CoV is the ratio of the standard deviation to the mean of a distribution. It can be seen as a normalized measure of its variability.

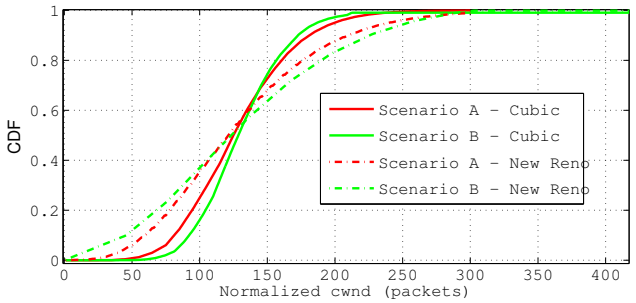

FIGURE 3.12: Congestion window - Intra-DC and FTTH - TCP Cubic and New Reno

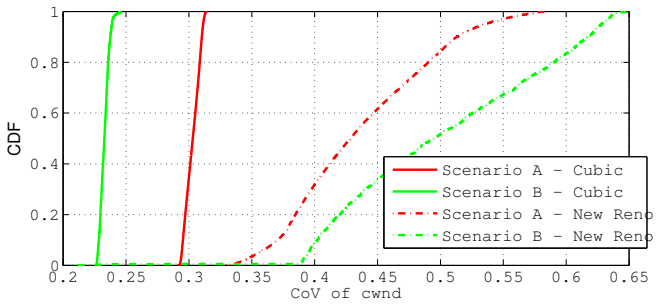

FIGURE 3.13: CoV of congestion window - Intra-DC and FTTH - TCP Cubic and New Reno

As stated earlier, fairness and efficiency have to be assessed jointly. We report in Figures 3.14 the distribution of utilization for the FTTH and intra-DC scenarios for both TCP Cubic and New Reno. We can now conclude that the better fairness of TCP Cubic is not achieved at the expense of a lower link utilization.

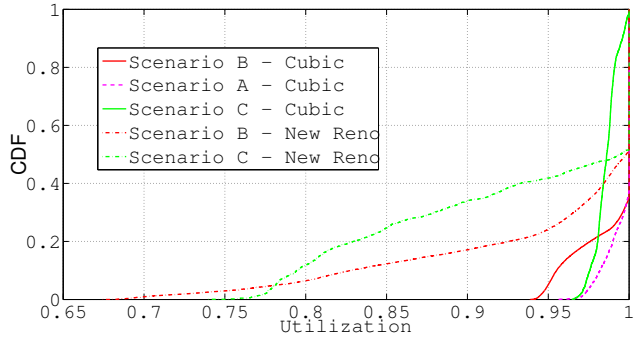

FIGURE 3.14: Utilization - Intra-DC and FTTH - TCP Cubic and New Reno

3.5.2 Impact of the buffer size

In this section, we investigate the impact of the buffer size on the utilization of the queue (and consequently of the server). We report results only for the intra-DC scenario. We vary the buffer size at the bottleneck from 10% of the BDP to 100% of the BPD for both TCP Cubic and New Reno - see Figures 3.15 and 3.16, where we present the normalized occupancy of the queue. Several conclusions can be drawn from these figures.

First, both TCP Cubic and New Reno are greedy in the sense that the larger the buffer size, the larger the queue occupancy. It is not necessarily a good news as larger queue occupancy means larger set-up latency for new incoming flows and larger jitter for time sensitive traffic, e.g., Web searches in a DC [VHV12]. Second, TCP Cubic is more greedy than New Reno. Third, TCP New Reno is clearly less efficient than TCP Cubic for buffer sizes smaller than 60% of the BDP as we observe a significant fraction of mass at zero, meaning that the buffer is often empty, hence the server is likely to be underutilized.

Overall, for the case of TCP Cubic, our model suggests that this version of TCP is able to survive with buffer sizes as small as 30% of the BDP. The experimental results obtained in [LSM07] are thus not pathological behaviors of TCP Cubic, but are likely to be due to another cause, e.g., a bad implementation (the author in [LSM07] used an early implementation of TCP Cubic in the Linux kernel). Note however that when the buffer size becomes very low, other technical problems might appear in real network appliances (such as competition between reading and writing into buffers). Hence, while the behavior observed in [LSM07] does not seem to be due to TCP Cubic itself, it is likely to be observed with other real experimental networks.

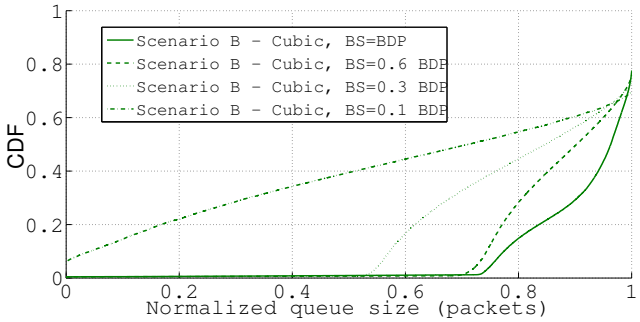

FIGURE 3.15: Impact of buffer size - Intra-DC - TCP Cubic

3.6 Conclusion

In this chapter, we have derived a fluid model for TCP Cubic, that allowed to predict the values of various metrics such as distribution of the window sizes of N connections,

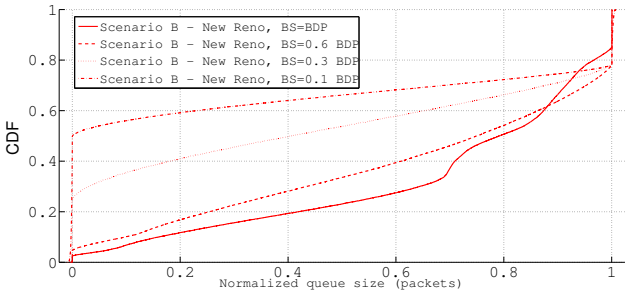

FIGURE 3.16: Impact of buffer size - Intra-DC - New Reno

throughput, RTT, loss rate and queue size. We proved a scaling property that allowed us to run the model for cloud networking scenarios of interest, without entailing a state explosion and hence a prohibitive computational cost. The model is validated against NS-2 simulations in these scenarios. We exhibit that the fit is very good for the intra-DC and FTTH scenarios, while it is less good for the inter-DC scenario. In this last mode, TCP Cubic operates in Cubic mode and causes loss synchronization amongst the connections. We further investigate this issue in Chapter 5 to understand if it comes from the simulator or can be observed in a real network.

We finally assess the fairness and buffer size impact for TCP Cubic and TCP New Reno. TCP Cubic is at once more efficient and fair than TCP New Reno, in particular in the case of low buffer sizes. Our results show that, in contrast to TCP New Reno, TCP Cubic is able to survive with buffer sizes as small as 30% of the BDP, thereby shedding some light on the possible cause of bad utilization observed in experimental works for such buffer sizes.

Chapter 4

Performance Analysis of Orange cloud solution: Cube

4.1 Introduction

In the previous chapter, we have analyzed the stability of TCP in scenarios that represent the environment where Cloud customers evolve (latency of 20ms and bandwidth equal to 100 Mbps, assuming a client with FTTH connection) and intra Data Center environments (latency of 1ms and bandwidth equal to 1Gbps and 10Gbps), by means of an analytical model employing the Mean Field theory [BW09] . Thanks to the Mean Field based model, we could study the performance of TCP Cubic using a fluid model that was based on mean-field approach. The results of the proposed analytical model have been validated by mean of the NS-2 network simulator.

We wanted to go one step further and we decided to study and compare the behavior of real TCP flows in a Data Center environments with the ones obtained with our model. We have then decided to carry out several experiments in real networks.

In this chapter, we will show in details the topology and characteristics of equipments used in our experiments. We will describe the challenges that we have faced in order to execute our campaign of experiments and the methodology used to understand the obtained results.

4.2 Cube Beta Infrastructure as a Service (IaaS)

We have studied TCP performance in real networks by means of two testbeds. The first testbed, called Cube, is an experimental network used by Orange Lab in order to test the new services provided by the France Telecom (FT) company. Since this testbed uses the real network infrastructure of FT, we do not have administrative rights on the

network equipments but only on the end hosts, which are Virtual Machines deployed over different servers where we do not have administrative rights also.

The purpose of this section is to present one of Orange clouds solution: Cube. Orange Business Service offer "Cube" as an IaaS (Infrastructure as a service). Cube is part of Montsouris datacenter. Four virtual machines were tested in Cube to transfer data to 4 others virtual machine in Sophia. These tests were conducted with various tools, in particular "Iperf" (https://iperf.fr) and perl scripts which we devised specifically for either read data from the disc, or generate data in memory.

4.2.1 Cube environment

The objective of the Cube program [Gro13] is to build an internal private cloud IaaS capability for Orange group entities, using technologies and processes consistent with the Orange Business Services offer to the external market. The service will be delivered from selected tier 1 group data centers, operated by internal IT operations teams and accessible to all group entities via the Group Intranet Network (GIN).

4.2.1.1 Sandbox Environment

The Cube Beta offer is an on-demand or personal IaaS service providing hands on experience to application architects, developers and infrastructure teams. Subscribers can request virtual machines from a catalog and use it for experimentation in developing and deploying applications in a cloud environment. The objective is to enable greater understanding of how cloud technologies and services can be across the group to transform Orange IT supply chain and to validate assumptions on potential transformation scenarios.

The Sandbox service is accessed via a simple self-service portal called E2C developed within Orange Labs Networks and Carriers OLNC. The standard VMware application programming interfaces (APIs), is also available for use in custom projects.

Two data centers host the service, Lodz (Poland) and Montsouris (France) and with an initial capacity of approximately two hundred virtual machines at each center.

The service is open to all group affiliates via the Group Intranet Network (GIN). As this is a prototype, a single "zone" or organization is created in each center shared by all users.

4.2.1.2 Service Description

Services that a cloud subscriber (user) can consume are commonly referred to as a service catalog, where a set of pre defined options can be selected via a web-based portal (subject to appropriate authorization, and initial set up), and will be provisioned in the majority of case automatically. The subscriber can then create infrastructure

environments or projects which can be used to support application development and test production.

Each subscriber can create a cloud project, which can have a maximum of 10 virtual machines from a given list. A selection of standard (x86, Windows and Linux) images from the common bundle catalog is available to configure the machines. Users can request to upload custom images which will be assessed on a case by case basis.

Sandbox service users access the service via a portal called E2C. This portal leverages vCloud API and E2C Engine API for a restricted set of functionalities. The portal presents a tab called "catalog" where users can find, browse, and deploy templates from the catalog. Those templates represent containers provisioned with one VM, called vApps.

Sandbox Service users can deploy and manage Load Balancers in their Cloud Projects. Users can select the VMs to be load balanced and on which port. The load balancer can have three behaviors: load balance TCP, HTTP or HTTP with session tracking.

4.2.2 Orange Group Architecture

Figure 4.1 shows the relationship between the platforms and the GIN network.

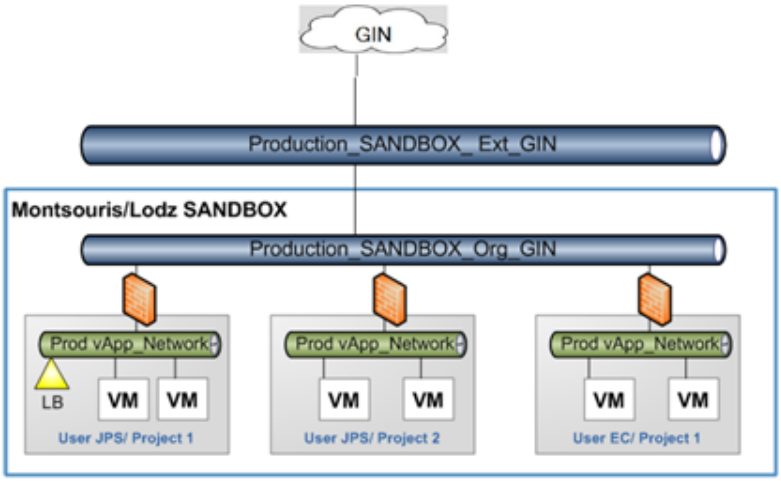

FIGURE 4.1: GIN Network and Platforms Relationships

Figure 4.2 shows the connection between the Cube platforms, the GIN, and the countries. Countries need to configure local firewall access to the Cube services.

In Figure 4.3 we report all machines we used in testing. We created four virtual machines in Cube Montsouris (two Linux machines and two Windows machines). On Sophia side, we have four test machines, too.

Outgoing traffic for Sophia to the outside are allowed only through one of the following ports:

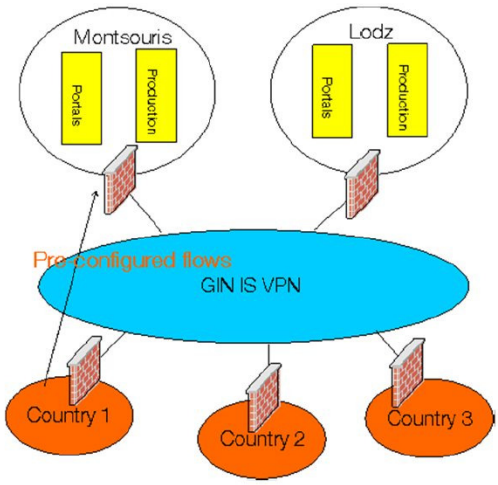

FIGURE 4.2: Cube Platforms, GIN & Countries Connections

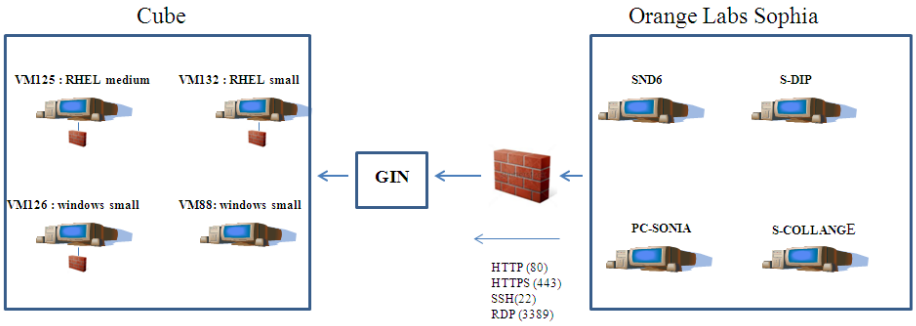

FIGURE 4.3: Sophia/Cube machines

TABLE 4.1: Characteristics of the Cube machines

	Power	Memory	Storage	tcp congestion control protocol
VMwindows-1 (small)	1vCPU	2GB	32 GB	Compound TCP
VMwindows-2 (small)	1vCPU	2GB	32 GB	Compound TCP
VMlinux-132 (small)	1vCPU	2GB	20 GB	TCP Cubic
VMlinux-125 (medium)	2vCPU	4GB	25 GB	Bic TCP

- HTTP (80)
- HTTPs (443)
- SSH (22)
- RDP (3389)

Characteristics of Cube machines are given in table 4.1.

4.3 Disk Benchmarking

4.3.1 Motivation

In this section, we present results obtained with benchmarking tools for disk read and write operations. The starting point of this analysis was a problem observed during traffic captures on some Cube instances. We indeed observed large periods of silence (no packet captured) in the tcpdump files when dumping traffic on our Cube machines. In addition, tcpdump reported a number of "packets dropped by kernel" events. The latter type of events might be to a too loaded CPU or a too slow disk. As the CPU of the instances were apparently not excessively used, we decided to further investigate in details the read and write performance of our Cube instances.

Benchmarking disk performance requires an a priori idea of the type of I/O workload generated by the application we want to use. Fortunately, in our case the I/O workload we want to test is a simple one as the tcpdump process is writing all frames of all TCP transfers in a single file.

Before talking further about disk performance, we can note that disk I/O performance is not the only suspect in our investigation. In particular the transport layer is a possible suspect as TCP might decide to stop sending for a period of time. Also, the vitualization layer, VMware in the case of the Cube, might be responsible for the capture problem we face. Virtualization might be responsible either directly if the system administrator can cap the access to resources (disk, memory, CPU) of the instances or indirectly if the load imposed on the disk by all the VMs on the physical server is too high. Last but not least, the hardware itself can the root cause behind I/O performance. It turned out, as we will see later in this section, that the blame can not be put on the transport layer, but on the virtualization layer itself. However, we deem that the study of disk I/O performance is a worth investigating problem per se and we decided to address the following questions:

- What is the actual difference between raw and legacy (through the file system) operations?
- What is the pure impact of the hypervisor, i.e., when a single VM is accessing a local disk?
- What is the impact of the request size? Indeed, to write 128 MB of data, a process can issue 10^3 requests of 128 KB or 128 requests of 1 MB.

The rest of our study on the I/O performance in a cloud environment is a follows. First, we recall the different software (file system, scheduler) and hardware components (mainly controller) involved in I/O operations. We also present the state of the art tools

that we use, namely dd and hdparm. Next, we investigate the I/O performance of the hosts involved in the TCP transfers we performed in Cube. Last, we report results of application of dd and hdparm in a small tested available at I3S, where a server can be booted either under a native CentOS operating system or under VMware or Xen. The flexibility offered by this testbed enables to study the minimal impact of virtualization as we test the same hardware with or without virtualization.

4.3.2 Reading and Writing from disks and through file systems

I/O devices are by far the slowest memory components of a computer. To compensate for the speed discrepancy with the CPU and RAM, a number of strategies are put in place to improve performance, which results in delaying as much as possible disk.

- Components involved in optimizing write operations:
  - Caches of the file systems are the primary components involved in improving, from the application viewpoint.
  - The scheduler of I/O requests then seeks to aggregate requests as much as possible, which it does by delaying the sending of write operations to the disk controller.
  - The disk controller seeks to minimize the moves of the disk heads (for magnetic disks) by appropriately shuffling the order in which requests are submitted to the disk itself.
- Components involved in optimizing read operations:
  - The is mostly the operating system that seeks to optimize read performance by asking more data than was actually requested by the application. It is based on the assumption of data locality, i.e., that an application which asked to read a block of a file is likely to read the following blocks in the near future. The effectiveness of this strategy depends on the exact behavior of the application.

dd and hdparm

hdparm [Ubub] is a performance and benchmarking tool for hard disks. It is primarily used to tune and optimize disk parameters, but also has a switch to use it as a simple benchmark tool. hdparm can perform two benchmarks:

- The speed of reading through the buffer cache to the disk without any prior caching of data. (Timing buffered disk reads, -t option).

- The speed of reading directly from the Linux buffer cache without disk access. (Timing cached reads, -T option).

The first option gives an indication of the throughput of the processor, cache, and memory of the system under test. The second one measures how fast the drive can sustain sequential data reads, without any filesystem overhead. It is also usual to run these tests couple of times to get accurate results. The command to use is:

hdparm -Tt /dev/sda

The second tool used to assess disk and memory performance is dd [Ubua] [jlo08] [Rom10], it is a command on Unix and Unix-like operating systems whose primary purpose is to convert and copy a file. On Unix, device drivers for hardware (such as hard disks) and special device files (such as /dev/zero and /dev/random) appear in the file system just like normal files; dd can also read from (and in some cases write to) these files.

As a result, dd can be used for tasks such as backing up the boot sector of a hard drive, and obtaining fixed amount of random data. The dd program can also perform conversions on the data as it is copied, including byte order swapping and conversion to and from the ASCII and EBCDIC text encoding.

Read/write tests can be performed on Raw disk and also a Filesystem. For example the writing test on a filesystem is done by creating a "woueb" file consisting of 256,000 4KB blocks:

time sh -c "dd if=/dev/zero of=/home/woueb bs=4096 count=256000 conv=fdatasync && sync"

Where :

- bs : is the block size, it is a unit measuring the number of bytes that are read, written, or converted at one time
- count : is the number of blocks to be copied.
- conv=fdatasync : it tells dd to require a complete "sync" once, right before it exits. So it commits the whole data, then tells the operating system: "OK, now ensure this is completely on disk", only then measures the total time it took to do all that and calculates the benchmark result. It avoid being fooled by the OS that would cache the data before pushing them to the disk.

For the reading test from the filesystem, it is the opposite: we read "woueb" file that was previously created:

time sh -c "dd if=/home/woueb of=/dev/zero bs=4096 count=256000 && sync"

To make dd tests against a raw device instead of a filesystem, we use the following command line:

dd if=/dev/sda of=/dev/zero bs=4096 count=256000

4.3.3 Results on Cube experiments

4.3.3.1 I/O requirements: back of the envelope computation

In this section, we compute a trivial upper bound on the I/O requirement of the tcpdump process we run in Cube. As we will see later, the network path between the Cube datacenter in which we performed experiments and the Sophia Orange Lab is not managed in a symmetric manner. There is apparently no restrictions in the up direction, from Sophia to Cube, with apparently 300 Mb/s. In contrast, in the down direction, we are limited to around 3 to 4 Mb/s per connection. The total capacity is however apparently symmetric. We thus assume that our tcpdump process has to write on the disk at a speed of $\frac{300}{8}\frac{96}{1500} = 2.4$ MB/s as we capture only the first 96 bytes of the full MSS packets sent over the network interface[1]. In contrast, if tcpdump has to write on the disk for a network speed around 3Mb/s, it needs a modest 2.4KB/s.

We report in Figure 4.4 the average writing speed over several experiments made with tcpdump. To compute this speed, we divided the size of the file by the difference of the first and the last timestamps of the captured frames. Note that these values are upper bounds as the operating systems might tell the tcpdump process that the writing is completed while the write operation is still in progress but absorbed by the writing buffers.

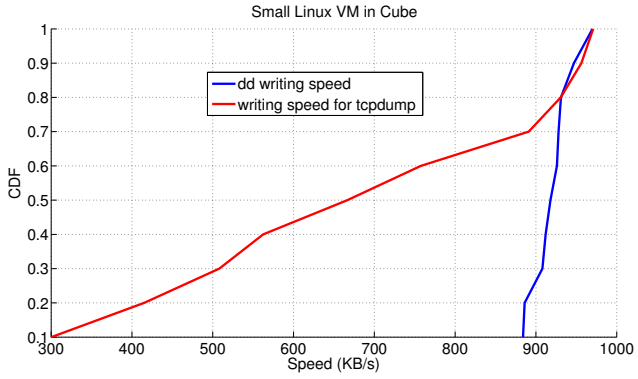

FIGURE 4.4: Writing speed, dd-tcpdump, small Linux VM in Cube

[1]We discovered during our experiments that the default snaplen value of TCP, i.e., the default maximum size of a captured was not 96 but 1500 bytes in some OS.

4.3.3.2 Cube instances and SND6 performance

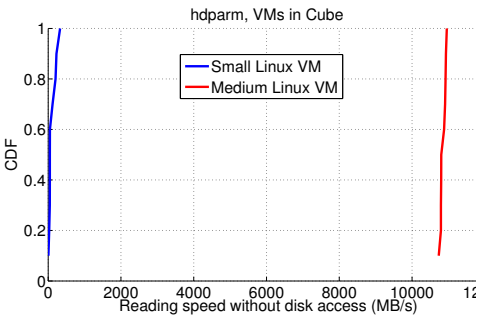

FIGURE 4.5: Reading speed without disk access (MB/s), Linux VMs in Cube

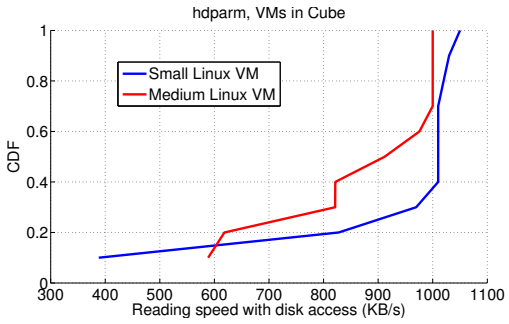

FIGURE 4.6: Reading speed with disk access(KB/s), Linux VMs in Cube

We performed several hdparm tests for both Linux VMs in Cube. Figure 4.5 represents the CDFs of the reading speed without disk access, for different tests made on both Linux VMs. This figure represents the results of the hdparm option "-T". The blue curve represents the results for the small Linux VM. The red curve is for tests performed on the medium Linux VM. Each time, 10 tests were carried out, then we plot the CDFs to see the variability of results.

When comparing the results of both VMs, we find that medium Linux VM performance is far better than those obtained by the small Linux VM when reading directly from the buffer cache. Its is reasonable to believe that this result is dependent on the hardware (CPU, memory) of the physical server of this medium VM that must be more powerful.

Results of the "-t" hdparm option are presented in Figure 4.6, where we plotted the CDFs for reading speed with disk access. We note also that performance is similar on both instances and not very high. This suggest a limitation imposed by the hypervisor and the way disks are exposed to the virtual machines.

In our early tests, we did not consider the "conv = fdatasync" option. We report the speed results in Figure 4.7. Note that in this case the speed was of the order of Gb/s, such speed values correspond to memory access speed and not disk speed. So, it was necessary to add the "conv = fdatasync" option, and also to empty the read/write cache each time to avoid measuring the speed of the memory, using the command line : *echo 3⟩ /proc/sys/vm/drop_caches*

The results presented in the rest of this section make use of this option to measure performance related to the disk and not to the operating system.

We report in Figure 4.8 the CDFs for reading/writing speeds results from and to a filesystem, and also the reading speed from the raw device, performed on the small Linux VM. We also plotted on the same figure, the hdparm results with disk access in order to compare them with the dd reading speeds from the raw device. Note that the four

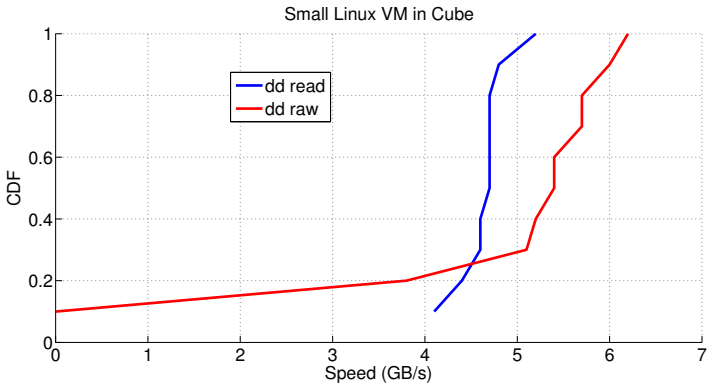

FIGURE 4.7: Disk performance using dd without conv option, small Linux VM in Cube, Reading speed (GB/s)

curves possess the same speed unit KB/s. We observe that dd results are very closer on both filesystem and raw device even though we would have normally expect that raw results should be better than read through the filesystem. Additionally, hdparm results look fairly consistent with the raw device results.

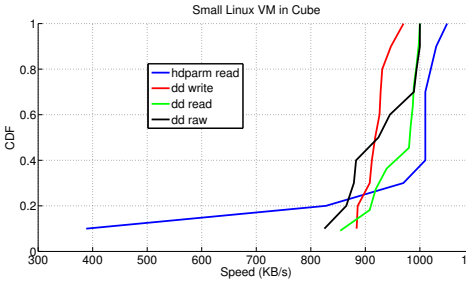

FIGURE 4.8: Disk performance using dd with conv option, small Linux VMs in Cube

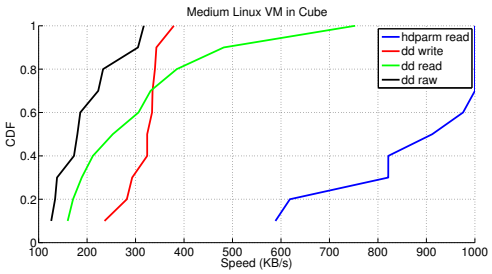

FIGURE 4.9: Disk performance using dd with conv option, medium Linux VMs in Cube

We report in Figure 4.9 dd and hdparm results for the medium Linux VM. We can notice that the disk performance given by dd, on both filesystem and raw device, decrease for the medium VM compared to the small one, were values vary between 100 and 500KB/s rather than 800 to 1000KB/s. But hdparm results still closer to small Linux VM results. In fact, this difference among hdparm and dd results may be caused by the load of physical servers that host both VMs.

We further performed hdparm tests on one Windows VMs in Cube. We report results with and without disk access in Figures 4.11 and 4.10, respectively. We observe through these results that the reading tests for the Windows VM are higher in the morning, which may be caused by the fact that the physical server of this VM is more loaded in the afternoon. We observe also, that the reading speed values given by the hdparm tool have the same range and are similar to Linux VMs results with disk access.

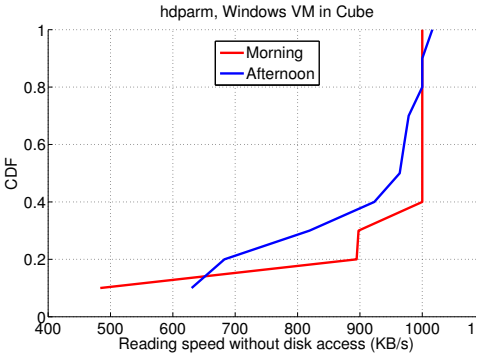

FIGURE 4.10: hdparm, Reading speed without disk access (KB/s), Windows VM in Cube

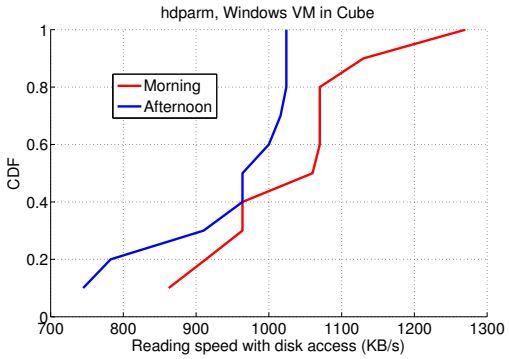

FIGURE 4.11: hdparm, Reading speed with disk access (KB/s), Windows VM in Cube

Intermediate conclusion: the above results of disk benchmarking are difficult to interpret as we are not expert in disk benchmarking. Let us first recall that the reason why we did these tests was to find the root cause of the silence periods in the tcpdump traces during the network transfers. We can still draw general conclusions:

- the disk performance is apparently low.
- it depends on the type of VM we asked for but with no logic, e.g., small instance can have better performance than intermediate instances. This might be due the fact that these VMs we're running in different physical servers and apparently the physical server for the small instance is newer than the one of the larger instance.
- results depend on the time of the day. This was more to be expected in a cloud context.

To move forward, we decided to:

1. perform the same tests with the machine used in Sophia that feature different hardware and are shared among people.
2. perform tests in a controlled environment with servers that could be virtualized or run in native mode. These servers are fully under our control and should enable to assess the impact of virtualization on observed disk performance.

Performance of machines at Sophia

We made some tests using hdparm tool to evaluate the 2 local machines in Sophia, SND6 and SDIP. These machines will be used to carry out tests in both directions i.e., from Sophia to Cube and from Cube to Sophia. We report hdparm results in Figures 4.12 and 4.13.

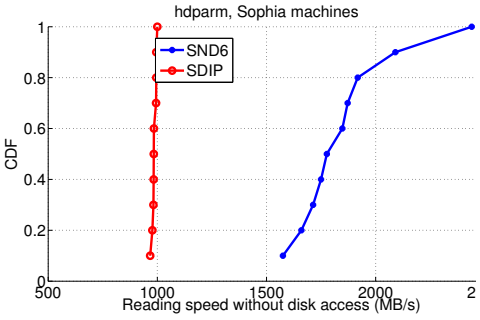

FIGURE 4.12: Reading speed without disk access (MB/s), SND6/SDIP

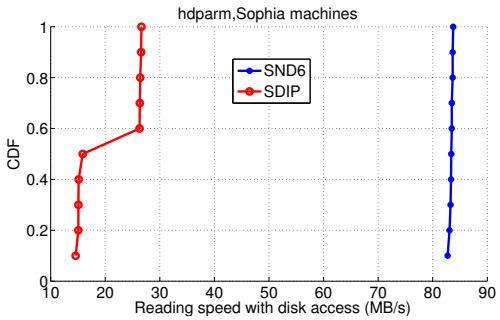

FIGURE 4.13: Reading speed with disk access (MB/s), SND6/SDIP

From the results presented in Figures 4.12, it is clear that SND6 performance is better compared to SDIP performance. We observe a correlation between performance and physical hardware characteristic as SND6 is newer than SDIP. Indeed, it is known that newer machines tend to have simultaneously better hard driver as well as CPU and memory chips and buses.

We can conclude from these tests that the performance of the machines in Sophia are clearly higher than the ones of the VMs in Cube. We however still observe performance variation that we attribute to the fact that the machines are shared among many users. This is why, in the next section, we move to the study of machines in isolation with or without virtualization in place.

4.3.4 Results obtained with the I3S testbed

In order to evaluate the impact of virtualization on disk performance, we performed a series of measurements using 2 Linux machines in I3S, Vsignet1 and Vsignet2. Vsignet1 is a physical server, booted alternatively from different hard drives containing the three operating systems analyzed : a native (non virtualized) CentOS, VMware and Xen. The 2 DELL servers vsignet1 and vsignet2 have the following characteristics:

- Model: Dell PowerEdge R410 and R510
- Processors: 8 cores - 2 * quad-core Intel Xeon processor 5600 series
- System speed: 2.13GHz, Bus speed: 4.80GT/s
- Memory: 12GB of DDR3 1067MHz
- RAID Controller: PERC H200 (6Gb/s)
- Network interfaces: Embedded Dual-port Broadcom NetXtreme II 5716

4.3.4.1 Native performance

We report in Figures 4.14 and 4.15 the disk performance obtained by dd and hdparm tools for both I3S machines. Results show that both machines possess stable and similar performance : Writing speeds of around 80MB/s, reading speed from the filesystem of about 100MB/s. We can observe also that the reading speed results from the raw device are more consistent with hdparm results.

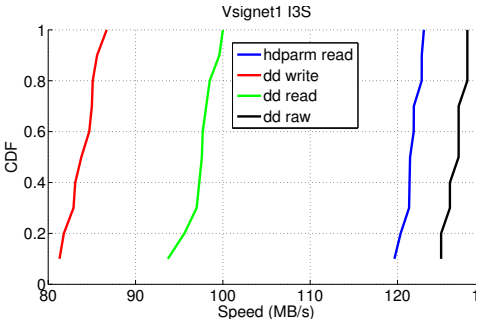

FIGURE 4.14: Reading/Writing Speed (MB/s), Vsignet1 machine

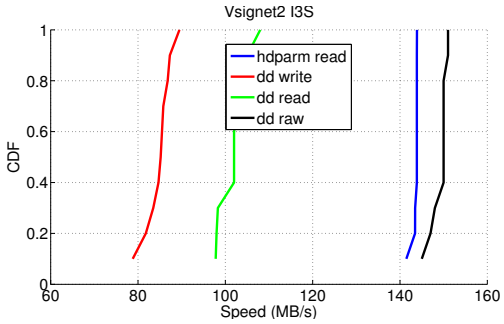

FIGURE 4.15: Reading/Writing Speed (MB/s), Vsignet2 machine

4.3.4.2 Performance under Xen and VMware

We want to study the behavior of hdparm and dd when it comes to virtual machines. We consider the 2 virtualized environments Xen and VMware, and we report the results in Figures 4.16, 4.17, 4.18 and 4.19. We observe that in a virtualized environment, dd and hdparm tools speeds achieve better performance compared to the native case.

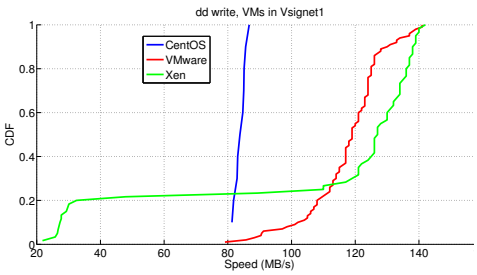

FIGURE 4.16: dd write (MB/s), the three OS

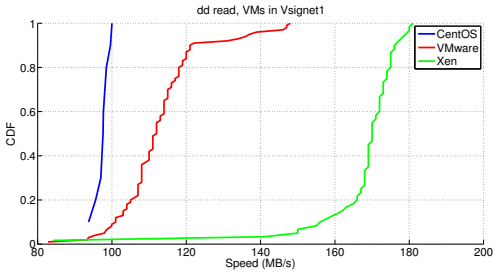

FIGURE 4.17: dd read (MB/s), the three OS

We believe that the root cause behind this observation is that dd and hdparm do not interact directly with the actual disk controller which is under the control of the hypervisor, but with the hypervisor itself through what is called the front end driver. It is likely that the front end reports that a write operation as completed before it is actually completed by the real driver. Hence the overestimation of performance that we observe.

This holds for disk write but does not hold for disk read. Wet further observe different performance between Xen and VMware. It seems that Xen and VMware do not use the same driver or,more likely, have a different way of parametrizing it.

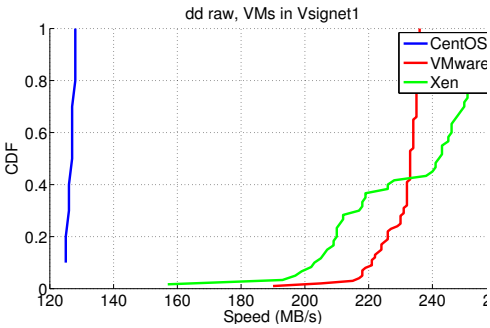

FIGURE 4.18: dd raw (MB/s), the three OS

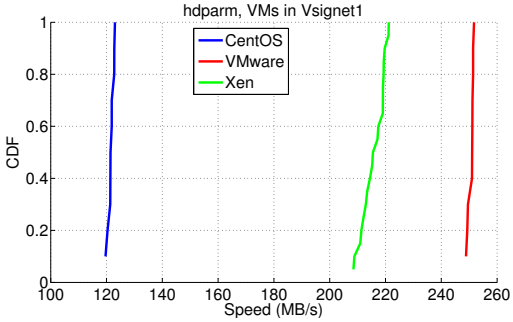

FIGURE 4.19: hdparm with disk access(MB/s), the three OS

4.3.4.3 Impact of block size

References on the dd command say that for some uses of the dd command, block size may have an effect on performance. For example, when recovering data from a hard disk, a small block size will generally cause the most bytes to be recovered. Issuing many small reads is an overhead and may be non-beneficial to execution performance.

For greater speed during copy operations, a larger block size may be used. When dd is used for network transfers, the block size may have also an impact on packet size, depending on the network protocol used.

So as to evaluate the impact of the block size on performance, we performed various tests with different block sizes on Vsignet2. For the various tests, the file size is always equal to 134MB. We then report the different curves obtained by dd, the write/read from a filesystem and also from the raw device in Figures 4.20, 4.21 and 4.22, respectively.

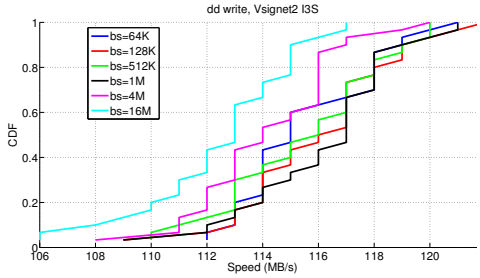

FIGURE 4.20: Disk performance using dd, different block sizes, writing speed (MB/s), Vsignet

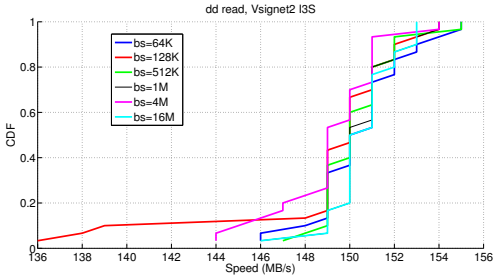

FIGURE 4.21: Disk performance using dd, different block sizes, reading speed (MB/s), Vsignet2

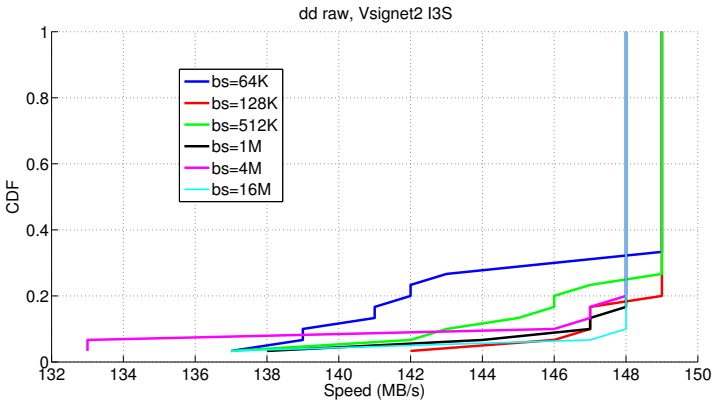

FIGURE 4.22: Disk performance using dd, reading speed from the raw device, Vsignet2

From the results of writing/reading from a filesystem and the raw device, we observe that varying the block size introduced a slight variation in the speeds values on the order of a few MB/s.

Then, in order to check the previous assumptions for the block size effect on a virtualized environment, we reboot Vsignet1 on VMware and we performed a set of tests on the same virtual machine with 128MB file size, with 2 block sizes 100MB and 64KB.

When comparing the results of Figure 4.23 with the native case in Figure 4.14, we find that the read/write speeds from/to a filesystem are comparable with the native case. While hdparm and raw device results are almost doubled, which may be due to virtualization. We can observe also that the results with two block sizes provide correlated performances, so the block size parameter does not introduce serious performance degradation's as it was mentioned in the literature. Thanks to hdparm and dd tools

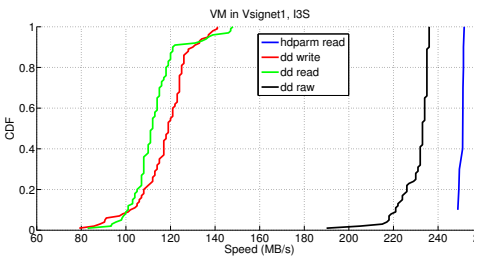

FIGURE 4.23: Disk performance (MB/s), one VM in Vsignet1, 128 1M blocks

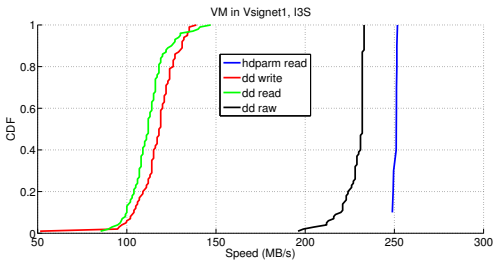

FIGURE 4.24: Disk performance (MB/s), one VM in Vsignet1, 2000 64KB blocks

, we have had an idea about the performances of VMs in Cube, machines in Sophia, and machines in I3S. These results show that the disk performance on a virtualization environment depend on several factors :

- virtualization configuration

- the server load
- physical Hardware ...

By comparing the results of two VMs, we find that the read/write speeds for the small Linux VM are higher than those for the medium VM. So, to make tests from Cube to Sophia, we chose the small VM as it is more powerful than the medium one, and also because it possesses TCP Cubic as congestion control protocol.

The tests we did in I3S suggest that virtualization does not degrade performance in terms of disk access. Hence the bad performance observed with Cube is a configuration problem and the blame can not be put directly on the virtualization layers to be fair, we have to mention that Cube was a first prototype inside Orange. It has now been upgrading. However we did not have the possibility to redo experiments on this new IaaS service due to lack of time.

Regarding tests from Sophia to Cube, SND6 machine seems to be more powerful, so it is better suited for tests from Sophia to Cube.

4.4 Network Benchmarking

In this section, we report on the network measurements performed from Cube machines. The scenario we focus on is a simple one: we generate traffic from or to one Cube machine to one host located in Orange Labs in Sophia. The network connection between the two sites is provided by the Group Intranet Network (GIN). As we will see, the path is not managed in a symmetric manner: a single connection can obtain far more bandwidth in the up (outside to Cube) than down (Cube to outside) direction. Hence, we will separate in this section the analysis of the two directions.

We present hereafter the tools we used, namely Iperf, perl-file and perl-data, followed by our methodology. Next, we report the measurements in the Sophia to Cube and then on the Cube to Sophia direction. We use this specific ordering as the firewalls setup in the direction Sophia to Cube direction enables to use Iperf natively (without any SSH tunnel) as well as perl-file and perl-data, which allows to calibrate the tools against each other. In the reverse direction, we resort on comparing the performance of our tools with Iperf in a SSH tunnel.

4.4.1 Description of the tools: Iperf, Perl-data and Perl-file, and the trace analysis tools

4.4.1.1 Iperf

Iperf [fANR] is a commonly used network testing tool that can create TCP and UDP data streams and measure the throughput of a network that is carrying them. It is a

tool for network performance measurement written in C.

Iperf was developed by the Distributed Applications Support Team (DAST) at the National Laboratory for Applied Network Research (NLANR), a research lab that merged with the University of California San Diego's CAIDA group, but which was shut down on December 31, 2006.

This tool allows the user to set various parameters that can be used for testing a network, or alternatively for optimizing or tuning a network. Iperf has a client and server functionality, and can measure the throughput between the two ends, either unidirectionally or bi-directionally. It is an open source software and runs on various platforms including Linux, Unix and Windows.

Iperf options that we use during the tests are :

- -i : Sets the interval time in seconds between periodic bandwidth, jitter, and loss reports
- -l : The length of buffers to read or write
- -s : Run Iperf in server mode
- -c : Run Iperf in client mode, connecting to an Iperf server running on host
- -t : The time in seconds to transmit for. Default is 10 seconds
- -p : The server port for the server to listen on and the client to connect to
- -w : Sets the socket buffer sizes to the specified value. For TCP, this sets the TCP window size

Example : iperf -c 10.114.7.132 -p 80 -t 100 -i 1 -l 10K

4.4.1.2 Perl Socket

We created the following Perl [Wal87] files :

- *serv_file.pl* and *client_file.pl* are files to execute when sending a file.
- *serv_data.pl* and *client_data.pl* are files to transfer data generated in memory .

For reasons of port management, communications are tolerated in the direction Sophia outward, so we set up the server on VM132 in Cube. Iperf does not offer this feature, which is why we devised thoses tools. Once the socket [Uni80] is created, the transfer can be done in one of the two directions. So, it remains to indicate at the client side the direction of data transfer. The principle of sending data generated in the memory is as follows: we created a string of fixed size (16K bytes) at the VM132 in Cube, and we keep sending the data for 100 seconds. Whereas to transfer a file, after generating

the file using the dd command, we send (in the direction of transfer it into blocks of 16K-byte size).

We use in conjunction with these perl files a packet capture tool tcpdump to determine the TCP statistics (stats and distributions). Also, we changed the setting *tcp_no_metrics_save* = 1. This prevent TCP to remember some characteristics of the last connection. Therefore the results of different successive tests will be independent.

4.4.1.3 Sniffing/analyzing tools

To capture and analyze the traffic performance, it was necessary to use software tools. We used various public tools and an internal tool developed at Orange Labs.

- Dipcp : DIP (Datawarehouse IP) is a traffic analysis tool developed at Orange Labs. It is a tool that allows traffic analysis following two modes:
  - A real-time mode: DIP reads a snapshot of IP segments from one interface Ethernet and calculated performance indicators.
  - A stored-capture mode : analysis method of stored capture. This is the mode that we used.

  Example of the used command :

  dipcp -C hm -T hum -S up=eth:00:14:5e:19:36:24 -g 0 -r File1.cap

  After applying the "dipcp" command to a capture file, a new file ".Csv" is generated. This file contains the computed statistical indicators.

- Tcpdump : Tcpdump [Mar10] is a command-line packet analyzer. It allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. Tcpdump is free software and works on most Unix-like operating systems. It uses the libpcap library to capture packets.

- Tcptrace : Tcptrace [Ost94] is a tool written by Shawn Ostermann, for analysis of TCP dump files. It can take as input the files produced by several popular packet-capture programs, including tcpdump, snoop, etherpeek, HP Net Metrix, and WinDump. This tool can produce several different types of output containing information on each connection seen, such as elapsed time, bytes and segments sent and received, retransmissions, round trip times, window advertisements, throughput, and more. It can also produce a number of graphs for further analysis.

- Wireshark : Wireshark [Coma] is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol

development, and education. Originally named Ethereal, in May 2006 the project was renamed Wireshark due to trademark issues.

4.4.2 Methodology

Cube being a cloud solution that relies on virtualization, a challenge we face when measuring the network performance is to understand the impact of the path as compared to the impact of the virtualization and especially the sharing of the server resources. Note that with the current setup of Cube, we have no direct way to assess the activity of the other VMs sharing the server. We hence apply a simple strategy to assess the stability and representativity of our data:

- We perform and compare measurements at different times of the day; we perform 10 separate measures.
- We analyze jointly the time series of the transfers and the corresponding CDFs for the metrics of interest. The two are mandatory as while CDFs offer a compact representation of data, they obscure the time behavior. Hence, our joint analysis.

The metrics we focus on are; the throughput, the congestion window of TCP and the RTT measured by TCP.

4.4.3 Sophia to Cube

We dump traffic on the sender side, which, in the case of this direction, is the machine in Sophia. We have no capture problem, i.e., silence periods in the tcpdump. Hence, we can focus on network measurement only.

Initially, before evaluating the behavior of each tool and its benefits, we first calibrate Iperf by selecting the most appropriate parameters for the read/write buffer and the socket size.

4.4.3.1 Calibration of Iperf

- Read/write buffers length "-l"

The first parameter that we tested is "-l", which represents the number of bytes that Iperf uses when read/write, the default value is equal to 8K bytes. We vary this parameter between 8 and 32K bytes. Different tests have been performed on the night from Sophia to Cube, while there is less traffic in those schedules.

We report in Figure 4.25 the 3 CDFs of throughputs for the 3 buffer sizes, 8, 16 and 32K bytes. We observe a high similarity between results when the buffers are equal to 8 and 16K bytes, but they are different from tests with 32K bytes. We note also that increasing the buffer size to 32K bytes decreases throughput. So we set the buffer size to 16K bytes for the remaining tests.

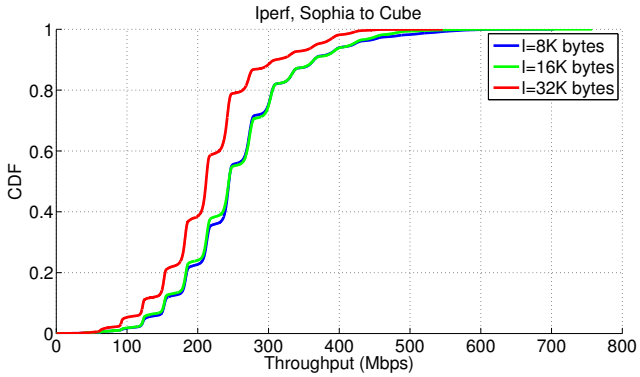

FIGURE 4.25: CDF Throughput(Mbps), Iperf l=8, 16, 32K bytes

- TCP window size "-w"

The second parameter to test is the "-w" parameter, which gives the TCP window. For this case, we have considered the default value of the "-l" parameter, i.e.,8K bytes. We performed several Iperf tests from Sophia to Cube with "-w" equal to 40K bytes on both sides. In Figure 4.26, we report the time evolution of the congestion window, we can observe that the window values exceed 40K bytes, and reaches up to 70 K bytes.

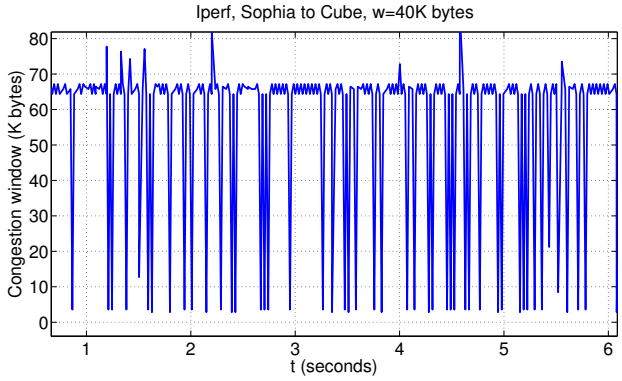

FIGURE 4.26: Congestion window, Iperf Sophia to Cube, w = 40 k bytes

Clearly, the "-w" option does not fill its purpose. It is better to change the Operating System parameters to adjust TCP windows rather than making it through Iperf. So, for the rest of Iperf tests, we will no longer consider this parameter.

4.4.3.2 Measurement results

One should run several tests for an accurate evaluation of the TCP metrics, we observe that there is consistency between the various tests. With the read/write buffer size is set to 16K bytes, we report the results for the congestion windows, throughputs and RTTs.

- Congestion window

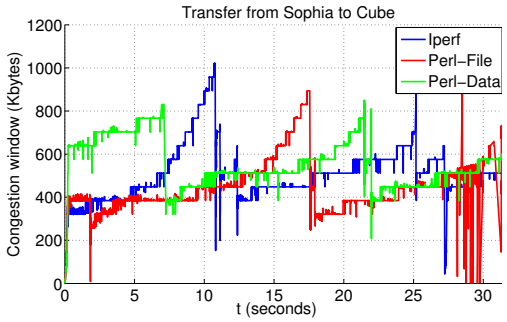

FIGURE 4.27: Congestion window (Kbytes), from Sophia to Cube, Iperf, Perl-File/Data

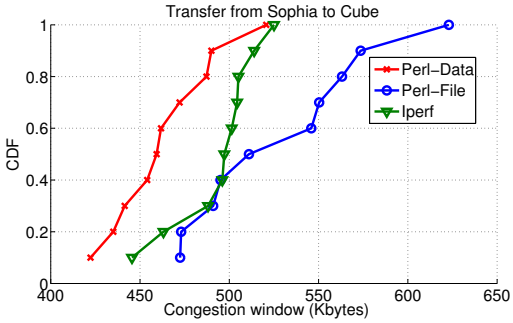

FIGURE 4.28: CDFs of congestion window from Sophia to Cube, Iperf, Perl-File/Data

We used 3 methods to send data from Sophia to Cube:

- Iperf tool.

- Perl-File: it is used to transfer data by reading a file on disk. We created a file of size 600M bytes using the dd command. Then we sent it into blocks of size 16K bytes, with the same read/write buffers size of Iperf.

- Perl-Data : data is generated in memory in the form of a character string of size equal to 16 K bytes (as the case of Iperf). Then this string is sent continuously for a period of 100 seconds.

We present in Figure 4.27 the time evolution of the congestion window for the 3 transfer methods, Iperf, Perl-File and Perl-Data. We provide in this figure a zoom on a specific moment in time of one simulation per transfer method that we performed. We observe that all curves have similar behavior, and the congestion windows vary between 200 and 1000 Kbytes.

To compare the various transfer types, we drew the CDFs of the average congestion windows resulting from all 10 tests, for each type of transfer (Iperf, Perl-File, Perl-Data). There are three curves on Figure 4.28 for the three transfer types. Each curve consists of 10 points, which are the values of the average congestion windows of 10 tests.

Iperf and Perl-Data CDFs are similar while the CDF of Perl-file reaches higher congestion window values than the other two cases.

- Throughput

Figure 4.29 presents the time evolution for throughputs of 3 the transfer methods, where the rates vary between 50Mbps and 450Mbps.

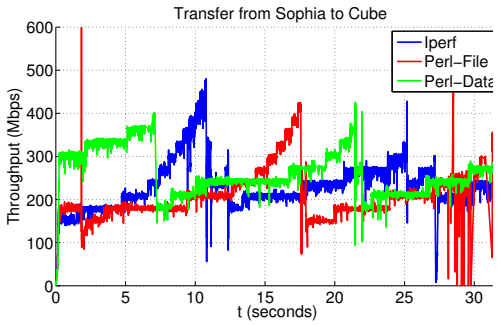

FIGURE 4.29: Throughput(Mbps), from Sophia to Cube, Iperf, Perl-File/Data

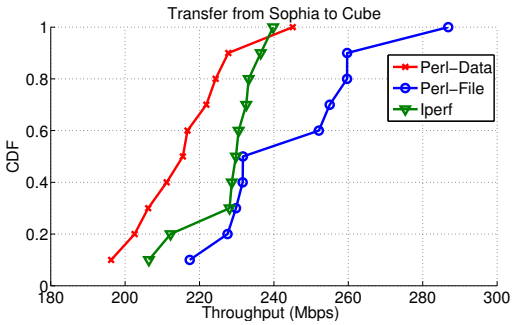

FIGURE 4.30: CDFs of throughput, from Sophia to Cube, Iperf, Perl-File/Data

Also, by comparing the CDFs of throughputs for the three transfer types , Figure 4.30, we find that Perl-Data is closer to Iperf than Perl-File.

If we go back to SND6 reading speed results obtained by hdparm in Figure 4.13, we find that these values are in the range of 80MBps, which is equal to 640Mbps. The reading speed values are very high compared to the throughput values obtained in Figure 4.29 , so for SND6 machine the File transfer is not limited by disk performance.

- RTT

Similarly, for the RTTs of these tests, we have plotted the CDFs, in Figure 4.31. Average RTTs values vary between 17.4ms and 18.5, and again, Iperf is close to Perl file and both are somewhat outperformed by Perl data. We have no clear explanation for this phenomenon. The thirty experiments were interleaved with each other, hence it is a priori excluded that network conditions for Perl file were better than for the other tools. We decided to leave aside the investigation of this problem and focus on the other side if the transfer, from Cube to Sophia.

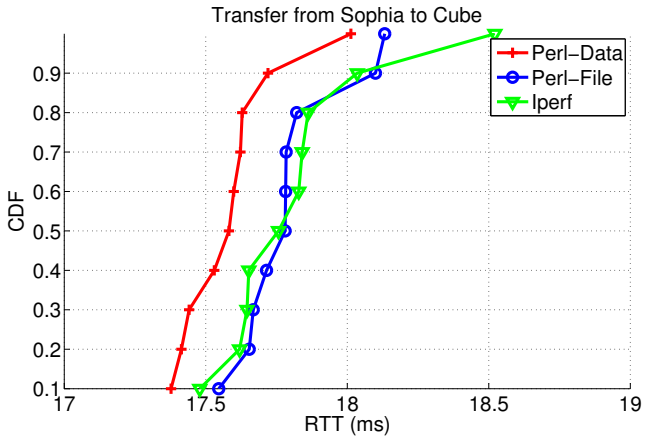

FIGURE 4.31: Results from Sophia to Cube, CDFs of RTTs of the 10 tests

4.4.4 Cube to Sophia

4.4.4.1 Silence during transfers

To make tests from Cube to Sophia, we chose the virtual machine that features TCP Cubic as congestion control protocol i.e., the VM132. We performed the different tests with the same parameters as the direction Sophia to Cube i.e., "-l"= 16K bytes and Perl buffer size is equal to 16K bytes.

Tests with Iperf from Cube to Sophia were made using an SSH Tunnel [wik] [Val10] because of firewall configuration.

- Congestion window

We report in Figure 4.32 the time evolution of the congestion windows for Iperf and Perl transfer tests.

We note that these congestion window values in this direction are lower compared to the Sophia to Cube direction. We also observe some silent periods during the transfer.

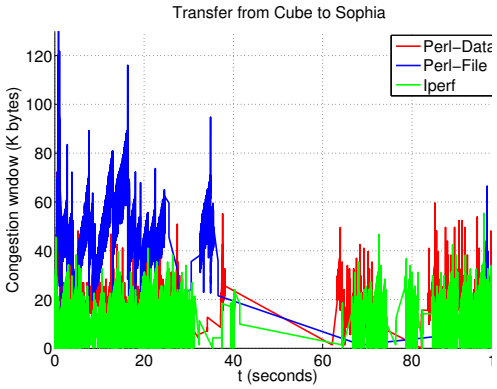

FIGURE 4.32: Congestion window(Kbytes), from Cube to Sophia, Iperf, Perl-File/Data

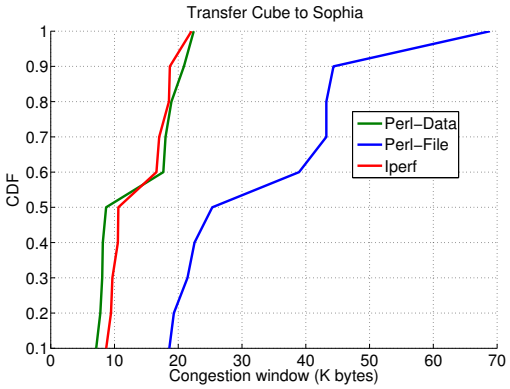

FIGURE 4.33: CDFs of congestion window, from Cube to Sophia, Iperf, Perl-File/Data

To compare the two transfer types, we report CDFs of average congestion windows in Figure 4.33. We observe again a great similarity of Perl-Data and Iperf results.

- Throughput

All throughput values vary between 3 and 25Mbps against about 250 Mbps in the direction towards Sophia Cube, see Figure 4.34. Hence the link is not symmetric or, more probably, it was not engineered similarly in the two directions. We observe that Perl throughput values are comparable with those resulting of Iperf transfers. We note also the existence of silent periods.

In order to check the impact of disk performance on the file transfer results, from Cube to sophia, we compare the disk performance with throughput on the small Linux VM.

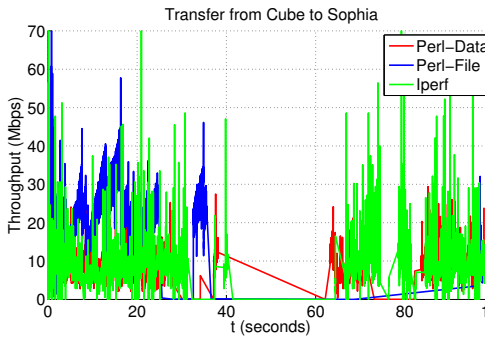

FIGURE 4.34: Throughput(Mbps), from Cube to Sophia, Iperf, Perl-File and Perl-Data

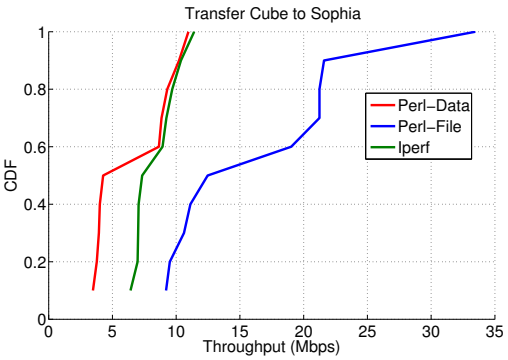

FIGURE 4.35: CDFs of throughput(Mbps), Perl-Iperf, from Cube to Sophia

Figure 4.6 shows that the reading speed values vary between 600KB/s and 1000KB/s, i.e., 4.8Mb/s to 8Mb/s. This is low as compared to the throughput results in Figures 4.34 and 4.35. So disk performance on the small Linux VM limits the throughput values.

The astute reader may think there is a contradiction between what we say here and what we said in the disk section before where we stated that while disk performance of VMs were not in par with the ones of typical physical machine, it would be enough for our needs. However, in the present experiments, we discovered that tcpdump default value on red had systems was to dump the full frames. Hence, our needs of disk speed was equal to the throughput achieved by the flow. For the next set of experiments, we have modified this setting to capture only the headers.

- RTT

We plotted the CDFs of RTTs for different tests in Figure 4.36. Medians are around 16.6ms. Figure 4.36 shows that RTTs for Iperf transfers i.e. using SSH tunnel, are larger than Perl-transfer RTTs. This is caused by the the additional encryption used by SSH tunnel that adds latency.

From the results presented above, it is clear that when sending data from Cube to Sophia, we are facing a problem of silence periods in the traces. Note that the silence periods are observed when capturing traffic both at the sender and the receiver side. It is thus not only a problem of performance when writing on the disk of the Cube machine. On the other hand, there is a performance issue on the Cube machine as we have "packets dropped by kernel" events on the machine.

We can thus suspect that we are facing several problems :

- A disk performance issue
- An issue either at the virtualisation layer or at the transport layer if, e.g., the loss rate is too high and TCP stops emitting data

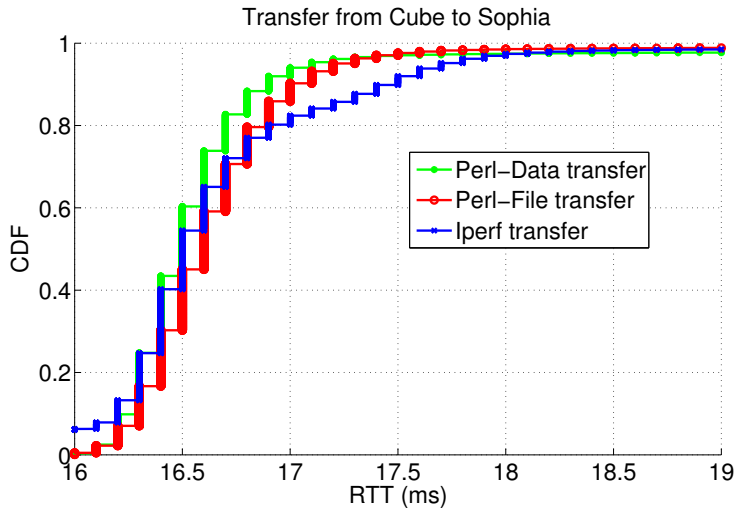

FIGURE 4.36: CDFs of RTTs (ms), Perl-Iperf, Cube to Sophia

To investigate these issues, we:

- Analyzed the tcpdump files and especially the sequence numbers before and after some silence periods

- Performed several transfers in parallel. Indeed, TCP transfers are independent from each other and it is very unlikely that silence periods could affect all TCP transfers simultaneously if the root of the problem is the congestion algorithm of TCP.

- Ran a simple time measurement script in the virtual machines that is writing the current time every second. If ever the machine is stopped for a while, then this is the indication that the root of the problem is the hypervisor.

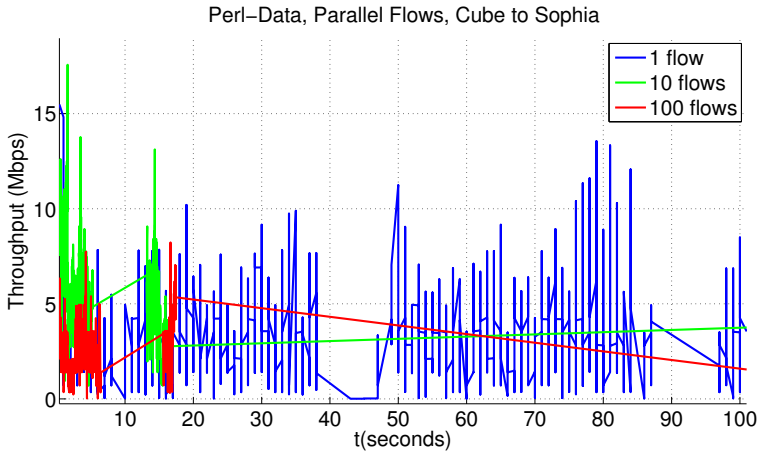

FIGURE 4.37: Average throughput per flow (Mbps), Perl-Data, Parallel flows from Cube to Sophia

We report in Figure 4.37 the time series for the average throughput per flow when we have 1, 10 and 100 parallel flows. We note that as the number of flows increases, silent periods increases, but the average throughput per flow remains at around 4Mbps.

The first step that we have undertaken is to reduce the number of bytes to retrieve using tpdump, to alleviate the problem of disk performance. So, for the remaining set of parallel tests, we used the tcpdump option "-s 96" to capture the set of bytes needed to determine TCP metrics.

At the end of each capture we recovered the information returned by tcpdump; captured packets, packets received by filter and packets dropped by kernel. If the number of packets discarded by the kernel is equal to 0, tcpdump has therefore captured all the packets.

We also used a "tcp.analysis.ack_lost_segment" filter provided by Wireshark. This allows to identify the acknowledgments for which Wireshark can not see the segment sent. If such segment are observed , this means that the capture tool missed some packets.

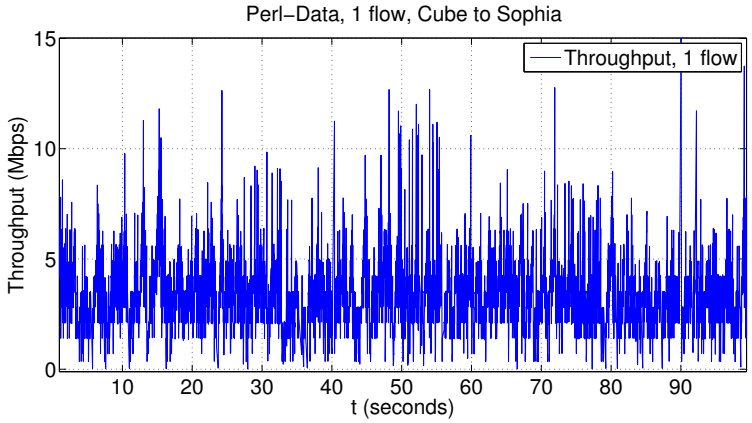

FIGURE 4.38: Average throughput per flow (Mbps), Perl-Data, 1 flow from Cube to Sophia

For example, with a single flow, we have :

- 27908 packets captured
- 27908 packets received by filter
- 0 packets dropped by kernel
- tcp.analysis.ack_lost_segment gives 0

So, there is no packets rejected by the kernel and the packets are sent continuously, where the average rate is at around 3.6Mbps, see Figure 4.38.

4.4.4.2 Measurement results for several transfers in parallel

- Ten flows

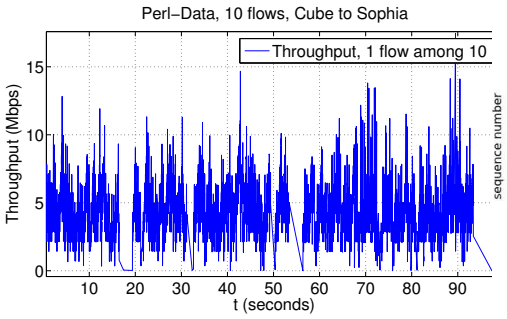

FIGURE 4.39: Average throughput per flow (Mbps), Perl-Data, 1 among 10 flows Cube to Sophia

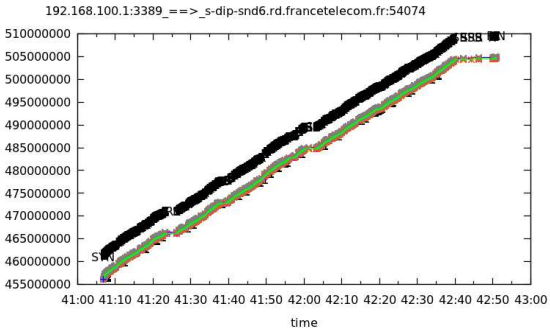

FIGURE 4.40: Time sequence graph, Perl-Data, 1 among 10 flows from Cube to Sophia

For 10 parallel flows, we got 0 packets dropped by kernel, which means that tcpdump was capturing all packets. But we still observe small silence periods, which must be caused by another problem.

We report in Figure 4.40 the sequence numbers for 1 of the 10 tests. By zooming in on silent periods, we find that there is an interruption of transmission, and then the transfer is resumed from the last sequence numbers. When checking the throughput evolution in Cube side (Figure 4.39), we found the same silent periods seen from two sides, so there was no packets exchange.

- Seventeen flows

We further tested the scenario of seventeen flows. In this case the average throughput per flow remains at around 4Mbps, and we have several problems :

- The kernel starts dropping packets

- There are no silence periods with no packet sent from Cube to Sophia, "tcp.analysis.ack_lost_segment" returns a set of segments whose acknowledgments have not been captured by tcpdump

The results with option "-s 96" were better than before, but from 17 parallel flows, the kernel starts dropping packets. Also, we have again silent periods with no packet sent from Cube to Sophia. To check the status of the small virtual machine in Cube when sending packets from Cube to Sophia, we created a script that is executed in parallel with the transfer. This script records the system time every second.

Time history results recovered show that there is time jumps, ie, there are time periods which have not been stored, these time intervals correspond to the missing

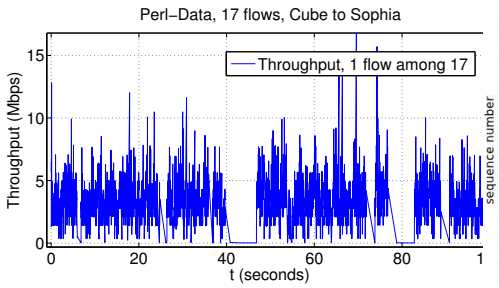

FIGURE 4.41: Average throughput per flow (Mbps), Perl-Data, 1 among 17 flows from Cube to Sophia

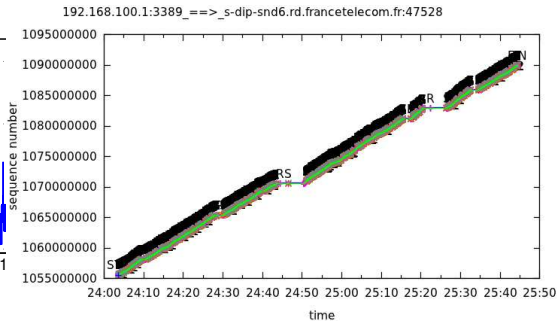

FIGURE 4.42: Time sequence graph, Perl-Data, 1 among 17 flows from Cube to Sophia

silent period observed in throughput curves. This means that the virtual machine have been stopped during these periods.

- One hundred flows

In the case of 100 parallel flows, the overall flow rate can reach 300Mbps. But we observe more silent periods.

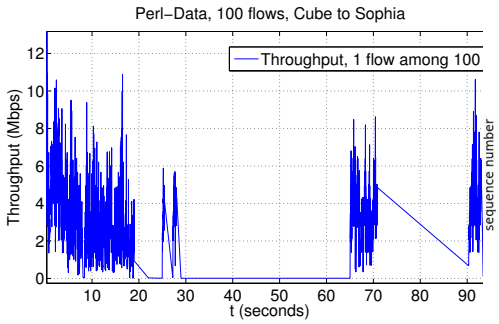

FIGURE 4.43: Average throughput per flow (Mbps), Perl-Data, 1 among 100 flow from Cube to Sophia

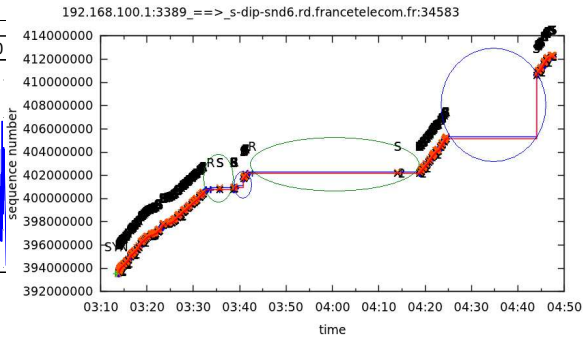

FIGURE 4.44: Time sequence graph, Perl-Data, 1 among 100 flow from Cube to Sophia

Figure 4.44 shows the sequence numbers of a selected flow. 4 silent periods are observed. The green circles represent periods where no packet has been sent to SND6 because there is a continuation in sequence numbers. While blue circles represent packets sent by the VM but not captured by tcpdump because of disk performance. Sequence numbers are larger than those observed at the beginning of silent periods values.

We can see that regardless of the number of parallel flows, the writing speed for tcpdump files remains consistent with dd writing speed.

For each set of parallel flows (1, 50 and 100), We plotted the CDFs of throughput for some flows picked at random in Figures 4.45, 4.46 and 4.47. The average throughput is

equal to 3.5Mbps, 3.8Mbps and 3.4Mbps for 1, 50 and 100 parallel flows, respectively. Therefore, increasing the number of parallel flows, increases the global throughput. We report in Figure 4.48 the mean throughputs for various number of parallel flows, using boxplots format. We observe that the mean throughput by flow vary between 3 and 4 Mbps.

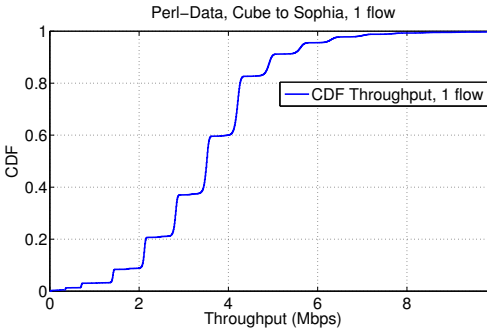

FIGURE 4.45: CDFs of throughputs (Mbps), 1 flow

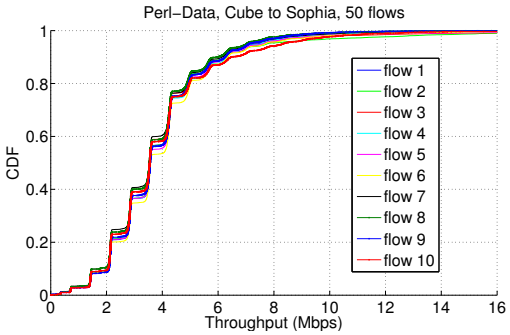

FIGURE 4.46: CDFs of throughputs (Mbps), 10 flows among 50

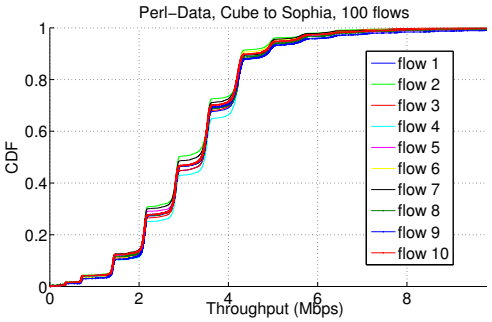

FIGURE 4.47: CDFs of throughputs (Mbps), 10 flows among 100 parallel flows

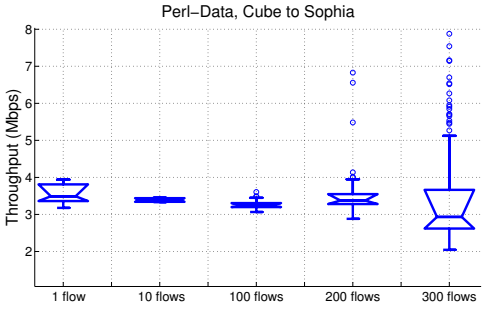

FIGURE 4.48: Box plot throughputs (Mbps), {1, 10, 100, 200, 300} parallel flows

Through these results we can deduce that from Sophia to Cube a shaper limits the average throughput per flow to 4Mbps.

- Packet loss rate

In Figure 4.49 we report the loss probability for different parallel flows, due to the shaper, excluding silence periods due to the virtualization/disk issues. During silent periods with no packet sent from Cube to Sophia, 0 packets were exchanged, so we recovered the trace part before tcpdump starts to drop packets. Then we use these new traces to calculate the loss rate with more precision than the whole trace. We observe that the packet loss rates for 1 to 100 flows are close and vary between 4% and 6%. For these flows the average throughput per flow was about 4Mbps. For 200 and 300 parallel flows, packet loss rate evolve to 10% and the average throughput per flow becomes smaller, because it has reached the threshold capacity for all flows.

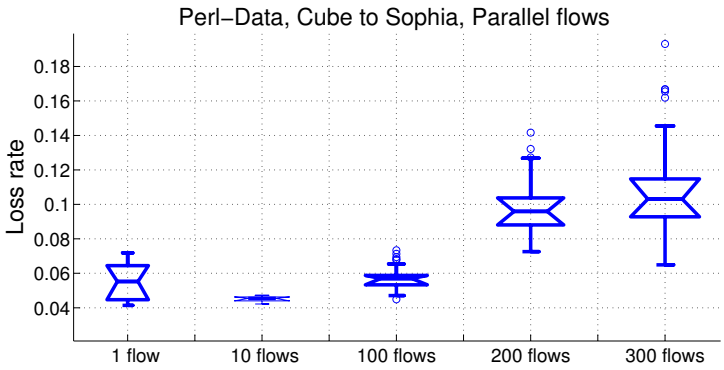

FIGURE 4.49: Box plot loss, {1, 10, 100, 200, 300} parallel flows

4.4.5 Conclusion

In this chapter we have analyzed measurement from/to Cube data center. Several tests have been made between Sophia and Cube, in both directions. Thanks to the various scenarios made, we could distinguish the different characteristics of connections to/from Cube, where the network path between the Cube datacenter and the Sophia Orange Lab is not managed in a symmetric manner. From Sophia to Cube, we can reach apparently 300 Mb/s. In contrast, in the down direction, we are limited to around 3 to 4 Mb/s per connection.

When increasing the number of parallel flows from Cube to Sophia, we encountered some factors limiting the performance of these transfers. All results indicated that there were several roots to the problem: the disk access, the virtualization layer and a shaper. Through different types of experiments, we were able to provide evidences of the existence of these different issues. The initial objective of this chapter was to investigate the competition of several TCP Cubic flows in parallel. We however modified this initial goal into troubleshooting the observed performance issued and delineating between system (disk and hypervisor) and network (shaper) issues. As stated at the beginning of the chapter, Cube was an experimental network that was apparently not optimally engineered but has been turned off and replace by another cloud solution inside Orange. However, this engineering issue was a chance for us as it enabled us to underscore the interplay between system and network issues in a typical cloud solution.

Chapter 5

Synchronization of TCP Cubic connections

5.1 Introduction

In this chapter, we investigate the issue of synchronization among TCP Cubic sources in detail through experimentations in a controlled testbed, measurements with Amazon EC2s servers located in the US and NS-2 simulations. We demonstrate that several factors contribute to the appearance of synchronization in TCP Cubic. We also propose and evaluate two propositions to the TCP Cubic algorithm to alleviate the amount of packets lost during the synchronization episodes.

5.2 Motivating examples

As a motivating example of the synchronization problem, consider the time series of a total congestion window of 10 TCP Cubic flows established between the same pair of sender/receiver that compete for a shared bottleneck, obtained with NS-2, in Figure 5.1. The clear Cubic shape that appears regularly indicates that the flows are synchronized. Note that the code of TCP Cubic in NS-2 is a fork of the one in the Linux kernel. One can obviously argue that the simulations set-up does not catch the complexity of a real operational IP network, and thus synchronization might be the result of ideal simulation conditions. This is why we present in Figure 5.2 the congestion window evolution of 10 transfers in parallel between a server in an Amazon data center of Oregon and a server in our lab. We have obviously no control on the path, but we can clearly observe some periods of high synchronization (two of them highlighted here by red rectangles for an easier reading).

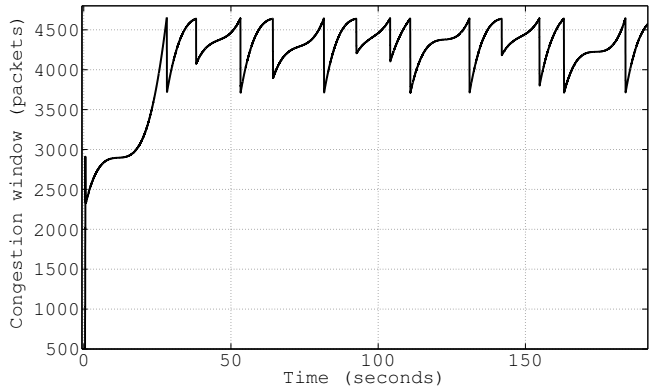

FIGURE 5.1: Total window size (NS-2): 10 TCP Cubic flows,sharing a common bottleneck

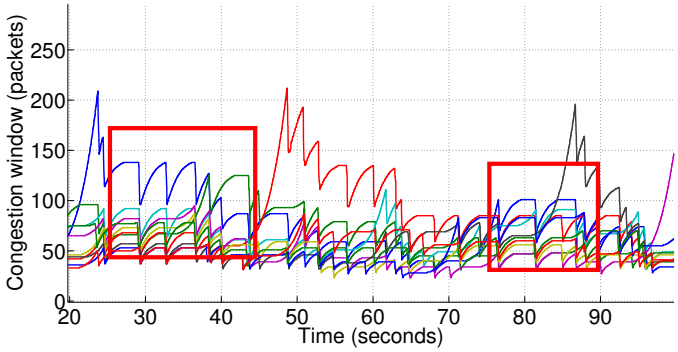

FIGURE 5.2: 10 TCP Cubic transfers between France (I3S lab) and Amazon EC2 data center of Oregon

In [HR08], Hassayoun and Ros found that high-speed versions of TCP may be prone to strong packet-loss synchronization between flows. The authors studied several high-speed versions of TCP and observed, through simulation, the existence of synchronization among sources for all flavors of them.

In [HR09], the same authors evaluate the potential impact of the Random Early Detection (RED) [FJ93b] queue management algorithm on high-speed TCP versions. They study the relation between buffer size, active queue management and loss synchronization. Their study focuses on several metrics: loss synchronization, goodput, link utilization, packet loss rate, and convergence to fairness for high-speed flows. For large buffers, RED strongly reduces the synchronization rate as expected, whereas with droptail, the fraction of synchronized sources is often close to 100%. In contrast, with medium to small buffers, the loss synchronization is roughly similar with both types of queue management strategy.

5.3 Experimental set-up

5.3.1 Testbed

We have created a set of experimental scenarios in our laboratory using the testbed presented in Figure 5.3. It consists of 3 multi-core Dell servers, 2 acting as TCP client or server and one as router. All links are 1 Gb/s links. The router uses netem [1] to control the path latency and capacity, and also the buffer size at layer 3. We use the default FIFO/droptail as server scheduling/queue management policy at the bottleneck.

Various scenarios are created by varying the latency and buffer size. We set the buffer size at the router to {10%, 30%, 50%, 100%} of the bandwidth delay product BDP (i.e., the product of the minimum latency and the capacity of the path). For each scenario, we compare the performance of Cubic with the ones of TCP New Reno. TCP New Reno is used here as a baseline for comparison as it is known to be less sensitive to synchronization than any high speed TCP version [HR08].

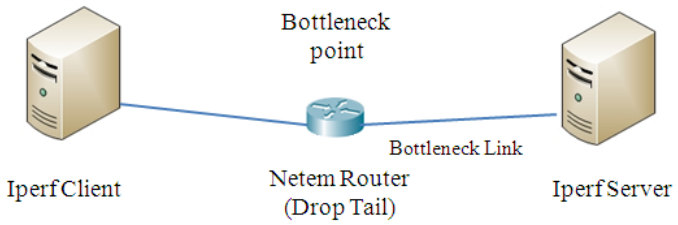

FIGURE 5.3: Experimental network setup

5.3.2 Scenarios

We consider, similarly to [BSC+13], several typical cloud networking scenarios:

- Scenario A - Cloud-clients. We consider here a set of clients that simultaneously download content from a data center (DC). We assume that they share the 1 Gbps access link of the DC and that they have a low path latency to the DC, 20 ms (a typical latency for FTTH clients in France).
- Scenario B - Intra-DC. We consider a set of transfers within a data center (DC) where the path capacity is set to 1 Gbps while the latency is low, 1 ms, reflecting the small physical distance between the servers.
- Scenario C - Inter-DC. This scenario is similar to the previous one, except that the path latency is one order of magnitude higher. We consider 50 ms of latency.

[1] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

Links between machines in our testbed are at 1Gbps. However, we cannot operate netem at such a high speed when controlling both the capacity and the buffer size. We thus constrain the capacity to 100 Mb/s and we inflate the latency of the path in such a manner that the bandwidth-delay product of the path be the same or similar to the consider scenario.

5.4 Experimental results of cloud scenarios

5.4.1 Cloud center scenarios

Scenario A (Cloud-clients)

Table 5.1 contains the targeted (ideal) parameters of the scenario, as well as the ones used in our testbed due to our technical constraints. Note that we define hereafter the bandwidth-delay product of a path (BDP) as the product of the capacity of the bottleneck and the minimum latency of the path.

	Ideal parameter	Testbed parameter
Throughput	1 Gbps	100Mbps
RTT (ms)	20	200
Buffer size (packets)	50	[0.1, 0.3, 0.6, 1]* BDP
BDP (packets)	1667	1667

Table 5.1: Cloud clients scenario

We vary the buffer size (BS) at the bottleneck from 10% of the BDP to 100% of the BDP for both TCP Cubic and New Reno.

Time series of the total window size of one of our experiments taken at random (which were all quantitatively and qualitatively similar), summed over all the connections, is presented in Figure 5.4, for both TCP Cubic and TCP New Reno. From this figure, we note that:

- The congestion window for TCP Cubic reaches larger values compared to New Reno. This means that the number of packets above $BDP + BS$ is larger in TCP Cubic than in New Reno, which causes more losses with TCP Cubic.

- TCP Cubic flows are more synchronized than New Reno. This is indicated by the window reduction during loss episodes closes to 20%. Indeed, a reduction of 20% of the aggregated congestion window is only possible if all sources experience packet loses simultaneously. In contrast, in the New Reno case, flows are less synchronized giving an overall window decrease after loss clearly smaller than 50% (TCP New Reno decreases its congestion window by 50% upon loss detection).

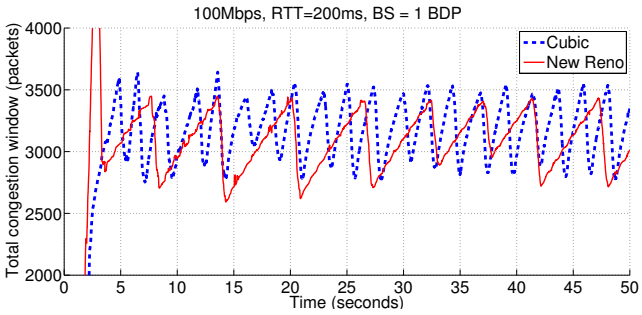

FIGURE 5.4: Total window size (packets)

We report in Figures 5.5 and 5.6 the number of losses and number of synchronized flows per congestion event, of both TCP Cubic and New Reno. We notice that these two criteria are proportional, with higher values for TCP Cubic compared to New Reno, which means a higher synchronization of TCP Cubic connections.

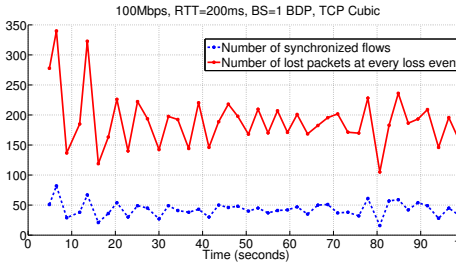

FIGURE 5.5: Number of synchronized flow and lost packets at each congestion epoch, TCP Cubic

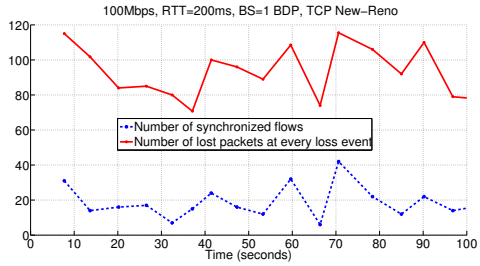

FIGURE 5.6: Number of synchronized flow and lost packets at each congestion epoch, New Reno

Scenario B (Intra-DC)

Table 5.2 contains the targeted parameters of the scenario, as well as the ones used in our testbed.

	Ideal parameter	Testbed parameter
Throughput	1Gbps	100Mbp
RTT (ms)	1	10
Buffer	50	1000
BDP	84	84

TABLE 5.2: Intra-DC scenario

The BDP for this scenario is equal to 84 packets. If one adds to it a buffer size equal to the BDP, it gives an average of 1 packet per flow which is low for our 100 flows in parallel. In such a scenario, the Linux kernel reduces automatically the MTU to values

as low as 40 bytes. This phenomenon leads to different congestion window sizes to obtain a fixed bandwidth, making the analysis of results more complex. To work around this issue, we used a larger buffer of 1000 packets.

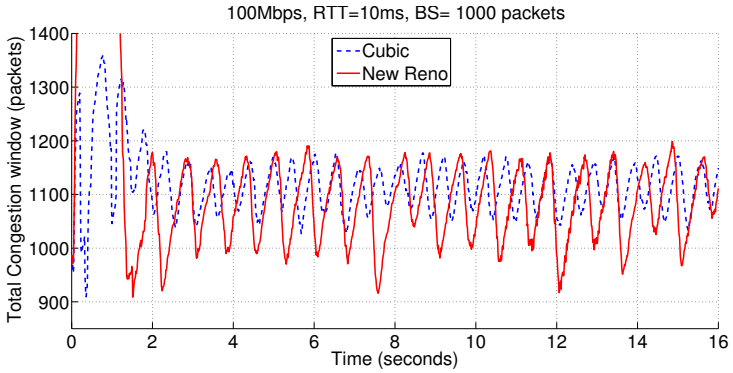

FIGURE 5.7: Total window size (packets)

We report in Figure 5.7 the time series of the total congestion window of both TCP Cubic and New Reno. Note that in this case, TCP Cubic operates in the TCP mode, and therefore, a smaller synchronization is detected. Indeed, the reduction of the total congestion window is less than 20%. Figures 5.8 and 5.9 show clearly that the number of losses per congestion event and synchronized flows approaches the one of TCP New Reno.

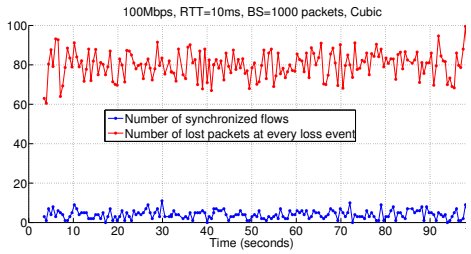

FIGURE 5.8: Number of synchronized flow and lost packets at each congestion epoch, TCP Cubic, BS = 1000 packets

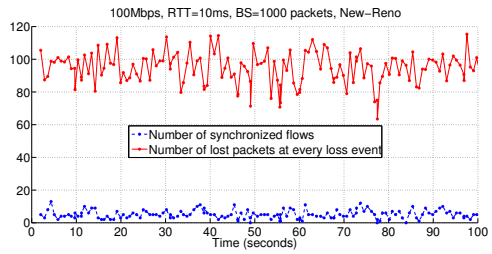

FIGURE 5.9: Number of synchronized flow and lost packets at each congestion epoch, New Reno, BS = 1000 packets

Scenario C (Inter-DC)

Table 5.3 contains the targeted parameters of the scenario, as well as the ones used in our testbed.

For large BDP, the congestion window growth for New Reno is much slower compared to TCP Cubic, so we double the simulation time for New Reno to 200 seconds instead of 100.

	Ideal parameter	Testbed parameter
Throughput	1Gbps	100Mbp
RTT (ms)	50	500
Buffer	500	[0.1, 0.3, 0.6, 1]* BDP
BDP	4167	4167

TABLE 5.3: Inter-DC scenario

With the larger BDP of this scenario, TCP Cubic operated in its cubic mode and we observe again a high synchronization of TCP Cubic sources, see Figure 5.10, where the number of synchronized flows for TCP Cubic is close to 100 while it is below 30 for TCP New Reno.

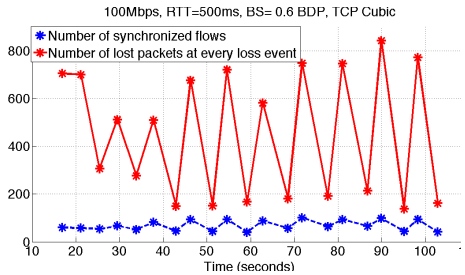

FIGURE 5.10: Number of synchronized flow and lost packets at each congestion epoch, Cubic, BS = 0.6 BDP

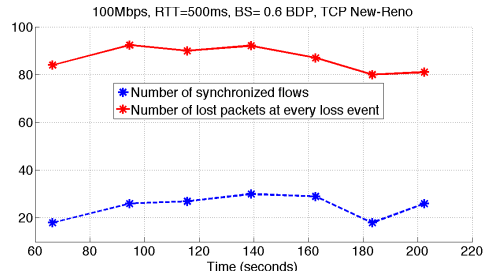

FIGURE 5.11: Number of synchronized flow and lost packets at each congestion epoch, New Reno, BS = 0.6 BDP

5.4.2 Synchronization vs. background traffic

A well-known mechanism to combat synchronization consists in introducing randomness into the network. This can be done by introducing background traffic or inducing random drops through an appropriate buffer management mechanism such as RED [LHB05].

It is known that RED can indeed break synchronization among TCP Cubic sources [HR09], even though the results in [HR09] where obtained purely through simulation. We tested in our tesbed the resilience of synchronization to background traffic. We thus performed again experiments with Scenario C, where synchronization was highly pronounced, adding 100 short flows during the experiment. These flows are short scp transfer. They form a Poisson process with mean inter-arrival time of 1s. The files sent through scp have a size equal to 2MB.

Background traffic starts at time t = 200 seconds in Figure 5.12. We can notice that the overall window is reduced and reaches a value lower than 2BDP, but the TCP Cubic shape of the total window persists, meaning that all flows are still synchronized.

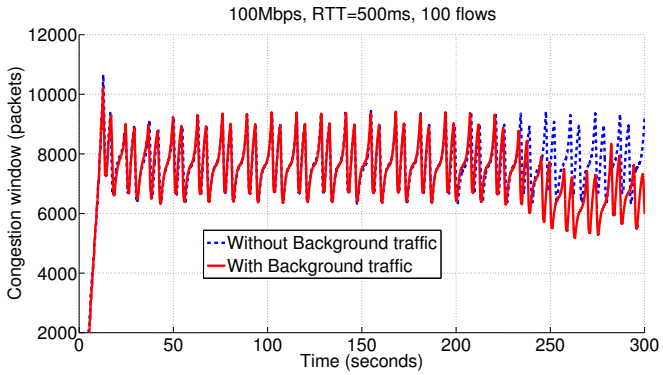

FIGURE 5.12: Time series of window size (packets) with and without background traffic, BS= 1 BDP

We did not test the case of flows with heterogeneous RTTs. Moreover, experiments carried out in [Lei07] show that two flows having different RTTs get synchronized. Indeed, even though flows gets different amount of bandwidth, when congestion events occurs, both flows suffer from packet losses. Such results suggest that even if flows have different RTTs, synchronization occurs. This is also a consequence of the fact that the window growth of TCP Cubic in its cubic mode is independent from the RTT but only from the time since the last loss.

5.4.3 The impact of Fast Convergence

Fast convergence (FC) is designed to make TCP Cubic more fair as it leaves a chance to fresh flows to grab some bandwidth. It is thus not advisable to unset this option in the general case. Still, when focusing on the issue of synchronization, FC becomes a potential suspect of synchronization. Indeed, when performing FC, a source sets its $w_{\max}$ to a value lower than the estimated available bandwidth (the congestion window at the moment where loss occurs). As a consequence, when the number of flows is constant, as it is the case in our experiments, a source that performs FC will reach the available throughout (its share of $BS + BDP$) in an aggressive manner, see for instance Figure 5.1. This aggressive behavior around the equilibrium point can make all sources (even the ones that would plateau at this level) loose some packets and thus enforce their synchronization.

To test the relation between FC and synchronization, we performed again experiments with Scenario C with and without FC for a typical run. We report in Figure 5.13 the total window time series with and without FC. As the extent of window oscillations remains similar, we can conclude that FC is not the only factor behind synchronization.

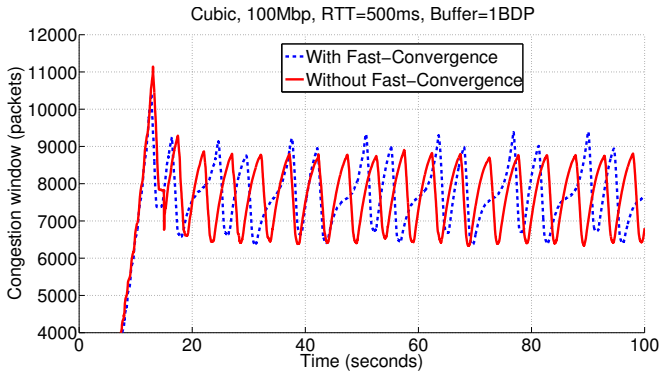

FIGURE 5.13: Time series of total window size (packets) with and without FC, BS= 1 BDP

5.5 Root cause of synchronization

As observed in Section 5.4, TCP Cubic experiences more losses than Standard TCP per congestion event. Therefore, TCP Cubic senders have a higher probability to be synchronized.

Intuitively, high speed TCP variants are more aggressive and therefore, lead to a higher drop rate as compared to the legacy New Reno approach, where the congestion window grows linearly. While this is true for other high speed TCP protocols, like High Speed TCP, in the case of TCP Cubic, if the flat part of the cubic function matches the optimal network capacity, then we can expect to have (at least for this optimal scenario) a low drop rate. Indeed, TCP Cubic is supposed to slowly enter and exit the flat part.

Reality is unfortunately more complex, in fact, several key reasons explain why TCP Cubic flows synchronize each other:

- First, the way TCP Cubic reaches the capacity of the network, which might correspond to its equilibrium point (when the cubic curve becomes flat) or not, depending on the accuracy of the estimate made.
- Second, the way the congestion window actually tracks the cubic curve in the actual implementation can worsen the synchronization phenomenon by letting the source remains a smaller amount of time on its plateau.
- Third, the competition among TCP Cubic flows where the aggressive nature of their probing process far from the equilibrium point can lead to losses for all competing flows.

We discuss each of these points in details in the remainder of this section.

5.5.1 Behavior of TCP Cubic around the equilibrium point

Let *epochstart* be the time right after a congestion event (i.e. $t_0 = epochstart$). Hence, at t_0, the Cubic window will be equal to $0.8 * last\_cwnd$. Using Eq. (2.1), we can see that theoretically, whatever the value of w_{max} and the experienced RTT are, $w_c(t)$ will reach w_{max} at $t_{max} = epochstart + V_{Cubic}$. Furthermore $w_c(t)$ will reach $w_{max}+1$, $w_{max}+2$, $w_{max}+3$ and $w_{max}+4$ at $t_{max}+1.35s$, $t_{max}+1.7s$, $t_{max}+1.95$ and $t_{max}+2.15$, respectively. These consigns values are just the consequence of the design of TCP Cubic whose window growth was made independent from the RTT. Therefore, while there is 0.35s between $w_{max}+1$ and $w_{max}+2$, there is only 0.2s between $w_{max}+3$ and $w_{max}+4$ respectively. Indeed, as $w_c(t)$ moves away from w_{max}, it increases faster. Figure 5.14 provides a graphical description of the period length between 2 successive expected increases of the congestion window.

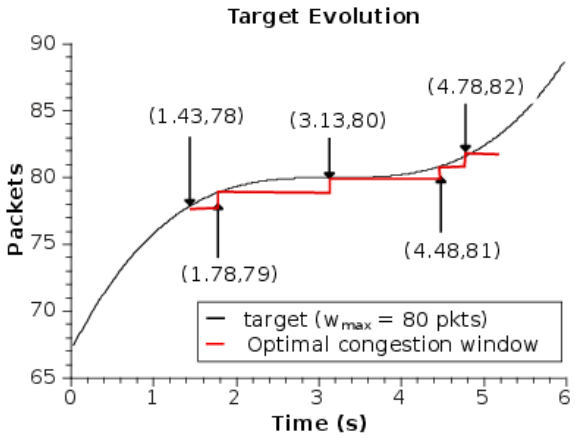

FIGURE 5.14: Target Evolution

Consequently, three different scenarios can be drawn, based on the relative positions of the flat region and the total network available capacity, i.e. BDP+BS. We seek to understand when a source is going to send more than one packet in an RTT when reaching the network capacity. Indeed, if each source adds a single packet, like in New Reno, synchronization should be mild. If they send two packets or more, synchronization will be high.

First scenario: $w_{max} = BDP + BS$. If $cwnd = w_{max} + 1$ leads to a congestion, since between $w_{max}+1$ and $w_{max}+2$ there is a period equal to 0.35s, flows with a total RTT (i.e. propagation delay plus buffering time) smaller than 0.35s will detect the congestion at $w_{max}+1$. Flows with RTTs larger than 0.35s can potentially detect the congestion only when at $w_{max}+2$ (i.e., in a single RTT, such a TCP Cubic flow will increase twice its congestion window). Note that whatever the RTT experienced by New Reno TCP, this last protocol is able to detect a congestion when the congestion window exceeds the

total network capacity by only 1 packet, since it increases its window by at most one MSS per RTT.

Second scenario: $w_{max} = BDP + BS - 1$. When $w_c(t) = w_{max} + 2$, congestion occurs but since between $w_{max} + 2$ and $w_{max} + 3$ there is a period equal to 0.25s, if the total RTT is larger than 0.25s, the connection will potentially increase its congestion window twice (or more depending on the experienced RTT) ending with a congestion window equal to $w_{max} + 2$ or more.

Third scenario: $w_{max} = BDP + BS + 1$. If $w_c(t) = w_{max}$ already exceeds the total network capacity by one packet, since between w_{max} and $w_{max}+1$ there is a period equal to 1.35s, theoretically, only flows with an RTT larger than 1.35s will increase twice their congestion windows before detecting a congestion. Hence, after a congestion event, w_{max} will be set again to $w_{max} = BDP + BS + 1$ and the number of losses will be small. Note that the theoretical TCP Cubic target is able to converge to a $w_{max} = BDP + BS + 1$ from any w_{max} value, like shown in Figure 5.15.

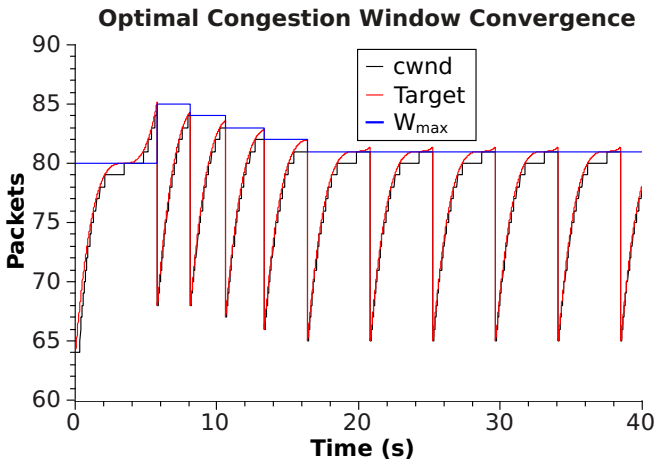

FIGURE 5.15: Converge properties of the optimal congestion window ($BDP + BS = 80$).

To sum up the three above scenarios: (i) overestimating the bottleneck is not a big issue as there is little chance that the sources increases several times its congestion window when entering the flat region (it should have a RTT larger than 1.35s); (ii) precisely estimating the bottleneck precisely means that the source will be too aggressive if the RTT is larger than 0.35 s and (iii) if the source underestimates the capacity, the RTT for which it becomes too aggressive is 0.25s. The latter scenario is thus the more dramatic one. We can note that Fast Convergence, that forces to set its $w_{\max}$ equal to $0.9 * w_c(t)$ upon a loss leads exactly to the latter scenario. FC is thus a net contributor to the too high aggressiveness of a TCP Cubic source.

As an illustrative example, the Amazon EC2 experiment presented in Figure 5.2 was a case where the base RTT (measured by ping) was 190 ms. Hence, when adding the

buffer size along the path (which we do not know), there is a high probability that the RTT will be above 250 ms. This RTT combined with the use of Fast Convergence explains why we observe episodes of synchronization.

The above analysis assumes a perfect source in isolation. In practice, the actual implementation as well as the competition among TCP Cubic flows worsen the situation as we discuss below.

5.5.2 Tracking of cubic function in the actual implementation of TCP Cubic

In the real life, the tracking of the target window is not perfect. We have extracted the algorithm used by TCP Linux from NS-2, which is supposed to be the same as the one in some Linux kernels, to build our own simulator and be able to trace the several variables used inside. We have found that, assuming a constant reception of ACKs and a total RTT of one second, when the congestion window reaches w_{max}, it will stay in the flat region for period shorter to 1.35s (around 0.8s as we can see in Figure 5.16). Such a result was confirmed by NS-2 assuming the same RTT. Staying a shorter period on the plateau can lead to have too many losses when getting above the network capacity.

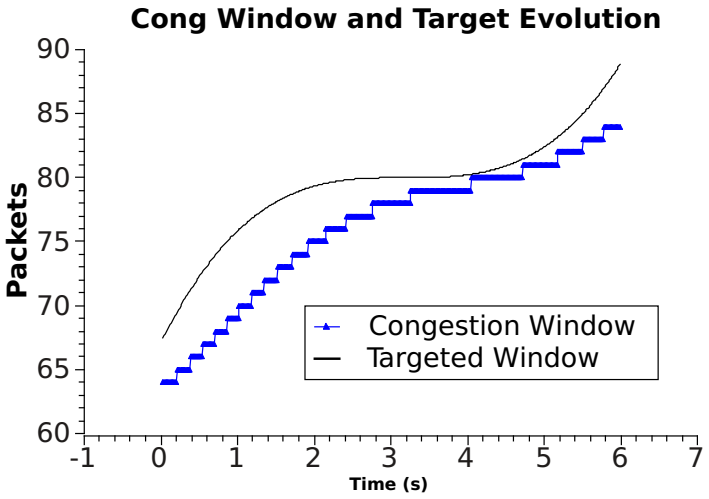

FIGURE 5.16: More real Cubic congestion window evolution.

5.5.3 Competition around the equilibrium point

Let us suppose that during a given congestion event, the total capacity was exceeded by n packets only (where n is equal to the number of flows) as the legacy New Reno version of TCP does, and that the congestion window of each TCP Cubic flow was equal to the actual share that each connection deserves. In this scenario, it is highly likely that not every flow would experience a packet loss. Put differently, the synchronization

between flows would be low. However, those TCP Cubic flows that did not experience losses will enter their convex region, and thus their congestion window will grow faster and during the next congestion event, the number of dropped packets will increase. This will finally lead to a high synchronization between flows.

Figure 5.17 illustrates graphically our arguments provided in this paragraph by zooming on a specific moment in time of one simulation we performed. We observed a first loss event where only two flows are affected. We next observe that the flows that experience losses will soon again plateau around the equilibrium point. In contrast, the ones that did not lose enter the aggressive probing part of the cubic curve. Even, if they are just leaving their plateau as it is the case here, the number of losses that they induce in the buffer is such that all four sources losse packets at the same time instant, i.e., they become synchronized.

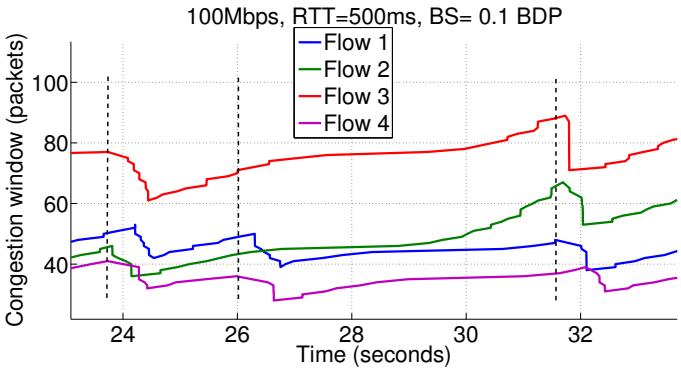

FIGURE 5.17: TCP Cubic leading to high synchronization

5.5.4 Discussion

From the analysis presented above, it is clear that the RTT of the connection plays a key role to determine the level of synchronization we might expect. Referring back to the methodology presented in Section 5.4, it becomes clear, in light of what we discussed in this section, that increasing the RTT to obtain the same BDP as in the ideal cloud scenarios that we devised, was introducing a bias towards more synchronization. For the intra data-center scenario (scenario B) where the ideal RTT was 1ms, synchronization is likely not to occur. This is confirmed by our experimental results (see Figure 5.8) because the RTT in the experimental testbed is still low (10ms). It should be the same in the inter data-center case (scenario C) where the ideal RTT is 50 ms, while we observed synchronization by working at 500 ms. It is even highly possible that TCP Cubic operates in the TCP mode and not the Cubic mode in such a scenario, in which case the means-field model that we proposed in [BSC+13] demonstrated that no synchronization should be present.

As illustrative examples of the above points, we report in Figure 5.18 a typical experiment made between a pair of servers in the Oregon data center of Amazon where the RTT was in the order of a ms. We never observed any synchronization in this case (out of the numerous trials we made). While Figure 5.18 reports the congestion window of each individual flow, Figure 5.19 reports the aggregate congestion window and we can observe that it never decreases by 20% (as 80% of 1200 is 960 and we are always above this line).

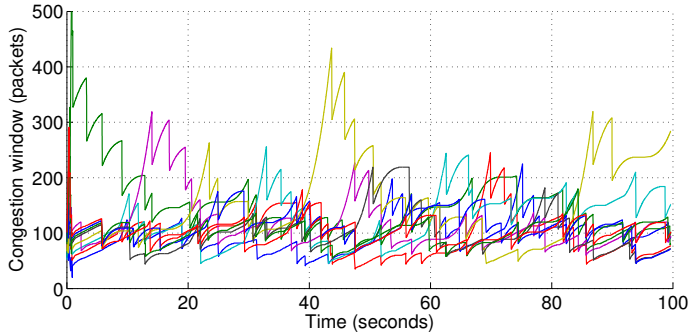

FIGURE 5.18: Intra data center transfers - 10 flows, individual Congestion Window, 10 flows

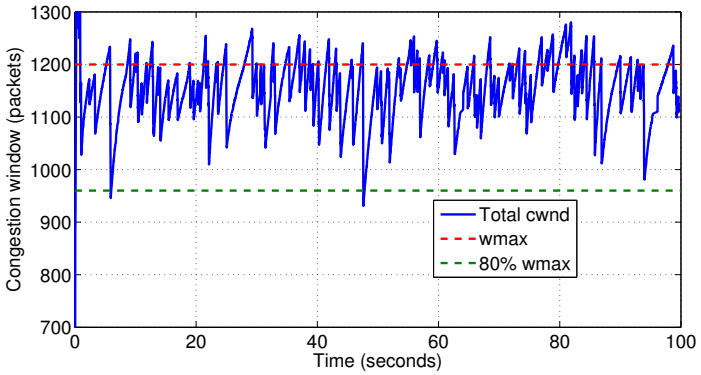

FIGURE 5.19: Intra data center transfers - 10 flows, aggregate Congestion Window

The previous experiment was obtained with 10 flows. With 100 flows between the pair of servers, we observe in Figure 5.20 that the flows now operate in the TCP mode of Cubic with no synchronization.

However for the case of a remote client or distant data centers transfers, synchronization is likely to pop up. The Amazon EC2 experiment in Figure 5.2, where 10 flows were created between France and the Amazon EC2 DC located in US, is a good illustration of this point. Additionally, since flow synchronization leads to a reduction of around 20% of the total traffic, buffer sizes smaller than 20% of the expected average BDP can lead

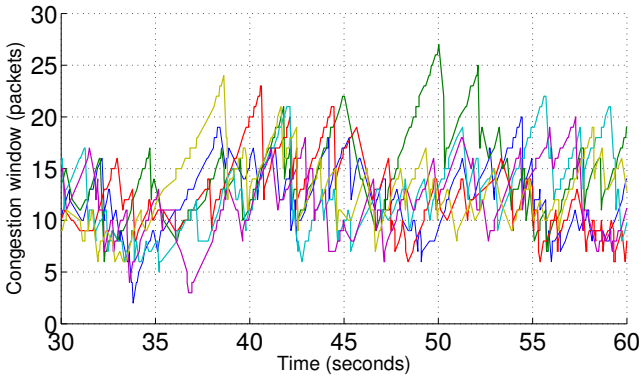

FIGURE 5.20: Individual Congestion Window, Intra data center transfers - 100 flows

to an under utilization of the available bandwidth, specially if the maximum experienced RTT of the traffic exceeds 250ms.

5.6 Alleviating Synchronization

In this section, we aim at investigating solutions to work around the problem of synchronization faced by TCP Cubic. As the root of the problem lies in behavior of TCP around the equilibrium point, we investigated the two following approaches:

- First, we linearize TCP Cubic when it operates close to its plateau. More precisely, we enforce TCP to increase by one MSS per RTT in the range $[w_{\max}-2, w_{\max}+2]$. We call this modification LinCubic.

- Second, as we observed that the actual implementation was not accurately tracking the cubic curve, we devised a version that fulfills this goal. We call this modification AccuCubic.

To evaluate the impact of those different modifications, we implemented them in NS-2 and started observing their behavior in the case of a single flow. We consider a link capacity equal to 1Mbps, a latency equal to 500ms and a buffer size equal to one BDP (41 packets). The network capacity is thus $w_{max_{ideal}} = BDP + BS = 82$ packets. In Figures 5.21 and 5.22, we report the evolution of the congestion window.

We can observe that FC indeed plays a significant role. It globally worsens the situation for TCP Cubic. We observe that LinCubic performs very well by precisely tracking the network capacity with or without FC. We have observed also that AccuCubic prefers that FC be turned off, but we do not have a clear explanation for this phenomenon.

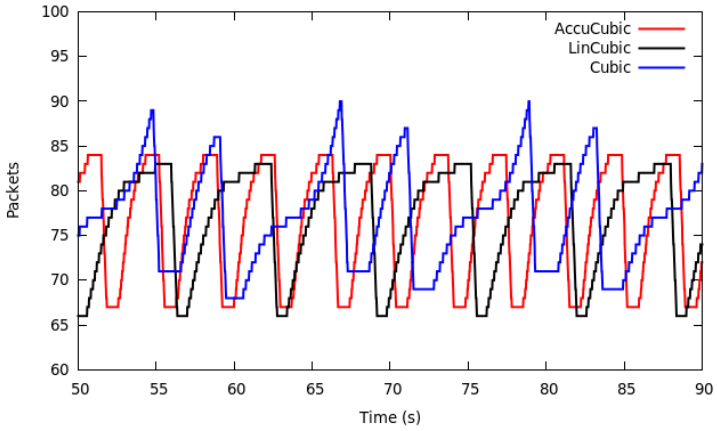

FIGURE 5.21: NS-2 Simulations - A single flow, With FC

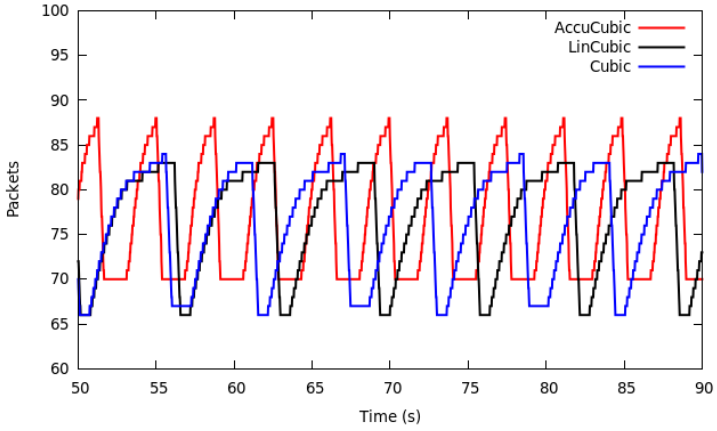

FIGURE 5.22: NS-2 Simulations - A single flow, Without FC

We further tested the potential benefit of those modifications in the case of 100 flows competing for the bottleneck. We consider various scenarios by varying the RTT from 100 to 500 ms and considering different buffer sizes from 0.1×BDP to 1 BDP. For each scenario, we performed 10 runs. We report the number of synchronized flows in the case of 500ms and a buffer size equal to one BDP in Figures 5.23 and 5.24 for a typical run. Results are consistent with the case of a single flow: LinCubic noticeably decreases the number of synchronized flows as well as AccuCubic when FC is turned on. When FC is turned off, only LinCubic performs better than TCP Cubic.

At this stage, we believe that even if the behavior of TCP Cubic can be improved, as exemplified by LinCubic and AccuCubic, the solution to combat synchronization might not be only sought in the TCP implementation itself. Indeed, those improvements might always be partly mitigated by the competition among TCP Cubic flows outlined in Section 5.5.3. Solutions to the problem of synchronization should thus also be looked for outside TCP itself, e.g., through the use of buffer management mechanisms like RED or Codel [NJ12b]. We further investigate this issue in chapter 6.

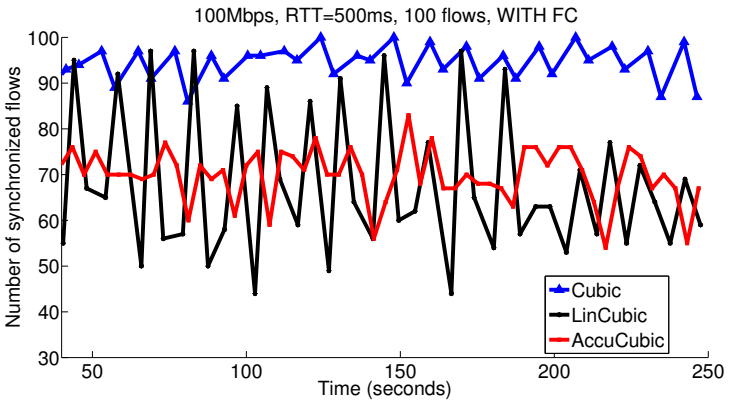

FIGURE 5.23: NS-2 Simulations -100 flows, RTT=500ms, With FC

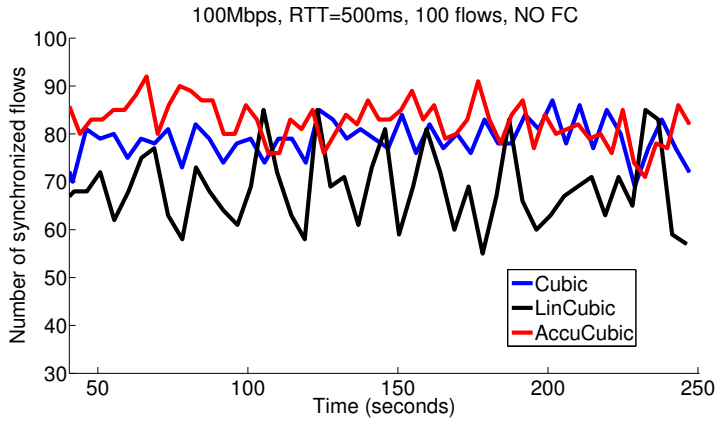

FIGURE 5.24: NS-2 Simulations -100 flows, RTT=500ms, Without FC

5.7 Conclusion

In this chapter, we have explored in detail the root causes behind the synchronization of TCP Cubic flows that can be easily observed through simulations for instance. We made use of a combination of experiments in a testbed, simulations and some experiments in the wild to analyze the extent of the phenomenon.

The controlled nature of our testbed enabled us to precisely analyze the phenomenon of synchronization and discover its root causes. Simple experiments in the wild (with a distant EC2 datacenter) confirmed that the phenomenon can affect real world transfers.

We discovered that while TCP Cubic is known to provide a form of fairness by making the window growth independent of the RTT of the connection (which TCP New Reno is unable to do as the window growth is tightly coupled to the RTT of each connection), synchronization is a subtle result of the interaction between: (i) the way TCP Cubic reaches the capacity of the network, (ii) the relation between the RTT of the connection and (iii) the window growth of the cubic function that occurs at specific time instant. In addition, Fast Convergence, that biases the estimate of the capacity made by TCP Cubic to give a chance to other connections to grab some bandwidth, significantly increases the synchronization phenomenon. Last but not least, even with a perfect estimation of the bottleneck capacity, synchronization can occur starting from an unsynchronized situation where some flows loose while some others do not. Indeed, the sources that did not loose are likely to start probing aggressively (due to the shape of the cubic function) which can result in massive losses for all flows later on. This can be observed especially if the RTT is large. When the RTT is low for all connection, TCP Cubic is quite immune to synchronization.

We proposed and evaluated two modifications to the TCP Cubic algorithm that aim at combating synchronization. They improved noticeably the situation and we intend to explore in the next chapter how they can be combined with advanced queuing mechanisms like CoDel, to further reduce synchronization.

Acknowledgment

This work was partly supported by AWS in Education Grant award.

Chapter 6

Impact of queue management mechanisms on synchronization

6.1 Introduction

In the previous chapter, we proposed and evaluated two modifications to the TCP Cubic algorithm LinCubic and AccuCubic, that aim at combating synchronization.

The solution to fight against the problem of synchronization might also be looked for outside TCP itself, e.g., through the use of buffer management mechanisms.

In this chapter, we intend to evaluate the potential impact of the queue management algorithms on synchronization. Also, we explore how LinCubic and AccuCubic can be combined with advanced queuing mechanisms like CoDel, PIE and ARED to further reduce synchronization.

6.2 Active Queue Management Mechanisms

Buffers was initially designed to avoid packet drops, but they can lead to highly elevated queuing latency and jitter. This phenomenon is called 'bufferbloat' and was introduced by Jim Gettys in late 2010 [GN11]. Bufferbloat may worsen the user-perceived Internet performance, most specifically for latency-sensitive applications such as real-time interactive multimedia, online gaming and even web browsing, especially when they share the bottleneck queue with long-lived TCP connections.

A popular counter measure is the adoption of active queue management (AQM) schemes in the network to improve the performance of the Internet. AQM mechanism is a congestion control mechanism at a router for controlling the number of packets in the router's buffer by actively discarding some arriving packets. It can shorter the average delays in the router's buffer and can also achieve higher throughput.

The primary goals of any AQM mechanism are:

1. Let the buffer absorb packet bursts while preventing it from sustaining long standing queues;

2. Break any synchronization between flows.

If possible, AQM mechanisms should also be able to protect flows from being starved by other more aggressive or misbehaving flows, as well as to support Explicit Congestion Notification (ECN) [Flo94].

Several surveys have been conducted in the literature to capitalize the extensive existing AQM research. Among them, we cite [Ada13] which is an AQM taxonomy published in 2013 where authors discuss the general attributes of AQM schemes, and the design approaches taken such as heuristic, control-theoretic and deterministic optimization. They revisit AQM research from 1993 with the first algorithm, Random Early Detection (RED), to 2011. They used a set of criteria to classify and compare AQMs: (i) mechanisms of operation; (ii) context of use; and (iii) performance criteria.

We present, in the remainder of this section, the AQM policies that we consider: ARED, Codel and PIE. Where CoDel and PIE are two AQM mechanisms that have recently been presented and discussed in the IRTF and the IETF as solutions for keeping latency low.

6.2.1 RED

Random early detection (RED), also known as random early discard or random early drop was introduced by Floyd and Jacobson [FJ93a], for controlling the average queue size two decades ago.

RED monitors the average queue size and drops (or marks when used in conjunction with ECN) packets based on some probability function. If the buffer is almost empty, all incoming packets are accepted. As the queue grows, the probability for dropping an incoming packet grows too. When the buffer is full, the probability reaches 1 and all incoming packets are dropped.

More precisely, the decision of dropping (or tagging) a packet is based on a running estimate of the average queue q at the buffer. This average value is updated with every incoming packet. A piecewise-linear drop probability function p(q) is used to select the packets that will be "marked" with a congestion signal(i.e., discarded or tagged). When q gets above a given threshold q_{min}, incoming packets are marked with probability $p(q) > 0$ up to a maximum value of p_{max} which is typically $\ll 1$. If q becomes greater than a threshold q_{max}, then p = 1. In the so-called "gentle RED" variant, p gradually increases from p_{max} to 1 when q is in the $[q_{max}; 2q_{max}]$ range, see Figure 6.1 .

Given that RED needs careful tuning of its parameters for various network conditions, most network operators do not turn RED on. In addition, RED is designed to control

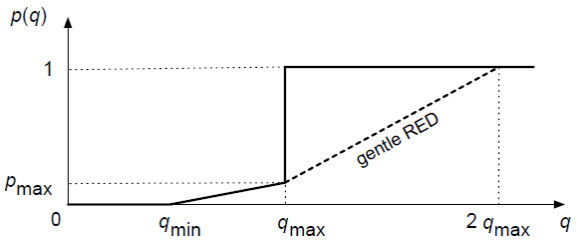

FIGURE 6.1: RED: drop probability function [HR09]

the queue length which would affect delay implicitly. It does not control latency directly. We provide in Figure 6.2 a simplified RED algorithm chart.

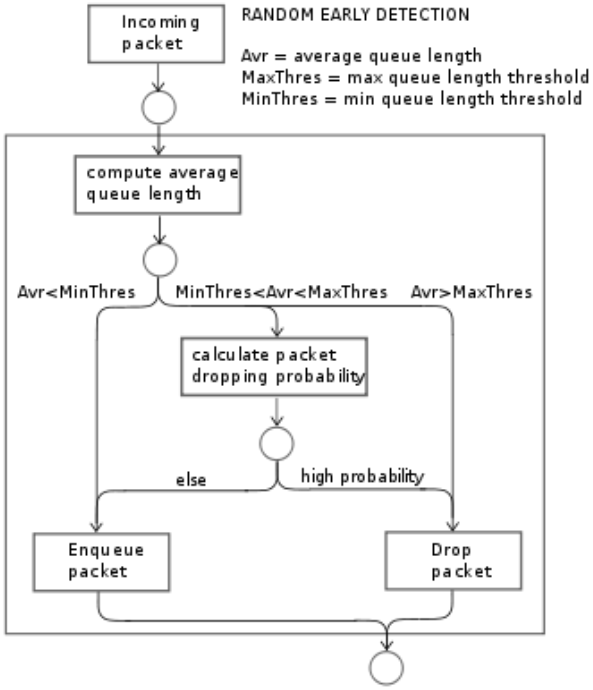

FIGURE 6.2: Simplified RED Algorithm Behavior [ed14]

6.2.2 ARED

Adaptive RED (ARED) [FGS01] dynamically adjusts RED's maximum drop probability (p_{max}). It observes the average queue length ($\overline{q}$) to infer whether to make RED more or less aggressive. Similar to RED, ARED keeps two thresholds (q_{min} and q_{max}) which, to correlate with a single target queuing value, are set to $0.5 * target_{queuing}$ and $1.5 * target_{queuing}$ in accordance with the rules in [FGS01]. If $\overline{q}$ oscillates below q_{min}, early detection is too aggressive. On the other hand, if $\overline{q}$ oscillates above q_{max}, early detection is too conservative. Using an Additive Increase /Multiplicative Decrease

(AIMD) policy, ARED adaptively changes p_{max} so that the average queue length oscillates around $(q_{max} + q_{min})/2$. ARED updates p_{max} periodically after every interval (500 ms by default), and adaptively sets most of RED's parameters based on a target average queue as an input parameter.

6.2.3 CoDel

CoDel [NJ12b] [NJ12a] was first published in May of 2012 by Kathy Nichols and Van Jacobson. It was proposed to control the latency directly to address the bufferbloat problem [GN11]. CoDel requires per packet timestamps, and packets are dropped at the dequeue function after they have been enqueued for a while.

It assumes that a small target queue delay is tolerable so as to achieve good link utilization. CoDel uses additional logic to avoid re-entering the dropping state too early after exiting it. CoDel only enters the dropping state when the minimum queuing delay has exceeded target delay for an interval long enough to absorb normal packet bursts. This ensures that a burst of packets will not experience packet drops as long as the burst can be cleared from the queue within a reasonable period. Figure 6.3 presents a simplified algorithm behavior for CoDel.

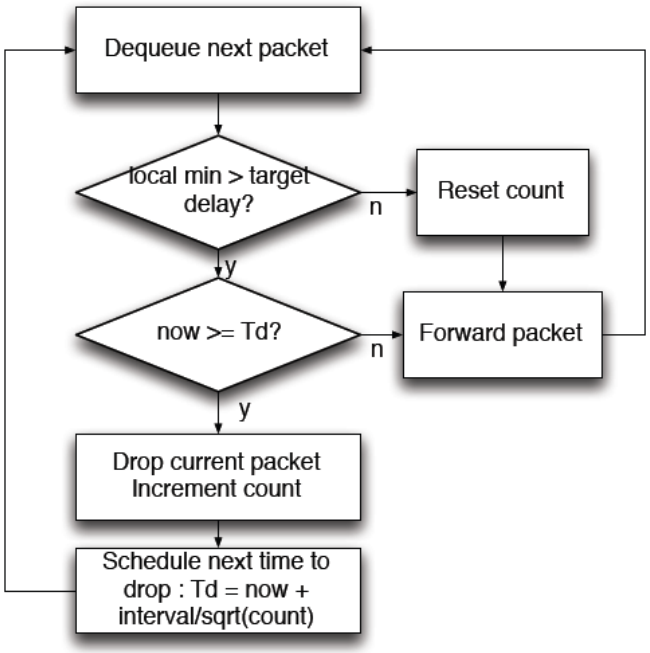

FIGURE 6.3: Simplified CoDel Algorithm Behavior [Whi13]

A new version of Codel is the Fair Queuing Controlled Delay FQ_Codel. It is queuing discipline that combines Fair Queuing with the CoDel AQM scheme. FQ_Codel uses a

stochastic model to classify incoming packets into different flows and is used to provide a fair share of the bandwidth to all the flows using the queue[1].

6.2.4 PIE

PIE (Proportional Integral controller Enhanced) [RPC+13] is being developed by Cisco and was first presented in the October 2012 IETF meeting. It is based on the theory of linear feedback control.

In [RPC+13], the authors describe in detail the design of PIE and its operations. PIE directly controls latency like CoDel. Similar to RED, PIE randomly drops packets at the onset of the congestion. The congestion detection, however, is based on the queueing latency like CoDel instead of the queue length like conventional AQM schemes such as RED. Furthermore, PIE also uses the latency moving trends: latency increasing or decreasing, to help determine congestion levels.

PIE attempts to control the queue depth via linear-feedback control, It monitors the queue depth, and adjusts drop-probability via linear control-theory mechanisms. It establishes three operating points for its linear controller, and switching between them in an attempt to select the mode that results in the best behavior.

The core of the algorithm consists in the prediction of the queuing latency for the packet that is currently at the tail of the queue, based on the current queue depth and an estimate of the egress data rate.

Authors in [NK13] have identified two key parameters that are common to RED, CoDel and PIE mechanisms:

- **Target delay** : CoDel starts dropping packets when the queuing delay has been above the target delay for a certain amount of time, while PIE continuously updates its dropping probability based on the difference between the current queuing delay and target delay. Although ARED does not explicitly maintain a target delay value, when ARED is used in a fixed bandwidth scenario, it is possible to derive its corresponding target delay from a given target queuing.

- **Interval**: Most AQMs require a certain time interval to update their dropping/-marking probability or decide whether to discard the incoming/outgoing packets. The use of this interval differs from one AQM to another. CoDel uses it to decide how long the queuing latency can stay above target delay before switching to dropping mode. On the other hand, PIE and ARED use this quantity to update the dropping/marking probability.

[1] http://tools.ietf.org/html/draft-hoeiland-joergensen-aqm-fq-codel-00

6.3 Bibliographical comparison of AQMs

[RPC+13] is the first paper that present PIE, where authors evaluate the performance of the PIE scheme in both NS-2 simulations and testbed experiment using Linux machines. They compare PIEs performance against RED in simulations and against CoDel in testbed evaluations.

Compared to RED, PIE quickly adjusts the dropping probability in a couple of seconds, and restore the control of the queuing latency to be around equilibrium value.

With experimental results, authors of PIE show that for TCP traffic, PIE and CoDel are able to control the queuing delay reasonably well. Under the mixture of TCP and UDP traffic, CoDel cannot control the latency under the target values of 5ms and 20ms respectively.

In [Whi13], the authors compare CoDel, PIE and Stochastic Flow Queue-CoDel (SFQ-CoDel). They tested 17 different user traffic scenarios that comprise different mixes four applications, e.g., VoIP/Gaming, Web browsing, File upload (either FTP or BitTorrent) and Constant bit rate UDP traffic.

The 17 scenarios are grouped into three groups; Light traffic, Moderate traffic and Heavy traffic.

Results show that :

- CoDel provides some attractive benefits relative to Buffer Control. The main benefits being improvements in gaming latency, page load time and short term TCP performance, as well as improvements in VoIP performance in certain conditions.
- SFQ-Codel shows extremely good performance in the majority of the tested scenarios, except BitTorrent.
- The PIE algorithm outperforms buffer control, and in most cases outperforms CoDel. Another concern is that PIE may require more elaborate tuning based on the network technology and conditions. One potential improvement in PIE that has been discussed by Cisco is to pair it with the SFQ concept. While this may give even further improvements in some scenarios, it would introduce the same issue that SFQ-CoDel has with BitTorrent.

In summary, SFQ-CoDel outperforms all other approaches in the majority of cases.

In [NK13], the authors perform an experimental evaluation of ARED, PIE and CoDel using real-world implementations, in both wired and wireless testbeds. This was the first study to compare these 3 AQMs through experiments. To better understand the basic behavior of CoDel, PIE and ARED, they conducted a first set of experiments using

only a single TCP flow and observed the trade-off between RTT and goodput in a given period. Results show that:

- CoDel and PIE produce higher maximum and longer distribution tail of queuing delay than Adaptive RED.
- CoDel achieves the best goodput among the three AQMs, while ARED suffers from low link utilization for lower target delay values of $\leq 10ms$ when only a single TCP flow is present at the bottleneck.

Authors further evaluate the three AQM mechanisms under different network conditions, and investigate how they behave in scenarios with different number of flows and different parameter settings (RTT, target delay and the interval).

When varying Target delay, under high congestion, PIE and CoDel show longer queuing delay distribution tails, meaning more fluctuations in RTTs.

For the Interval variation, as CoDel and PIE use an update interval time that can be adjusted from user-space and ARED uses a static fixed interval time of 500 ms, authors only study CoDel and PIE.

For CoDel, smaller interval value than the default one could be recommendable. PIE achieves better queuing delays with fine-grained intervals and better goodput with coarse-grained intervals.

Finally, for short RTTs, ARED is able to achieve significantly better queuing delay than CoDel and PIE with almost exactly the same goodput level. For intermediate (100 ms) and longer (500 ms) RTT ARED performs significantly better than CoDel and PIE in terms of queuing delay while loosing little goodput.

In [GKF13], the authors compared PIE and CoDel using Adaptive RED as a reference. Through simulations, they show that to achieve a small queuing delay PIE and CoDel both increase packet loss. Also, they found that PIE performs better than CoDel in terms of packet loss rates affecting video quality. And the performance of ARED is comparable to that of PIE and CoDel in constant capacity links. Simulations results show that ARED, PIE and CoDel can increase loss that may negatively impact some applications like unreliable video. So, to compare these AQMs, several results derived from these studies were conflicting.

Active queue management versus loss synchronization

In [HR09], the authors evaluate the potential impact of the Random Early Detection (RED) queue management algorithm on high-speed TCP versions : High Speed TCP (HSTCP), Hamilton TCP (H-TCP), BIC, TCP Cubic and Compound TCP (CTCP).

They focus on several metrics: loss synchronization, goodput, link utilization, packet loss rate, and convergence to fairness for high-speed flows.

The simulation setup in [HR09] is as follow: for every long-lived flow there are one or more background flows sharing the corresponding access links. And each background flow is modeled as an on-off process, with exponentially-distributed "off" (thinking time) periods of mean 4s alternating with "on" (activity) periods. The version of TCP used for background traffic is TCP New Reno.

The authors observe that RED tends to"break" loss synchronization between TCP flows. More precisely, it avoids causing global synchronization as much as possible. It strongly reduces the synchronization rate for large buffers (i.e., Buffer=$C \times \overline{RTT}$ and Buffer=$0.63 \times C \times \overline{RTT}/\sqrt{N}$, where N is the number of flows), whereas with droptail, the synchronization between sources is very high. In contrast, with medium (i.e., 2000 packets, or 2% of the BDP) to small buffers (i.e., 100 packets), RED and droptail give similar loss synchronization level.

6.4 Simulation results

In order to evaluate the potential benefit of AQMs for solving the synchronization issue of TCP Cubic, we started with a simulation study of the three active queue management algorithm Codel, PIE and ARED. This study is similar to Ros study where authors performed an experimental evaluation of these three AQMs using real-world implementations, in both wired and wireless testbeds.

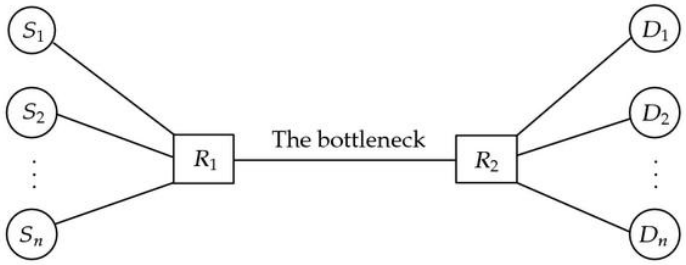

FIGURE 6.4: Structure of the network

The network topology used for NS-2 simulations is a dumbbell with one bottleneck link as shown in Figure 6.4. There are N long-lived flows sharing the bottleneck link with the capacity C.

We proceeded as in the paper [NK13] and compared the 3 AQMs in terms of delays and goodput, while varying certain parameters (delay, interval, RTT) through NS-2 simulations.

We considered the same parameter than in Ros study; the base RTT is equal to 100ms, the queue size is set to 1000 packets and the bottleneck link capacity is set to 10Mbps. 64 TCP flows are sharing the bottleneck link (heavy congestion), where the congestion control algorithm for this scenario was TCP New Reno.

These AQMs have two key parameters "Target Delay" and "Interval" that affect their performance. The defaults values of these AQMs parameters based on the Linux implementation are shown in Table 6.1. And we kept the default values for the remaining adjustable parameters.

Parameter \ AQM	RED	Codel	PIE
T_{delay}	5ms	5ms	20ms
Interval	500ms	100ms	30ms

TABLE 6.1: Default parameter values

We conducted a first set of NS-2 simulations and observed how the regarded mechanisms behave when varying their parameter settings.

6.4.1 Target Delay

We considered five target delay values (1, 5, 10, 20 and 30ms), where this range includes the default value for PIE and CoDel (20ms and 5ms, respectively). For these tests, we kept for each AQM the default value of the interval parameter.

In Figure 6.5 , we report the boxplots of RTTs for the 3 AQMs for each target delay value. The black boxplots are results for "ARED+*Gentle*-RED". For each value of the x-axis, the order of AQMs boxplots is that of the legend. As the base RTT is set to 100ms, the difference between values on boxplots and 100ms correspond to queuing delay at the bottleneck.

A first observation is that there is no difference between ARED and "ARED+*Gentle*-RED" results.

We observe that median and percentiles of queuing delay decrease proportionally to the decrease of target delay for all AQMs. This correlation was less for Ros results.

It is also observable that CoDel and ARED show longer queuing delay distribution tail, meaning larger fluctuations in RTTs. In contrast, Ros experiments show that ARED exhibits much shorter distribution tails, meaning more controlled queuing delay.

The last thing is that the queuing delays of NS-2 simulations are smaller than the Ros experimental results.

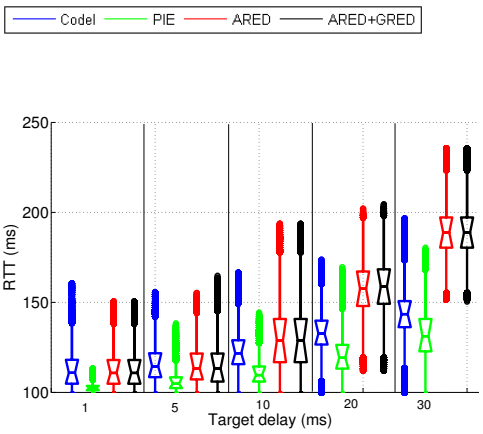

FIGURE 6.5: RTT, NS-2 simulation results

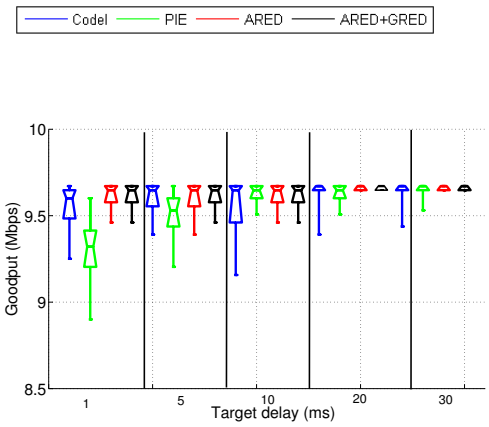

FIGURE 6.6: Goodput(Mbps), testbed experimental results

Figure 6.6 shows the achieved goodput of AQMs for different Target Delay values. We observe that the goodput values for all AQMs are always above 9Mbps, but compared to Ros results, there is more variation for Codel values.

6.4.2 Interval Time

The default interval value for Codel, PIE and ARED, are 100ms, 30ms and 500ms, respectively. We only change CoDel and PIE interval value, like in Ros study. We consider 3 Interval values: small (5ms), medium (30ms) and large (100ms) relative to the base RTT of 100ms. This range of values incorporates CoDel's and PIE's default Interval values (100 ms and 30ms, respectively).

Figure 6.7 shows the variation of delays for Codel, when varying Interval parameter. We observe that smaller values than the default one for Codel (100ms) leads to an improvement in queuing delays. These results are consistent with those presented by Ros. But simulation values remain low compared to their experimental results.

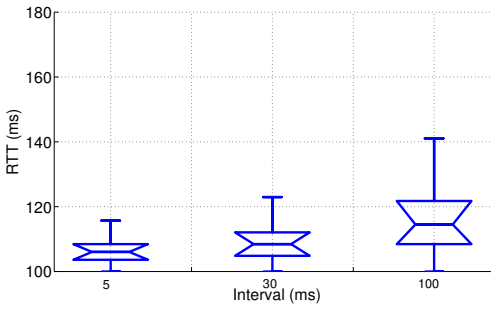

FIGURE 6.7: RTT (ms), Codel NS-2 simulation results

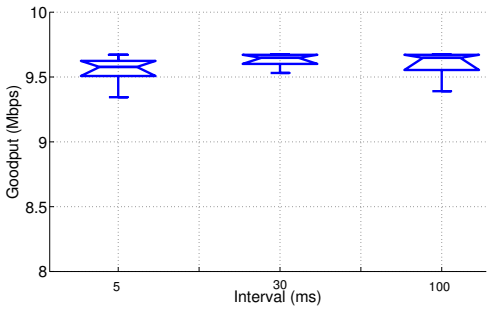

FIGURE 6.8: Goodput (Mbps), Codel NS-2 simulation results

We present in Figure 6.8 Codel achieved goodput. We can see that the variation of the interval value does not impact the goodput for Codel, that is always close to 9.6 Mbps.

Figure 6.9 shows that PIE achieves better queuing delays with smaller interval values. When comparing with Ros results we found that for simulations the delays remain lower than 120ms, while these values reach 150ms for Ros experiments. But the achieved goodput for PIE increases slightly with high interval values for both simulations and experiments, see Figure 6.10.

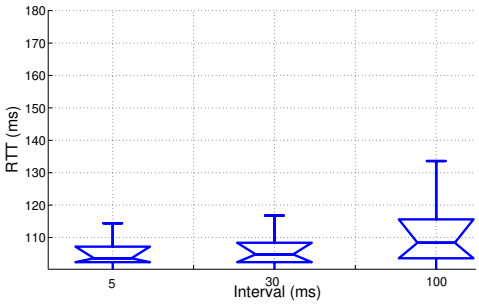

FIGURE 6.9: RTT (ms), PIE NS-2 simulation results

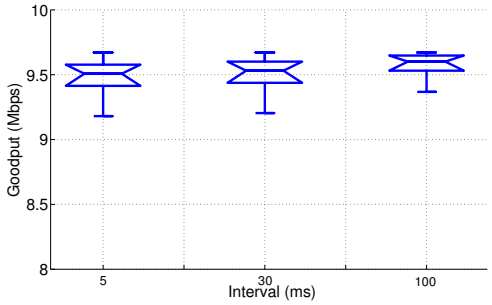

FIGURE 6.10: Goodput (Mbps), PIE NS-2 simulation results

6.4.3 Base RTT

The third parameter we tested is the base RTT. The default Interval value for CoDel in these experiments is equal to 100ms, which allows it to absorb a burst size of an interval worth of data, making it suitable for scenarios when RTT is reasonably close to 100ms.

We provide evaluation results for three values of RTTs:

- Short RTTs 5ms (intra-city/state transfers).
- Medium RTTs 100ms (continental and most inter-continental internet paths).
- Long RTTs 500ms (for few inter-continental paths, some developing countries and satellite links).

We report in Figures 6.11 and 6.12 the per-packet queuing delay and the achieved goodput for different RTT values.

We observe that with most AQMs, better queuing delay were achieved when RTT is medium while the goodput level deteriorates slightly. PIE is able to achieve better queuing delay than CoDel and ARED with almost the same goodput level when RTT is equal to 100ms. PIE undergoes small degradation in terms of goodput for larger RTTs. We note that even if ARED does not manage well the delays, it offers significantly better

goodput values in all cases.

The goal of the previous tests was to first compare the performance of these AQMs before assessing their impact on synchronization. The above results of NS-2 simulations are slightly different from the ones obtained by Ros using experimentation in a testbed.

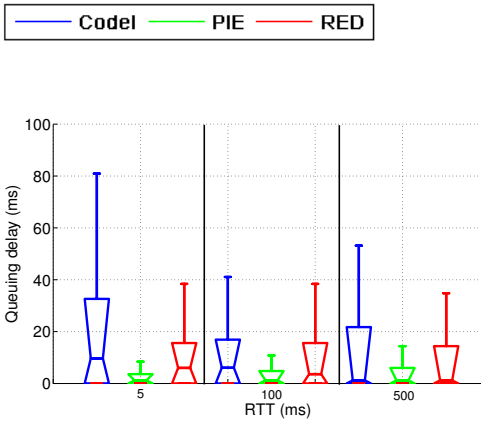

FIGURE 6.11: Queuing delay (ms), NS-2 simulation results

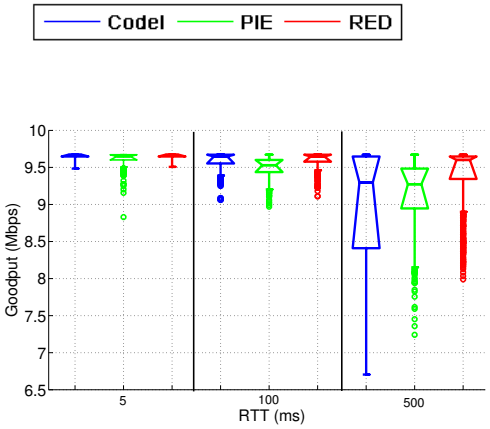

FIGURE 6.12: Goodput (Mbps), NS-2 simulation results

The reason behind this difference between NS-2 and experimental results may be caused by: (i) the implementation of some Active Queue Management (AQM) algorithms in the NS-2 simulator that is different from the one used in the netem modules implemented in the Linux kernel; and (ii) the configuration of the simulations does not catch the complexity of an operational IP network.

To move forward, we decided to :

1. Perform the same tests with experiments, in our testbed.
2. Implement LinCubic and AccuCubic as modules in Linux kernel
3. Explore how AQMs can be merged with LinCubic and AccuCubic

6.5 Experimental results

The only study that compares Codel, PIE and ARED against each other through experimentation is the one by Ros [NK13]. We observe differences between the NS-2 results that we found and Ros experimentation results. We thus conducted an evaluation of these AQMs through experimentations in a testbed in our laboratory.

6.5.1 Experimental setup

In order to perform tests of realistic network scenarios in a controlled manner, we need real network devices as well as network emulation capabilities. One of the most

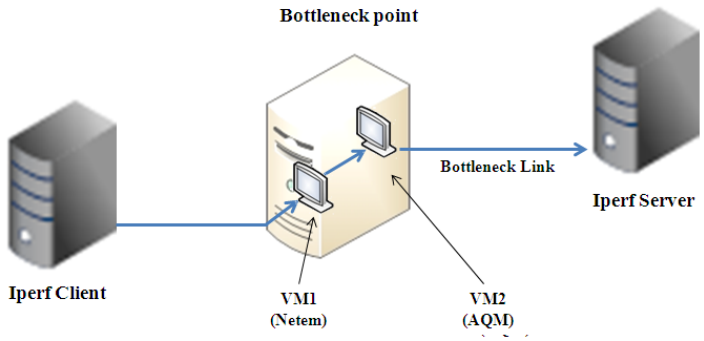

FIGURE 6.13: Experimental network setup

popular network emulators in the research world is NetEm [Fou09]. This free open source tool is widely used in different kinds of testbeds. The current version of Netem emulates variable delay, loss, duplication and re-ordering.

Based on [HJ13] and [NK13], if using Netem to introduce latency, we should use a separate middlebox. In particular, Netem does not work in combination with other queuing disciplines qdiscs [2], also called network schedulers that are commonly used as attempts to compensate for various networking conditions. We considered the same testbed that we used in the previous chapter, with a slight modification that allows us to use Netem and AQMs, separately.

Our wired testbed comprises 3 multi-core Dell servers, 2 acting as TCP client or server and one as bottleneck node, see Figure 6.13. We created 2 virtual machines VM1 and VM2 in the intermediate machine. The first VM uses Netem to control the path latency and the capacity, and the second VM uses the AQM to manage the buffer size. We use the default FIFO/droptail as server scheduling/queue management policy at VM1.

6.5.2 Ros comparison

We performed some of Ros experimentations using our testbed, where the base RTT is equal to 100ms, the queue size is set to 1000 packets, the bottleneck link capacity is set to 10Mbps and the number of long-lived flows sharing the bottleneck is equal to 16 instead of 64.

As a first step, we considered the same target delay values (1, 5, 10, 20 and 30ms). We report in Figure 6.14 the boxplots of RTTs for the 3 AQMs.

[2] http://en.wikipedia.org/wiki/Network_scheduler

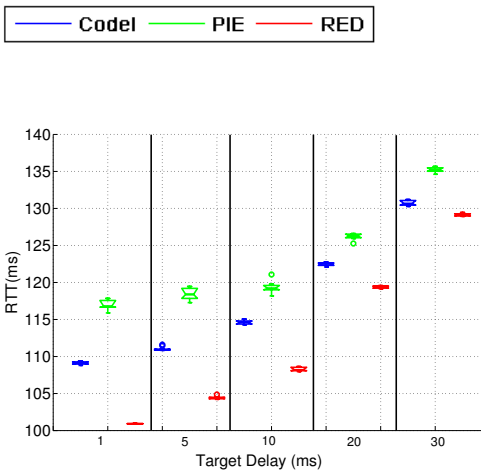

FIGURE 6.14: RTT(ms), 10Mbps, RTT=100ms, BS=1000 packets

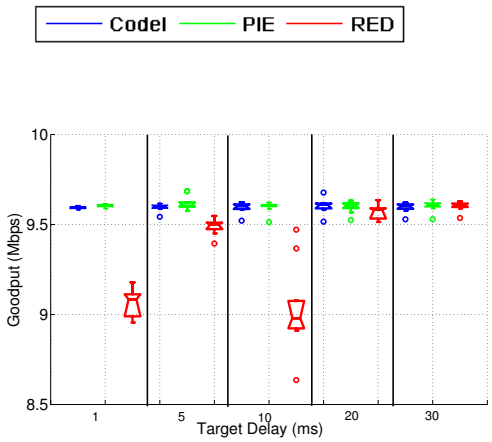

FIGURE 6.15: Goodput(Mbps) , 10Mbps, RTT=100ms, BS=1000 packets

We observe that with experimental results all queuing delays increase with target delay. ARED offers smaller queuing delay than Codel and PIE. It better controls the latency like in Ros results. As for the achieved goodput, it always stays above 9Mbps for all AQMs. We observe that ARED achieves larger goodput value for all target delay values compared to Codel and PIE.

We can see that compared to AQMs simulation results, the results of our experimentations seem qualitatively closer to Ros results, but quantitatively there is a slight difference. This may be due to :

- The use of virtualization in our testbed.
- The differences between our topology and the one in [NK13], are :
  - Dumbbell topology with 4 sender-receiver pairs.
  - Two sets of Dell OptiPlex GX620 machines acting as senders and receivers.
  - The senders and receivers are connected via two routers; the first router is acting as an AQM router implementing AQM on its bottleneck (egress) interface. The second router is acting as a delay node on both forward and reverse traffic, using the ipfw dummynet module (http://info.iet.unipi.it/ luigi/dummynet/).

  In our testbed, we have 3 machines, one sender, one receiver and a router on which we have 2 VMs.

The above results comparing the AQMs performance show that ARED better controls the latency. As next step, we will proceed to study the AQMs performance for high BDP scenarios, where synchronization was highly pronounced.

6.5.3 Experiments: Synchronization vs. AQMs

As synchronization was high for TCP Cubic flows, we tested the potential benefit of AQMs with scenarios where a large number of TCP Cubic flows share the bottleneck link. Then, we implemented LinCubic and AccuCubic as kernel modules and we compare AQMs performance with the potential benefit of these modifications that we proposed and detailed in the previous chapter. We considered the case of 100 flows competing for the bottleneck, where Fast Convergence is turned ON.

We consider a link capacity equal to 100Mbps, a latency equal to 350ms and a buffer size equal to one 1 BDP (i.e., 2916 packets). For each scenario, we performed 10 runs. We report the time series of the congestion window and buffer, the goodput and the number of synchronized flows.

Figures 6.16 and 6.17 show the time series of the congestion windows for our approches and the 3 AQMs and their CDFs. We first observe that the congestion window for Cubic, LinCubic and AccuCubic reaches the value of BDP + BS (2916×2) when the buffer management is Drop Tail (DT). In contrast, with AQMs, the congestion windows remain near 1 BDP, which means that the number of packets in the buffer is small, see Figure 6.18. So AQMs mechanisms inteligently manage the queue of packets to have low delays.

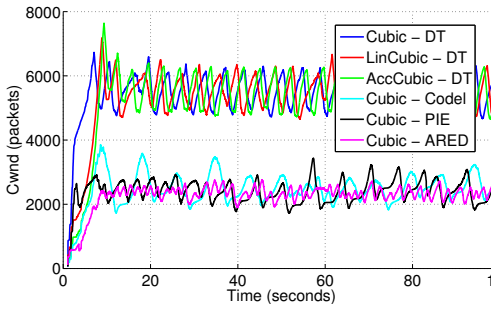

FIGURE 6.16: Cwnd(packets), 100Mbps, RTT=350ms, BS=1BDP

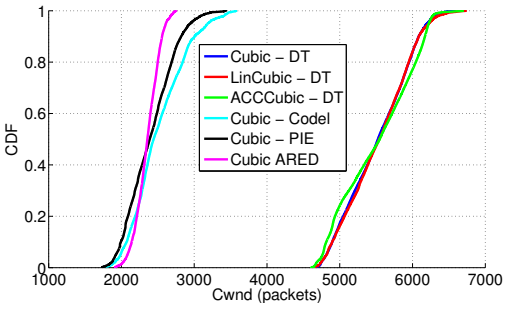

FIGURE 6.17: CDFs of the congestion window, 100Mbps, RTT=350ms, BS=1BDP

We report in Figure 6.19 the goodput results. Since there are 6 quantities (approaches + AQMs), we use one boxplot per quantity. We can observe that through the use of buffer management mechanisms like Codel, PIE and ARED, we can reduce delays while loosing little goodput.

We further report the number of synchronized flows in Figure 6.20, and their CDFs in Figure 6.21. Experimental results are consistent with simulation results : LinCubic decreases the number of synchronized flows as well as AccuCubic when FC is turned on. But, we can see that those improvements are mitigated by the competition among TCP Cubic flows.

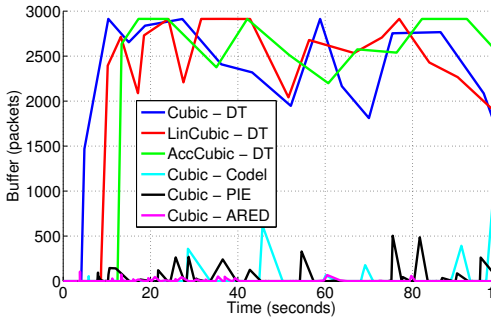

FIGURE 6.18: Buffer(packets) , 100Mbps, RTT=350ms, BS=1BDP

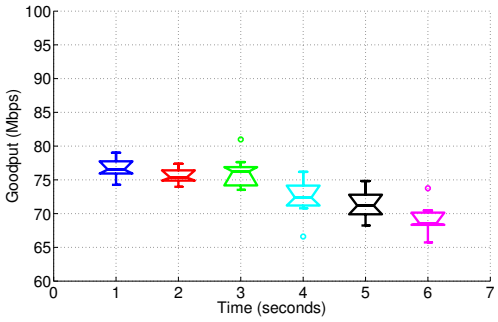

FIGURE 6.19: Goodput(Mbps), 100Mbps, RTT=350ms, BS=1BDP

When comparing the number of synchronized flows for the AQMs, we observe that ARED seems to be the most effective to reduce the level of synchronization between TCP Cubic flows. These results are consistent with the simulation results in [HR09] where the authors have shown that for large buffers, ARED is able to reduce significantly the synchronization between flows. Also, compared to Cubic, we manage to reduce this amount of synchronized flows to 41 flows for ARED instead of 81 flows for Cubic.

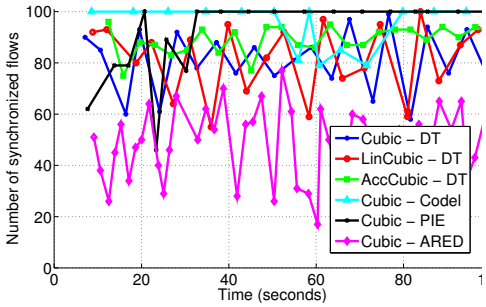

FIGURE 6.20: Number of synchronized flows, 100Mbps, RTT=350ms, BS=1BDP

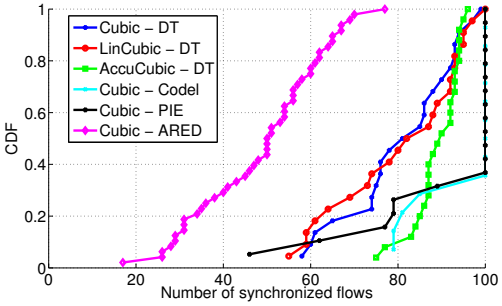

FIGURE 6.21: CDFs of the synchronized flows, 100Mbps, RTT=350ms, BS=1BDP

High synchronization between TCP Cubic flows when we apply AQMs on the router can be explained by the concurrency between TCP Cubic flows (more details in the previous chapter). So we proceeded to do the same test but with TCP New Reno for which synchronization was less important. We investigated whether with AQMs we still have lower rates.

We report in Figure 6.22, the time series for the window size for TCP New Reno whith Droptail, Codel, PIE and ARED queue management algorithms. With the use of AQMs, delays decrease but also the goodput undergoes degradation, see Figure 6.23.

Figure 6.24, presents the number of synchronized flows. We note that with AQMs, synchronization is higher compared to the case of droptail. The increase of the number of synchronized flows increases the packets loss rate. This increase in packet loss was

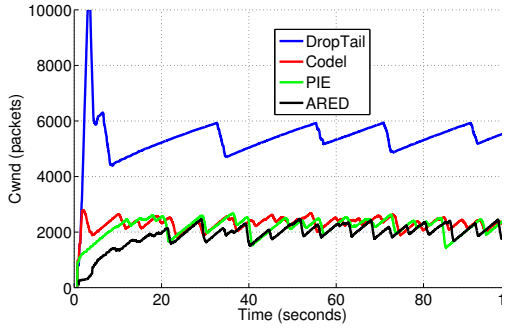

FIGURE 6.22: Cwnd (packets), TCP New Reno 100Mbps, RTT=350ms, BS=1BDP

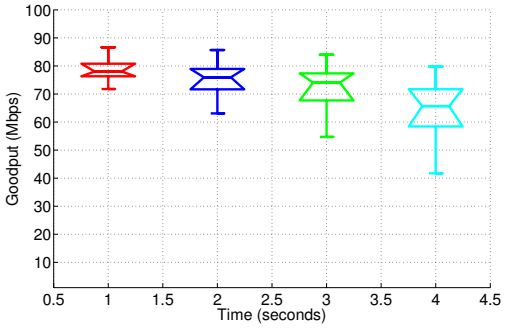

FIGURE 6.23: Goodput(Mbps), TCP New Reno 100Mbps, RTT=350ms, BS=1BDP

noticed in [GKF13], where authors show that through simulations PIE and CoDel both increase packet loss.

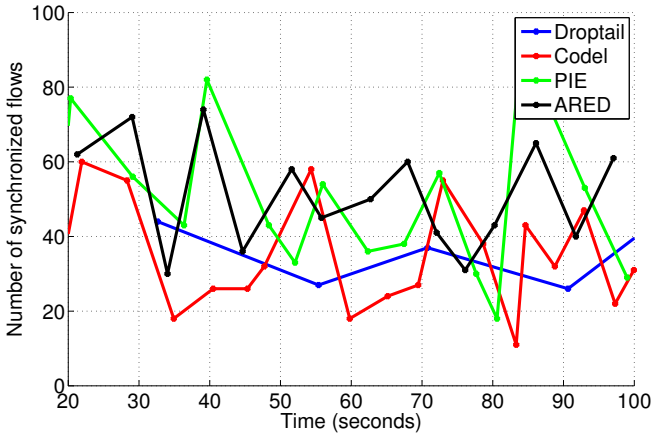

FIGURE 6.24: Number of synchronized flows, TCP New Reno 100Mbps, RTT=350ms, BS=1BDP

6.5.4 Combining LinCubic and AccuCubic with AQMs

To evaluate the impact of those different modifications when combined with AQMs, we considered scenarios where LinCubic and Accucubic are combined with AQMs and we started observing their behavior. We report in Figures 6.25, 6.26 and 6.27 the time series evolution of the window size for Cubic, LinCubic and AccuCubic when combined with Codel, PIE and ARED, respectively.

We observe that the Cubic shape of the total windows persists, meaning that all flows are still synchronized. So, through this combination we can improve the situation partially. We can see also that with default parameters of AQMs, the delays are reduced, but this causes often empty buffers, so an under-utilization of the link.

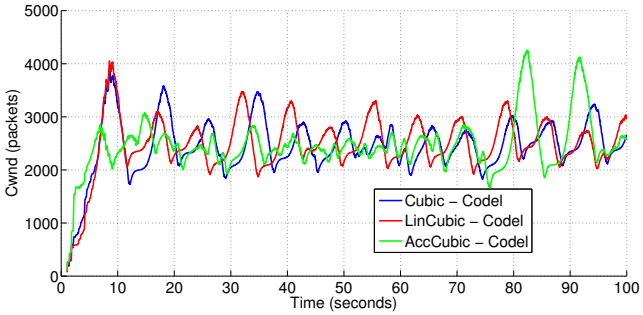

FIGURE 6.25: Cwnd(packets), Codel, 100Mbps, RTT=350ms, BS=1BDP

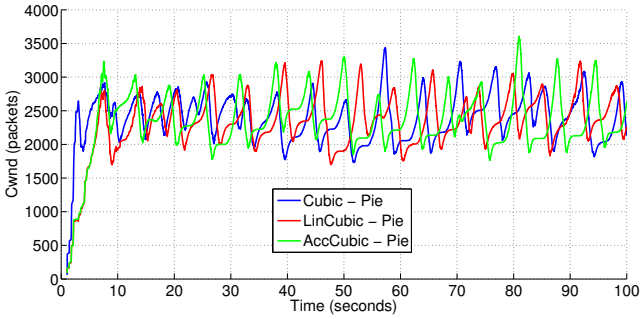

FIGURE 6.26: Cwnd(packets) , PIE, 100Mbps, RTT=350ms, BS=1BDP

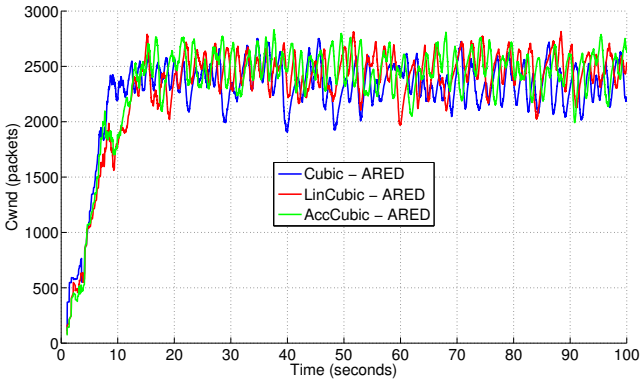

FIGURE 6.27: Cwnd(packets) , ARED, 100Mbps, RTT=350ms, BS=1BDP

So, given these results and also given that the default parameters of AQMs are small compared to the base RTT (350ms) that we have considered for these experiments, we proceeded by changing the Target delay and Interval parameters and see their impact on the link utilization and the goodput.

6.5.5 Parameter Sensitivity

We now want to assess how the AQMs mechanisms behave under varying the target delay and the interval parameters. We considered various target delay values ranging from 20 to 150ms and we reverted to use only TCP Cubic in these experiments. From Figure 6.28 we can make several observations: All AQMs's median stay around 350ms except the case of Codel with Target delay value equal to 150ms. But we still observe a set of values which are far from the median.

Figure 6.29 shows the achieved goodput of AQMs for Target delay values. We note that the rate is slightly increased with the increase of Target delay values, which is consistent with the fact that the larger the target values are, the better both the goodput and the utilization are.

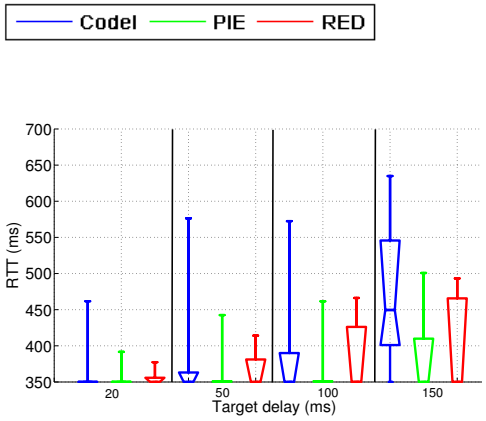

FIGURE 6.28: RTT (ms) - 100Mbps, 350ms - Cubic

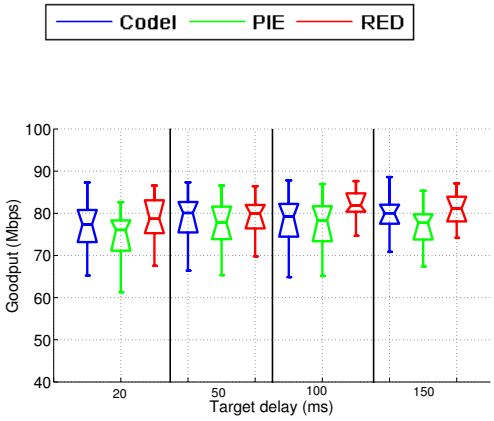

FIGURE 6.29: Goodput (Mbps)- 100Mbps, 350ms - Cubic

The second important parameter for the AQMs is the Interval. We consider three time-granularities at which AQMs might perform : 100ms, 200ms and 300ms. Figures 6.30 and 6.32 show Codel's and PIE's performance when interval is set to the above values.

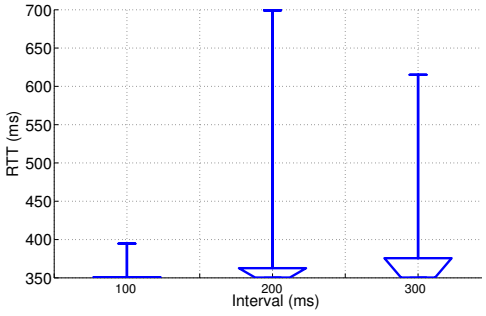

FIGURE 6.30: RTT (ms) - 100Mbps, 350ms - Codel

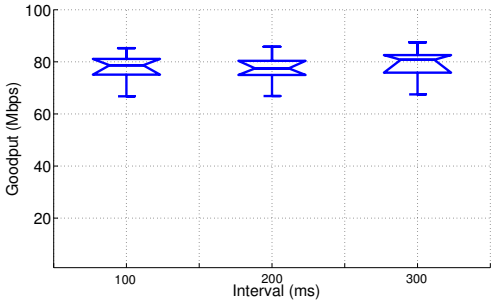

FIGURE 6.31: Goodput (Mbps)- 100Mbps, 350ms - Codel

These two AQMs achieve better queuing delays with fine-grained intervals (100ms) and slightly better goodput with the large intervals (300ms).

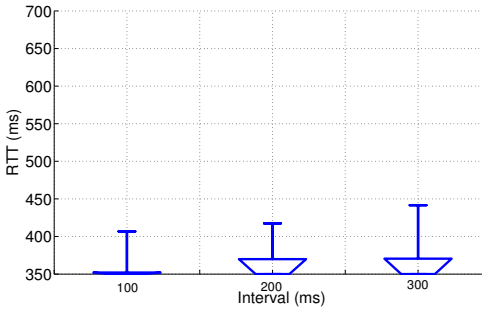

FIGURE 6.32: RTT (ms) - 100Mbps, 350ms - PIE

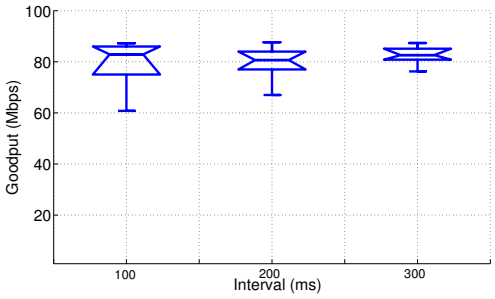

FIGURE 6.33: Goodput (Mbps)- 100Mbps, 350ms - PIE

6.6 Conclusion

In this chapter, we investigated how can we reduce the synchronization level through network level solutions based on several popular queue management mechanisms, PIE, Codel and ARED. Then we provided results when combining these AQMs with our approaches LinCubic and AccuCubic. AQM algorithms require setting Target delay and Interval parameters but can significantly reduce the level of synchronization, while ensuring small delays and a high link utilization. This is true for Cubic, but not for New Reno where AQMs can in fact slightly increase the level of congestion.

We provide hints that through the use of AQMs we may have smaller delays and reduce synchronization between flows, but there is a price to pay : a higher packet loss and goodput degradation.

We also observed that using AQMs for reducing synchronization was more efficient than the end host solutions that we proposed, namely AccuCubic and LinCubic. This is not a surprise per se, and it was one of the main objective of this study to quantify the efficiency of these two approaches to reduce the level of synchronization in TCP Cubic.

Conclusions and perspectives

Cloud computing enables a flexible access to computation and storage services. This requires, for the cloud service provider, mastering network and system issues.

The success of the Internet can be attributed in part to the congestion control mechanisms implemented in TCP. TCP has evolved to monitor network conditions and trends to be robust and scalable. However, the performance of TCP in data center networks has been a major concern recently because it was observed to lead to problems. It thus becomes important to investigate the performance of TCP in typical cloud environment.

In order to master network and system issues in cloud environments, we focused in this thesis on the performance of TCP Cubic, the default version of TCP in Linux and the most widely used in today's data centers.

Conclusions

In **Chapter 1**, we motivated our work by briefly reviewing some impairments in Data Center Networks and the variants proposed so far to overcome these impairments. Then, we made a brief history of some TCP protocols, a survey of some congestion control algorithms, and relevant works related to the topic of this thesis. Afterwards, in **Chapter 2**, we proposed an analytical model for a single-long lived TCP Cubic connection. Using this model, we have analyzed the window size, showing how this metric depends on the different modes of TCP Cubic and the buffer size. This model is actually used by people from Orange Labs for troubleshooting TCP connections.

A cloud computing environment, is characterized by an increasing number of competing flows for a single bottleneck link. We have derived in **Chapter 3** a fluid model for TCP Cubic that allows to predict the values of various metrics when a large number of connections are sharing a common bottleneck link, such as: (1) distribution of the window sizes; (2) throughput; (3) RTT; (4) loss rate; and (5) queue size. We exhibit that the fit is very good for the intra-DC and FTTH scenarios, while it is less good for the inter-DC scenario. In this last mode, TCP Cubic operates in Cubic mode and causes loss synchronization amongst the connections.

In **Chapter 4**, we have analyzed measurement from/to a cloud solution available internally at Orange, Cube. Several tests have been made between Sophia and Cube, in both directions. Thanks to the various scenarios made, we could distinguish the different characteristics of connections to/from Cube. All results indicated that there were several roots to the problem: the disk access, the virtualization layer and a shaper. These different issues enabled us to underscore the interplay between system and network issues in a typical cloud solution.

In **Chapter 5**, we have explored in detail the root causes behind the synchronization of TCP Cubic flows that can be easily observed through simulations for instance. Through the combination of experiments in a testbed, simulations and some experiments in the wild (with Amazon EC2 datacenter), we analyzed the extent of the phenomenon. The controlled nature of our testbed enabled us to precisely analyze the phenomenon of synchronization and discover its root causes. Simple experiments with a distant EC2 datacenter confirmed that the phenomenon can affect real world transfers. We proposed and evaluated two modifications to the TCP Cubic algorithm that aim at combating synchronization. These two approaches improved noticeably the situation. We further investigated in **Chapter 6**, how to combat the synchronization through network level solutions based on several popular queue management mechanisms like PIE, Codel and ARED. AQM algorithms require setting Target delay and Interval parameters but can significantly reduce the level of synchronization, while ensuring small delays and a high link utilization. Through the use of AQMs we may have smaller delays and reduce synchronization between flows, but there is a price to pay : a higher packet loss and goodput degradation. We have also shown that using AQMs for reducing synchronization was more efficient than the end host solutions that we proposed, namely AccuCubic and LinCubic.

Finally, we highlight several interesting perspectives and future research of our work.

Perspectives

As perspectives, it would be of first interest to consider the extension of the analytical model to have a more general model, encompassing the slow-start phase, and mixing TCP New Reno and TCP Cubic connections, to assess more general scenarios about these TCP versions.

Second, it would be interesting to extend this model by adding active queue management mechanisms like RED instead of DropTail. This would enable us to investigate the interaction between Transmission Control Protocol (TCP) and Active Queue Management (AQM) mechanism.

Third, in our work, we only considered flows with the same RTTs, but in reality, multiple

flows with various RTTs exist, and they are dynamically generated and terminated. It is anticipated that these factors strongly influence the dynamics, so it would meaningful to investigate these scenarios with our model by adding these details where multiple flows with different RTTs compete the bottleneck link.

Furthermore, we want to make public releases of the two approaches LinCubic and AccuCubic.

We also want to explore data center scenarios with a high dynamics in the number of flows and especially a competition between short and long flows. Due to the noise induced by short flows, long flows are likely to underestimate the network capacity, which, as we have seen, can lead to too many packets sent when reaching the actual capacity, and thus possibly, synchronization.

Through simulations and experiments that we performed for AQM mechanisms for the same scenario performed in [NK13], we observe that the results of experimentations in I3S testbed and NS-2 simulations seem qualitatively and quantitatively different. So, another perspective consists in the study of the specifications of these AQMs in NS-2 and the one implemented for the Kernel. Once simulations and implementations would be reconcile, it would allow us to assess with more accuracy the interplay between TCP Cubic and AQMs.

Appendix A

Computing parameters for a single TCP Cubic flow model

A.1 Transitions between states

A.1.1 State (A), TCP mode, Q(t)=0

The state (A) may be the initial state of the system. In this case W(t) is as follows (from Equation 2.5):

$$W(t) = w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)}\frac{t}{\tau}$$

The system leaves this state to state (B) (Cubic mode) when Q(t)=0, and the congestion window $w_c > W_{tcp}$. Or, to state (C) (TCP mode) if Q(t)$>$ 0, $w_c < W_{tcp}$, and $t_{AB} < t_{AC}$.

- Transition from state (A) to state (B)

We need to consider the difference $F(t) = w_c(t) - w_{tcp}(t)$:

$$W_{tcp}(t) - W_{cubic}(t) = w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)}\frac{t}{\tau} - C_{cubic}(t - V_{cubic})^3 - w_{max}$$

The transition occurs when:

$$W_{tcp}(t) - W_{cubic}(t) = 0$$

$$-C_{cubic}(t - V_{cubic})^3 - \beta w_{max} + \frac{3\beta}{b(2-\beta)}\frac{t}{\tau} = 0$$

$$-C_{cubic}t^3 + C_{cubic}V_{cubic}^3 + 3C_{cubic}t^2V_{cubic} - 3C_{cubic}tV_{cubic}^2 - \beta w_{max} + \frac{3\beta t}{b(2-\beta)\tau} = 0$$

Consider,

$$-C_{cubic}t((t-\frac{3}{2}V_{cubic})^2-\sigma)=0$$

Where $\sigma=-\frac{3}{4}V_{cubic}^2+\frac{3\beta}{b(2-\beta)C_{cubic}\tau}$

- if $\sigma<0$, there is only one solution t=0

- if $\sigma>0$,

$W_{tcp}(t)-W_{cubic}(t)=-C_{cubic}t(t-\frac{3}{2}V_{cubic}-\sqrt{\sigma})\times(t-\frac{3}{2}V_{cubic}+\sqrt{\sigma})$

There are three solutions: $t_0=0$, $t_1=\frac{3}{2}V_{cubic}+\sqrt{\sigma}$, and $t_2=\frac{3}{2}V_{cubic}-\sqrt{\sigma}$
Since $t_2<t_1$ so if $t_2<0$, choose t_1, else choose t_2

- Transition from state (A) to case (C)

At $t=t_{AC}$, $W_{tcp}(t_{AC})=C\tau$

$$w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)}\frac{t_{AC}}{\tau}=C\tau$$

So $t_{AC}=\frac{(2-\beta)b\tau}{3\beta}(\tau C-(1-\beta)w_{max})$

A.1.2 State (B), Cubic mode, Q(t)=0

The window size is :

$$W(t)=C_{cubic}(t-V_{cubic})^3+wmax$$

- Transition from state (B) to state (D)

At $t=t_{BD}$ $W_{cubic}(t_{BD})=C\tau$

$$C_{cubic}(t_{BD}-V_{cubic})^3+w_{max}=C\tau$$

So :$t_{BD}=\sqrt[3]{\frac{(C\tau-w_{max})}{C_{cubic}}}+V_{cubic}$

- Transition from state (B) to state (A)

At $t=t_{BA}$, $W_{cubic}(t_{BA})=W_{tcp}(t_{BA})$

$$W_{cubic}(t_{BA})-W_{tcp}(t_{BA})=0$$

$$C_{cubic}(t_{BA}-V_{cubic})^3+w_{max}-w_{max}(1-\beta)+\frac{3\beta}{b(2-\beta)}\frac{t_{BA}}{\tau}=0$$

So, to find t_{BA} value we use the same reasoning than section A.1.1

A.1.3 State (C),TCP mode, Q(t)>0

According to Equation 2.11, the window size W_{tcp} is:

$$W_{tcp}(t) = \frac{(w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)})}{2} + \frac{1}{2} \times ((\frac{3\beta}{b(2-\beta)} + w_{max}(1-\beta))^2 + 4(\frac{3\beta Ct}{b(2-\beta)} - \frac{3\beta w_{max}(1-\beta)}{b(2-\beta)}))^{\frac{1}{2}}$$

The system leaves this state to state (D) (Cubic Mode) if $w_c > W_{tcp}$, or to state (E) (Loss recovery) if the connection observes a loss.

- Transition from state (C) to state (D)

At $t = t_{CD}$, $W_{tcp}(t_{CD}) = W_{cubic}(t_{CD})$ So, to find t_{CD} value we use the simple expression of w_{tcp} and we considered the same reasoning than section A.1.1

- Transition from state (C) to state (E)

At $t = t_{BE}$, $w_c(t_{BE}) = C\tau + B + 1$

$$\frac{(w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)})}{2} + \frac{1}{2} \times ((\frac{3\beta}{b(2-\beta)} + w_{max}(1-\beta))^2 + 4(\frac{3\beta C t_{CD}}{b(2-\beta)} - \frac{3\beta w_{max}(1-\beta)}{b(2-\beta)}))^{\frac{1}{2}} = C\tau + B + 1$$

So $t_{CE} = \frac{b(2-\beta)}{3\beta C}(\frac{1}{4}(\frac{b(2-\beta C)}{3\beta C} \times 4(C\tau + B + 1 - \frac{(w_{max}(1-\beta) + \frac{3\beta}{b(2-\beta)})^2}{2})^2 - ((w_{max}(1-\beta) + \frac{3\beta}{3\beta C})^2) + \frac{3\beta w_{max}(1-\beta)}{b(2-\beta)})$

A.1.4 State (D), Cubic mode, Q(t)>0

The window size is:

$$W_{cubic}(t) = C_{cubic}(t - V_{cubic})^3 + w_{max}$$

- Transition from state (D) to state (C)

At $t = t_{DC}$, $W_{cubic}(t_{DC}) = W_{tcp}(t_{DC})$

$$C_{cubic}(t_{DC} - V_{cubic})^3 + w_{max} = \frac{(w_{max}(1-\beta))^2}{2} + \frac{1}{2} \times ((w_{max}(1-\beta))^2 + \frac{12\beta C}{b(2-\beta)} \times t_{DC})^{\frac{1}{2}}$$

So, to find t_{DC} value we use the simple expression of w_{tcp} and we considered the same reasoning than section A.1.1

- Transition from state (D) to state (E)

At $t = t_{DE}$, $w_c(t_{DE}) = C\tau + B + 1$
So: $t_{DE} = \sqrt[3]{\frac{C\tau+B+1-w_{max}}{C_{cubic}}} + V_{cubic}$

A.2 Sequence number

$S(t_k)$ and $A(t_k)$ are the sequence number and the acknowledgment number at time t_k. Assume t_0 is the starting time of a phase (more details about phases are given in the Section 2.3.1.1), and $t_{k+1} = t_k + R(t_k)$. We differentiate two cases according to the occupation of the buffer:

- Empty buffer Q(t)=0

The window size W(t)$\leq C\tau$, so : $t_{k+1} = t_k + \tau$ and

$$t_k = k\tau + t_0 \tag{A.1}$$

- Nom empty buffer Q(t)>0

The window size W(t)$> C\tau$, so : $t_{k+1} = t_k + \frac{W(t_{k+1})}{C}$ and $t_k = t_0 + \frac{1}{C}\sum_{i=1}^{k} W(t_i)$

We consider a greedy source (a TCP source that sends data as soon as it has the opportunity, i.e., as soon as an ACK is received or the window increases), where W(t)=S(t)-A(t)

$$\begin{aligned} W(t_{k+1}) &= S(t_{k+1}) - A(t_{k+1}) \\ &= S(t_{k+1}) - S(t_k) \end{aligned}$$

So

$$S(t_k) - S(t_0) = \sum_{i=1}^{k} W(t_i)$$

and $\forall k, t \in \{t_k, t_{k+1}\}$

$$W(t_k) \leq W(t) \leq W(t_{k+1})$$

We integrate the above equation between t_{k-1} and t_k :

$$\int_{t_{k-1}}^{t_k} W(t_k)dt \leq \int_{t_{k-1}}^{t_k} W(t)dt \leq \int_{t_{k-1}}^{t_k} W(t_{k+1})dt \leq \int_{t_k}^{t_{k+1}} W(t)dt$$

$$\int_{t_{k-1}}^{t_k} W(t)dt \leq (t_k - t_{k-1})W(t_k) \leq \int_{t_k}^{t_{k+1}} W(t)dt \tag{A.2}$$

- if Q(t)=0 (i.e., empty buffer): $t_k - t_{k-1} = \tau$, and Equation A.2 becomes:

$$\int_{t_0}^{t_k} W(t)dt \leq \tau \sum_{i=1}^{k} W(t_i) \leq \int_{1}^{t_{k+1}} W(t)dt$$

$$\int_{0}^{t_k} \frac{W(t)}{\tau}dt \leq S(t_k) - S(t_0) \leq \int_{1}^{t_{k+1}} \frac{W(t)}{\tau}dt$$

$$C(t_k - t_0) \leq S(t_k) - S(t_0) \leq C(t_{k+1} - t_1)$$

$$C(t_k - t_0) \leq S(t_k) - S(t_0) \leq C(t_{k+1} - t_1) \tag{A.3}$$

- if Q(t)> 0 (i.e., non empty buffer): $t_k - t_{k-1} = \frac{W(t_k)}{C}$

$$S(t_k) - S(t_0) = C(t_k - t_0)$$

Given that the system is conservative: $\forall\ t$

$$S(t) = S(t_0) + C(t - t_0) \tag{A.4}$$

A.3 Quality of Service metrics

Based on n_x, t_x quantities, we calculate some metrics that can be used in an evaluation of the developed model for TCP Cubic [Flo08].

- Packet loss rate: The total number of packets transmitted by a source during one cycle t_c is n_{t_c}. Knowing that a single packet is lost per cycle, the loss rate will be as follows :

$$\varepsilon_p = \frac{1}{n_{t_c}} \tag{A.5}$$

- Throughput: The average throughput during one cycle is as follows :

$$\overline{th} = \frac{n_{t_c}}{t_c} \tag{A.6}$$

- Mean buffer occupancy: The mean buffer occupancy is equal to the number of packets in the buffer during a cycle.

$$\overline{Q(t_c)} = \int_{0}^{t_c} (W(t) - C\tau)^{+} dt \tag{A.7}$$

- Average packet delay at the bottleneck link:

$$\overline{D} = \frac{\overline{Q(t_c)}}{C} \tag{A.8}$$

- Average RTT

$$\overline{R(t_c)} = \tau + \frac{\overline{Q(t_c)}}{C} \tag{A.9}$$

Appendix B

Résumé de la thèse

B.1 Introduction

Contexte et motivation

Le protocole de contrôle de transmissions (TCP) [CK05] est le principal protocole utilisé dans les réseaux Internet Protocol (IP). TCP est particulièrement utilisé dans le téléchargement de page WEB, le HTTP streaming et le peer to peer ce qui représente un large volume de transfert. D'après [JD04] plus que 90% du trafic internet utilise le protocole TCP. La qualité du service (QoS) du point de vue du consommateur (du service) reflète les performances, la stabilité et la robustesse de TCP dans les réseaux de centre de données. Ce dernier peut être utilisé comme métrique pour évaluer la qualité du service (QoS) du point de vue du consommateur.

Pour répondre aux exigences changeantes des applications dans Internet, diverses stratégies de contrôle de congestion TCP ont été conçues. Parmi elles on peut citer, Bic-TCP [XHR04a], TCP Cubic [HRX08] et Compound TCP [TSZS06]. Bic-TCP et TCP Cubic ont été conçus spécifiquement pour les larges produits délai bande passante. Le noyaux de Linux implémente et utilise par défaut TCP Cubic depuis la version 2.6.19, et c'est la version la plus utilisée de TCP de nos jours [YLX+11]. Il se caractérise par une fonction de croissance cubique de la fenêtre.

Le but de TCP Cubic est de parvenir à une répartition équitable de la bande passante entre les flux avec différents RTT en utilisant un taux de croissance de fenêtre indépendant du RTT. Au lieu de cela, il utilise une fonction du temps écoulé depuis le dernier événement de perte.

Quand il n'est pas utilisé dans un environement de production à large bande passante, TCP Cubic dispose d'un mode qui imite TCP New Reno [PFTK00], sans lui être identique.

L'étude de TCP Cubic (et d'autres version TCP haut débit) se prête bien á l'utilisent de modèles analytiques qui permettent la prédiction des performances et l'étude de l'impact de différents paramètres.

En effet, l'utilisation d'environnement expérimental ne permet pas assez de flexibilité. On est limité par une plage de possibilité plus restreinte. Un transfert massif de données est commun dans l'univers du cloud, que ce soit entre les différents serveurs du centre de données ou entre le centre de données et le consommateur du service. Dans ce cas de figure la couche de transport (TCP) est fortement sollicitèe et peut souffrir de problèmes de performance. A titre d'exemple TCP incast est un problème observé lorsque plusieurs périphériques de stockage envoient une grande quantité de donnée à un unique serveur causant la congestion de données au niveau du switch du serveur recevant ses informations [VPS+09a].

Les environnements Cloud sont caractérisés par une large bande passante. Les versions modernes de TCP (TCP Cubic), sont conçues pour fonctionner efficacement dans ces situations. Elles sont en mesure de sonder la bande passante disponible de manière non linéaire, contrairement à TCP New Reno, qui augmente ses fenêtres d'un MSS par RTT en régime stationnaire. Cependant, cette réactivité agressive a un prix. L'équité offerte par les versions modernes de TCP (TCP Cubic) n'est pas aussi élevée que les anciènnes versions TCP [LLS07]. Plusieurs études ont également souligné l'apparition de synchronisation entre les flux TCP Cubic concurrents [HR08]. Cela veut dire que les flux TCP Cubic concurrents pour un goulot d'étranglement tendent à perdre des paquets au même moment et la série temporelle de débit agrégé résultante présente un comportement cubique clair comme si un seul flux était actif.

Certaines études ont lié le problème de synchronisation avec le problème de dimensionnement du tampon. Le dimensionnement du tampon est un aspect clé pour la conception des routeurs et les performances du réseau. Après 20 ans de recherche, aucune étude ne propose une solution complétement satisfaisante au problème de dimensionnement du tampon. La régle empirique du produit bande passante délai (BDP) est encore largement utilisée car elle optimise la charge de la liaison de sortie d'un routeur. Cependant elle ne garantit pas ni le taux de perte, ni le retard subi par les paquets. Sous certaines conditions de trafic, une taille de tampon égale au BDP peut entraîner des problèmes de performance essentiellement liés à un temps très large (à savoir le phénomène de " bufferbloat"). Par conséquent, la communauté scientifique s'est concentrée sur l'étude des mécanismes actifs du tampon afin de résoudre le problème relatif au dimensionnement du tampon. Parmi ceux-ci, il y a des propositions de tampon adaptive et plus fréquemment aujourd'hui les mécanismes de gestion adaptative du tampon (AQM) afin d'assurer une qualité de service aux flux individuels. La détection précoce aléatoire RED [FJ93a] est l'une des premières disciplines AQM. Étant donné qu'il y a

besoin du réglage prudent de ses paramètres pour diverses conditions du réseau pour ces AQMs, la plupart des opérateurs de réseaux ont abondonné l'utilisation du RED [Ada13]. L'algorithme RED adaptif ou RED actif (ARED) [FGS01] se base sur l'observation de la longueur de la file d'attente moyenne afin de décider d'être plus ou moins agressif. Récemment, deux nouveaux AQMs Codel [NJ12b] [NJ12a] et PIE [RPC+13] ont été proposés pour contrôler la latence dans le but de résoudre le problème du "bufferbloat".

Notre travail vise à étudier la performance de TCP Cubic, largement utilisé de nos jours dans les data centres et plus précisément dans un environnement de cloud.

Plan et Contributions

Dans le chapitre 1, nous présentons les concepts et les motivations des travaux liés à cette thèse. Tout d'abord, nous présentons les efforts de recherche afin d'analyser les performances et les problèmes dans les réseaux de centres de données. Plusieurs travaux de recherche ont été faits pour: (i) étudier la nature du trafic dans les centres de données; (ii) identifier et résoudre les problèmes de performance; et (iii) proposer des comparateurs de performance et de coût des fournisseurs de cloud. Ensuite, nous examinons en détail TCP, le protocole de transport dominant dans les réseaux cloud, et les études liées au contrôle de la congestion, le dimensionnement de la mémoire tampon et les mécanismes de gestion de file d'attente active (AQM). En outre, nous présentons les études qui ont permis une évaluation des performances de TCP Cubic, par des simulations et des expériences. Enfin, nous présentons des modèles analytiques pour TCP.

Dans le chapitre 2, nous présentons d'abord un aperçu détaillé de l'algorithme TCP Cubic, puis nous décrivons le modèle analytique que nous avons développé pour étudier les performances d'un long flux TCP Cubic isolé. Grâce à ce modèle, nous révélons les différences entre les spécifications publiées de TCP Cubic et son implementation dans NS-2. Le modèle a été validé par comparaison avec des simulations NS-2.

Dans le chapitre 3, nous visons à développer un modéle analytique pour TCP Cubic afin d'analyser ses performances pour des scénarios cloud où un grand nombre de longs connexions TCP, par exemple, HTTP streaming ou flux de back-up, partagent un lien de goulot d'étranglement. Plus précisément, nous considérons trois scénarios: (i) un scénario au sein du centre de données (DC) avec beaucoup de trafic entre les serveurs physiques (intra-DC scenario); (ii) un scénario entre centres de données où les liens sont utilisés pour synchroniser ou sauvegarder des données (inter DC scenario); et (iii) un scénario de distribution de contenu où un grand nombre de clients haut débit, par

exemple, les clients FTTH, téléchargent simultanément le contenu du centre de données (FTTH scenario). Nos contributions sont les suivants:

- Sur la base d'une approximation de champ moyen, nous dérivons un modèle fluide pour TCP Cubic, qui permet de prédire les performances en termes de plusieurs paramètres. Nous montrons en outre une propriété de passage à l'échelle du modèle fluide, qui permet de l'utiliser pour une variété de scénarios réseau sans coût de calcul prohibitif. Nous validons ce modèle analytique grâce à des simulations NS-2 pour nos scénarios de cloud.

- Nous fournissons une comparaison étendue de TCP Cubic et New Reno pour les scénarios de cloud, l'évaluation de leur compromis efficacité/équité ainsi que l'impact de la taille du tampon sur leurs performances. En particulier, nous démontrons que TCP Cubic surpasse TCP New Reno, même pour les cas où le BDP est faible, ce qui est souvent le cas dans les scénarios de cloud. Ceci est intéressant car TCP Cubic est surtout connu pour son comportement pour des réseaux à larges BDP; et non pas pour les réseaux de petits BDP.

Dans le chapitre 4, nous étudions les performances de TCP dans les réseaux réels en utilisant deux bancs d'essai. Le premier banc d'essai, appelé Cube, est un réseau expérimental utilisé par Orange Labs pour tester de nouveaux services fournis par la société France Télécom (FT). Le deuxième banc d'essai se compose de quelques machines Linux dans le laboratoire I3S qui peuvent être démarrées soit sous un système d'exploitation natif CentOS ou sous VMware ou Xen. Ce chapitre comprend deux parties distinctes. Dans la première partie, nous présentons les résultats obtenus avec des outils d'analyse pour la lecture et d'écriture du disque. Nous utilisons deux outils dd [Ubua] et hdparm [Ubub].

Dans la deuxième partie du chapitre 4, nous présentons les mesures effectuées à partir du réseau de machines Cube. Nous générons du trafic en provenance ou vers l'une des machines sur Cube à un hôte situé dans Orange Labs à Sophia.

Ensuite, nous présentons les mesures de Sophia vers Cube, puis de Cube vers Sophia. En augmentant le nombre de flux parallèles de Cube vers Sophia, nous avons rencontré quelques facteurs limitant les performances de ces transferts: (i) l'accès au disque; (ii) la couche de virtualisation; et (iii) un shaper. La question de l'ingénierie dans ce chapitre a été une occasion pour nous qui nous a permis de reconnaître l'interaction entre les problèmes de système et de réseau dans une solution typique de cloud.

Dans le chapitre 5, nous étudions la question de la synchronisation entre les sources TCP Cubic en détail. Nous étudions l'ampleur et les causes profondes de la synchronisation en utilisant une approche expérimentale avec un banc d'essai hébergé au laboratoire I3S combinée avec des simulations.

La première permet d'expérimenter la mise en œuvre réelle de protocole dans un environnement contrôlé, tandis que le second permet d'explorer un ensemble plus large de scénarios de réseau.

Nos contributions à l'étude de la synchronisation de TCP Cubic sont les suivantes:

- Nous établissons expérimentalement la relation entre l'existence et l'étendue de la synchronisation avec des paramètres clés comme RTT et la taille de la mémoire tampon. Nous démontrons la résilience de la synchronisation au trafic de fond, et comment l'option Convergence rapide (Fast Convergence), qui vise à rendre TCP Cubic plus équitable pour les nouvelles connexions, renforce la synchronisation. Nous utilisons New Reno comme point de référence.

- Nous démontrons que plusieurs facteurs contribuent à l'apparition de synchronisation dans TCP Cubic: (i) le taux de croissance de la fenêtre de congestion lorsqu'une source TCP Cubic atteint la capacité du réseau et de son rapport avec le RTT de la connexion; (ii) la façon dont l'algorithme de congestion de TCP Cubic suit la courbe cubique idéale dans le noyau; et (iii) la concurrence entre les sources TCP Cubic et l'agressivité des sources qui n'ont pas connu de perte durant la dernière épisode de perte.

- Nous proposons et évaluons deux approches pour réduire le niveau de la synchronisation et donc le taux de perte des transferts TCP Cubic. Nous suggérons que la synchronisation est le prix à payer pour avoir une version TCP haut débit qui doit explorer la bande passante disponible dans le réseau d'une manière superlinéaire. Il est probable que nous puissions soulager la synchronisation, comme le fait nos modifications de TCP Cubic, mais l'éliminer complètement va être une tâche complexe.

Dans le chapitre 6, nous évaluons l'impact potentiel des algorithmes de gestion de file d'attente (CoDel [NJ12b] [NJ12a], PIE [RPC+13], et ARED [FGS01]D) sur la synchronisation. Aussi, nous explorons comment les deux approches que nous avons proposées dans le chapitre 5 peuvent être combinées avec des mécanismes de files d'attente avancés pour réduire davantage la synchronisation. Nous montrons que, grâce à l'utilisation des mécanismes AQM, nous pouvons avoir de plus petits retards et réduire la synchronisation entre les flux, mais il y a un prix à payer: une perte de paquets supérieure et une dégradation de débit utile. Par rapport à nos approches, l'utilisation des AQMs pour réduire la synchronisation est plus efficace.

Enfin, le contenu de cette thèse est résumé et les perspectives sont présentées au chapitre 7.

B.1.1 Résumé par chapitre

B.1.2 Chapitre 1 : État de l'art

L'objectif de travaux de thèse est de prévoir les performances TCP des centres de données (Data Centers) et d'évaluer l'impact des différents protocoles de transport et de contrôle de congestion les plus fréquents sur les plateformes de Cloud Computing, et les méthodes necessaires pour les détecter et les corriger.

Nous avons commencé par une étude bibliographique sur les travaux de recherche liés aux différentes parties de la thèse:

- Tout d'abord, nous avons étudié les problèmes de performance des services de Cloud computing et les solutions proposées, i.e., SNAP (Scalable Network-Application Profiler) qui permet d'identifer et de résoudre les problèmes de performances dans les datacenters.

- Ensuite, nous avons détaillé le protocole TCP qui est le protocole de transport dominant dans les réseaux de Cloud. Nous avons aussi présenté quelques études relatives au (i) probléme de congestion, (ii) les algorithmes de contrôle de congestion (i.e, BIC, TCP Cubic, HSTCP, Compound TCP, Reno, New Reno), (iii) de dimensionnement de la mémoire tampon et (iv) les mécanismes de gestion de file d'attente actives (AQMs).

- Nous avons présenté quelques études qui ont établi une évaluation de la performance de TCP Cubic, par des simulations ainsi que des expériences.

- Finalement, nous avons cité les méthodes que nous avons utilisées pour l'analyse de données dans cette thèse (i.e. les techniques mathématiques de champ moyen Mean field)

Chapitre 2 : Un flux TCP Cubic simple

Au niveau de ce chapitre, nous avons développé un modèle analytique détaillé pour une connexion TCP Cubic isolée.

Nous avons commencé par une analyse approfondie du protocole TCP Cubic qui est actuellement utilisé par défaut dans les noyaux Linux, depuis la version 2.6.19.

Parmi les critères de TCP CUBIC, (1) la fenêtre de congestion suit une fonction cubique, ce qui lui permet d'atteindre plus rapidement la taille optimale que dans le cas de TCP Standard (New Reno), (2) Le temps de convergence est indépendant du RTT, ce qui rend TCP Cubic plus équitable que les autres versions de TCP à haut débit dans les réseaux avec des RTT hétérogènes. Aussi, cette caractéristique de TCP Cubic le rend plus "Friendly" par rapport à d'autres protocoles à haut débit. TCP Cubic posséde deux modes de fonctionnement, le mode TCP et le mode Cubic. Le mode TCP doit être utilisé dans le cas des réseaux à faibles BDPs, tandis que le mode Cubic est déclenché pour les réseaux à larges BDPs. Chaque mode correspond à une façon spécifique de l'augmentation de la fenêtre de congestion:

$$w_c(t) = C_{cubic}(t - V_{cubic})^3 + w_{max} \tag{B.1}$$

$$w_{tcp}(t) = w_{max}(1-\beta) + \frac{3\beta}{(2-\beta)}\frac{t}{R(t)} \tag{B.2}$$

C_{cubic} et V_{cubic} sont des paramétres TCP Cubic, t est le temps écoulé depuis la dernière perte.

Dans la deuxième partie de chapitre, nous avons développé un modèle analytique pour une connexion TCP Cubic isolée. Pour valider ce modéle, nous avons utilisé des simulations NS-2, pour différents scenarios (ADSL, FTTH).

Parmi les scénarios, nous avons considéré le cas de FTTH avec débit de 100Mbps, un RTT de 50ms et une file d'attente égale à 50 paquets. Le BDP est égal à 417 paquets et TCP Cubic fonctionne en mode Cubic. Nous avons présenté au niveau de la Figure B.1 les séries temporelles de la fenêtre de congestion pour le modèle et les simulations NS-2.

Nous observons une bonne correspondance temporelle à la fois en terme de variations d'amplitude et de la fréquence des oscillations de ce paramétre.

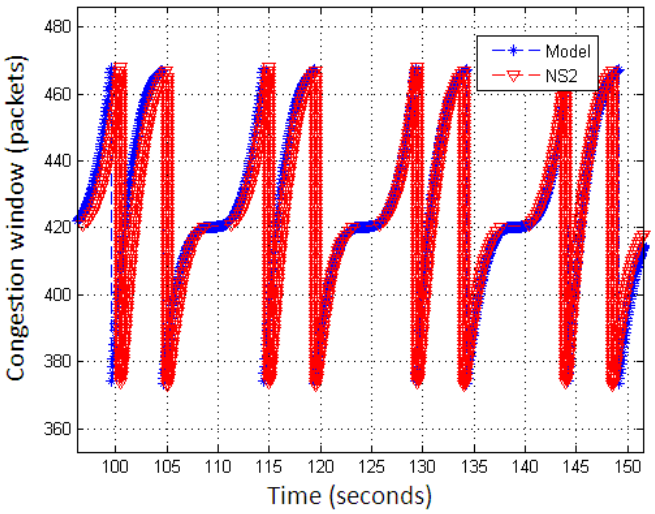

FIGURE B.1: Time series of the window, FTTH, $C = 100Mbps$

Chapitre 3 :Le modèle de champ moyen

Au niveau de ce chapitre, nous avons proposé un modèle analytique basé sur les chaînes de Markov et sur la méthode de champ moyen (Mean-field), pour comprendre les performances des centres de données, utilisés par de nombreux flux TCP qui partagent un même goulot d'étranglement. La validation du modèle a été faite par le biais de simulations NS-2.

Au niveau de chapitre nous visons à développer un modéle analytique pour TCP Cubic où un grand nombre de flux longs TCP Cubic partagent un goulot d'étranglement. Nous avons utilisé une approche de champ moyen (Mean-field) conduisant à un modèle fluide. La validation du modèle a été faite par une comparaison avec NS-2.

Nous avons consideré 3 scenarios: client FTTH, transfert au sein des centre de données, transfert entre les centres de données.

Dans le cas de scénario FTTH TCP Cubic opère dans le mode Cubic. Nous présentons dans la Figure B.2 la série de temps de taille de la fenêtre de congestion ainsi que la file d'attente pour le modèle et les simulations. Une fois que les simulations et le modèle ont atteint l'équilibre, nous observons une bonne correspondance temporelle à la fois en termes de variation des amplitudes et de fréquence des oscillations des deux métriques.

Néamoins, la correspondance entre le modèle et les simulations est moins bon dans le cas du scénario de transfert entre les centres de données. Dans le modèle fluide, seulement une fraction des sources subissent des pertes lorsque le tampon est plein. Les autres sources continuent d'augmenter leurs fenêtres, ce qui conduit à des pertes à une vitesse plus élevée que pour les simulations. Dans le cas des simulations toutes les

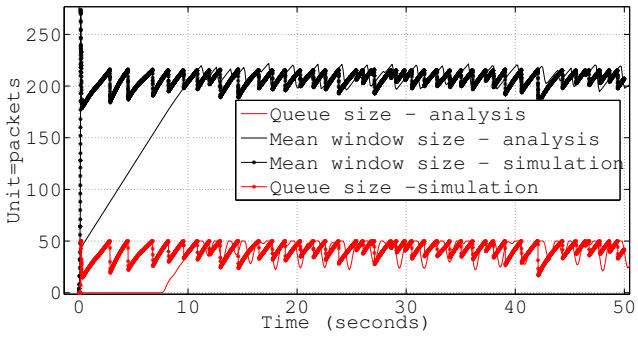

FIGURE B.2: Time series of queue size and average window size - FTTH scenario - Cubic

sources perdent simultanément des paquets. Une analyse plus approfondie sera effectuée dans le chapitre suivant sur l'origine de la synchronisation des flux TCP Cubic.

Ensuite, nous avons évalué l'efficacité et l'équité de TCP Cubic par rapport à celle de New Reno pour un ensemble de scénarios de réseau cloud.

Nous avons enfin étudié l'équité et l'impact de la taille du tampon de TCP Cubic et TCP New Reno. TCP Cubic est à la fois plus efficace et plus juste que TCP New Reno, en particulier dans le cas de faibles tailles de tampon. Nos résultats montrent que, contrairement à TCP New Reno, TCP Cubic est capable de survivre avec un tampon de taille aussi petite que 30% de la BDP.

Chapitre 4 : Cube

Nous avons fait plusieurs tests d'analyse comparative des performances disque et réseau du centre de données Cube d'Orange.

Nous avons comparé les résultats sur Cube avec ceux d'une plateforme de test au laboratoire I3S. Nous avons utilisé deux outils (hdparm et dd) pour vérifier les débits de lecture et d'écriture des disques. La question posée était de savoir si les accès disques dans un environnement virtualisé, peuvent limiter les débits des transferts réseaux. Une comparaison entre tous ces résultats nous a permis d'identifier les principaux facteurs limitant les performances sur ce Cloud.

Nous avons ensuite effectué plusieurs tests à l'aide de différentes méthodes de transfert de données ("Iperf et Socket Perl"). Ceci nous a permis d'identifier les différentes caractéristiques des connexions TCP vers / depuis Cube. Les tests de Sophia vers Cube avec un seul flux montrent un débit qui varie entre 100 et 400 Mbps. En variant le nombre de flux de 1 à 300 nous remarquons la dégradation des performances.

Les facteurs qui limitent les performances de ces transferts sont: (i) l'accès au disque; (ii) la couche de virtualisation; et (iii) un shaper. La question de l'ingénierie dans ce

chapitre a été une occasion pour nous de reconnaître l'interaction entre les problèmes de système et de réseau dans une solution typique de cloud

Chapitre 5 : Synchronisation des connexions TCP Cubic

L'objectif de cette partie était de mieux comprendre le comportement de TCP Cubic (version de TCP par défaut pour les noyaux Linux actuels) dans des environnements de Cloud Computing pour des scénarios avec un grand nombre de longs flux TCP. Dans de telles situations, les connexions Cubic ont tendance à se synchroniser les unes avec les autres et cette synchronisation est plus élevée que pour des connexions TCP standard. Ce phénomène dégrade les performances car toutes les sources TCP perdent des paquets au même moment et diminuent leur débit et le réseau peut se retrouver sous-utilisé.

Nous avons étudié ce phénomène en détail par des expérimentations sur un banc d'essai (testbed) contrôlé, des mesures avec les serveurs d'Amazon EC2, situé aux États-Unis et des simulations NS-2.

Nous avons démontré que plusieurs facteurs contribuent à l'apparition de cette synchronisation des connexions Cubic: (i) le taux de croissance de la fenêtre de congestion à son point d'inflexion, (ii) la façon dont la fenêtre de congestion Cubic suit la courbe cubique idéale dans le noyau, (iii) la concurrence entre les sources Cubic et (iv) l'agressivité des sources qui n'ont pas observé de pertes au cours de la période de saturation du tompon. Nous avons proposé et évalué deux modifications de l'algorithme TCP Cubic (LinCubic, AccuCubic), pour réduire cette synchronisation.

Pour évaluer l'impact de ces différentes modifications, nous les avons implémenté dans NS-2 et nous avons observé leur comportement dans le cas d'un seul flux TCP Cubic. Nous avons également testé le bénéfice potentiel de ces modifications dans le cas des 100 flux concurrents pour le goulot d'étranglement.

Les résultats montrent que LinCubic diminue le nombre de flux synchronisés ainsi que AccuCubic lorsque FC est utilisé. Lorsque FC est éteint, seul LinCubic performe mieux que TCP Cubic.

Dans le cas de 100 flux, nous considérons un scénario avec un RTT égal à 500ms et une taille du tampon égal à 1 BDP. Nous présentons le nombre de flux synchronisés dans la Figure B.3 pour une execution parmi les 10 effectuées. Les résultats montrent que LinCubic et AccuCubic diminuent le nombre de flux synchronisés.

Chapitre 6 : L'impact des mécanismes de gestion de file d'attente sur la synchronisation

L'objectif de cette partie est d'étudier et d'évaluer les performances de divers mécanismes de contrôle de files d'attente (AQM - Active Queue Management) pour savoir s'ils permettent de réduire la synchronisation des connexions TCP Cubic, et de comparer leur

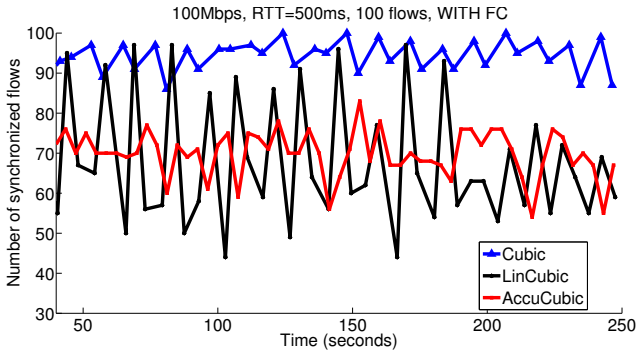

FIGURE B.3: Ns2 Simulations -100 flows, RTT=500ms, With FC

efficacité avec les deux approches LinCubic et AccuCubic.

Nous avons plus précisément considéré les mécanismes CoDel (Controlled Delay) et PIE (Proportional Integral Enhanced) en cours de normalisation par l'IETF et RED (Random Early Discard) qui est le plus ancien et présent dans la plupart des routeurs actuels. Après une première comparaison de ces mécanismes par simulations NS-2, nous avons étudié leur comportement et leurs performances sur des expérimentations.

La Figure B.4 représente le nombre de flux synchronisés pour chaque événement de congestion. On remarque que RED arrive à réduire la synchronisation entre les flux TCP Cubic. Donc, l'utilisation des AQMs pour réduire la synchronisation était plus efficace que les solutions que nous avons proposé, à savoir AccuCubic et LinCubic.

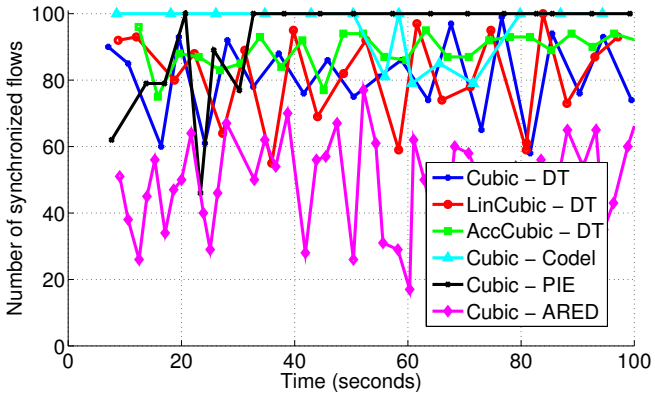

FIGURE B.4: Number of synchronized flows, 100Mbps, RTT=350ms, BS=1BDP

Bibliography

[Ada13] Richelle Adams. Active Queue Management: A Survey. *IEEE Communications surveys & Tutorials*, 15(3), 2013.

[AKM04] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing Router Buffers. *SIGCOMM Comput. Commun. Rev.*, 34(4):281–292, aug 2004.

[APU12] Adrian Arsene, Dino Lopez Pacheco, and Guillaume Urvoy-Keller. Understanding the network level performance of virtualization solutions. In *1st IEEE International Conference on Cloud Networking, CLOUDNET 2012, Paris, France, November 28-30, 2012*, pages 1–5, 2012.

[ARFK10] I Abdeljaouad, H Rachidi, S Fernandes, and A Karmouch. Performance analysis of modern TCP variants: A comparison of Cubic, Compound and New Reno. In *Communications (QBSC), 2010 25th Biennial Symposium on*, pages 80–83. IEEE, 2010.

[ATRK10] Alexander Afanasyev, N. Tilley, Peter L. Reiher, and Leonard Kleinrock. Host-to-Host Congestion Control for TCP. *IEEE Communications Surveys and Tutorials*, 12(3):304–342, 2010.

[BAC09] A. Blanc, K. Avrachenkov, and D. Collange. Comparing Some High Speed TCP Versions under Bernoulli Losses. In *PFLDNet*, 2009.

[BAM10] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, 2010.

[BLB08] M. Benaïm and J.-Y. Le Boudec. A Class Of Mean Field Interaction Models for Computer and Communication Systems. *Performance Evaluation*, 65(11-12), 2008.

[BMM07] Jean-Yves Le Boudec, David McDonald, and Jochen Mundinger. A generic mean field convergence result for systems of interacting objects. In *QEST*, 2007.

[BMP10] Charles Bordenave, David McDonald, and Alexandre Proutière. A particle system in interaction with a rapidly varying environment: Mean field limits and applications. *Networks and Heterogeneous Media*, 5(1), jul 2010.

[BMR02] F. Baccelli, D. R. McDonald, and J. Reynier. A mean-field model for multiple TCP connections through a buffer implementing RED. *Perform. Eval.*, 49(1-4), sep 2002.

[BSC+13] Sonia Belhareth, Lucile Sassatelli, Denis Collange, Dino Lopez Pacheco, and Guillaume Urvoy-Keller. Understanding TCP Cubic performance in the cloud: A mean-field approach. In *CLOUDNET*, pages 190–194, 2013.

[BtLB08] Michel Benaïm and Jean Yves Le Boudec. A class of mean field interaction models for computer and communication systems. *Perform. Eval*, pages 11–12, 2008.

[BW09] Michel Benaïm and Jörgen Weibull. Mean-field approximation of stochastic population processes in games. Working Papers hal-00435515, HAL, 2009.

[BWL10] Wei Bao, Vincent W. S. Wong, and Victor C. M. Leung. A Model for Steady State Throughput of TCP CUBIC. In *GLOBECOM*, pages 1–6. IEEE, 2010.

[CB07] Shan Chen and Brahim Bensaou. Can high-speed networks survive with Droptail queues management? *Elsevier Comput. Netw.*, 51(7), may 2007.

[CCB10] Jacopo Chicco, Denis Collange, and Alberto Blanc. Simulation Study of New TCP Variants. In *IEEE symposium on Computers and Communications (ISCC)*, Riccione, Italie, Jun 2010.

[CEH+07] Han Cai, Do Young Eun, Sangtae Ha, Injong Rhee, and Lisong Xu. Stochastic Ordering for Internet Congestion Control and its Applications. In *IEEE Infocom*, 2007.

[CEH+09] Han Cai, Do Young Eun, Sangtae Ha, Injong Rhee, and Lisong Xu. Stochastic convex ordering for multiplicative decrease internet congestion control. *Comput. Netw.*, 53(3):365–381, feb 2009.

[CK05] Vinton G. Cerf and Robert E. Kahn. A protocol for packet network intercommunication. *SIGCOMM Comput. Commun. Rev.*, 35(2):71–82, apr 2005.

[Col98] Denis Collange. Analyse dynamique de plusieurs connexions TCP synchronisées. Technical report, Orange Labs, October 1998.

[Coma] Gerald Combs. Wireshark. http://www.wireshark.org/.

[Comb] IBM Company. Aspera FASP High Speed Transport.

[ed14] Random early detection. Random early detection — Wikipedia, the free encyclopedia, 2014. Online; accessed 06-October-2014.

[EGG+05] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Part III: routers with very small buffers. *Computer Communication Review*, 35(3):83–90, 2005.

[fANR] National Laboratory for Applied Network Research. Iperf. http://iperf.fr/.

[FGS01] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. Technical report, 2001.

[FJ93a] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, aug 1993.

[FJ93b] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, August 1993.

[Flo94] Sally Floyd. Tcp and Explicit Congestion Notification. *SIGCOMM Comput. Commun. Rev.*, 24(5):8–23, oct 1994.

[Flo08] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166 (Informational), March 2008.

[Fou09] Linux Foundation. netem, 2009.

[GJN11] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361, August 2011.

[GKF13] Eduard Grigorescu, Chamil Kulatunga, and Gorry Fairhurst. Evaluation of the impact of packet drops due to AQM over capacity limited paths. In *2013 21st IEEE International Conference on Network Protocols, ICNP 2013, Göttingen, Germany, October 7-10, 2013*, pages 1–6, 2013.

[GN11] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark Buffers in the Internet. *Queue*, 9(11):40:40–40:54, nov 2011.

[Gro13] Orange Groupe. Cloud computing et entreprise : Orange et le cloud computing. http://fr.wikiversity.org/wiki/Cloud_computing_et_entreprise/Orange_et_le_cloud_computing, 2013.

[HJ13] Toke Hiland-Jrgensen. The state of the Art in Bufferbloat Testing and Reduction on linux. 12th March 2013.

[HR08] S. Hassayoun and D. Ros. Loss synchronization and router buffer sizing with high-speed versions of TCP. In *IEEE Infocom Workshops*, pages 1–6, 2008.

[HR09] Sofiane Hassayoun and David Ros. Loss synchronization, router buffer sizing and high-speed TCP versions: Adding RED to the mix. In *IEEE LCN*, pages 569–576, 2009.

[HRX08] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5), jul 2008.

[HUKC+11] Aymen Hafsaoui, Guillaume Urvoy-Keller, Denis Collange, Matti Siekkinen, and Taoufik En-Najjary. Understanding the impact of the access technology: the case of web search services. In *TMA*, 2011.

[Jac88] V. Jacobson. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[JD04] Hao Jiang and Constantinos Dovrolis. The origin of TCP traffic burstiness in some time scales. Technical report, in IEEE INFOCOM, 2004.

[jlo08] jlothian. Intro to benchmarking part 1: Disks/storage. https://lopsa.org/node/1711, 2008.

[JR11] S. Jain and G. Raina. An experimental evaluation of CUBIC TCP in a small buffer regime. In *NCC*, 2011.

[KSG+09] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The Nature of Data Center Traffic: Measurements & Analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[Kur70] T. G. Kurtz. Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes. *Journal of Applied Probability*, 7(1), 1970.

[Lei07] Shorten R.N. McCullagh G. Leith, D.J. Experimental Evaluation of Cubic TCP. *Protocols for Fast Long Distance Networks 2007, Los Angeles*, 2007.

[LGBP10] Patrick Loiseau, Paulo Gonçalves, Julien Barral, and Pascale Vicat-Blanc Primet. Modeling TCP throughput: An elaborated large-deviations-based model and its empirical validation. *Perform. Eval.*, pages 1030–1043, 2010.

[LHB05] Junsoo Lee, Joao P. Hespanha, and Stephan Bohacek. A study of TCP fairness in high-speed networks. Technical report, 2005.

[LHHL12] Tuan Anh Le, Rim Haw, Choong Seon Hong, and Sungwon Lee. A Multipath Cubic TCP Congestion Control with Multipath Fast Recovery over High Bandwidth-Delay Product Networks. *IEICE Transactions*, 95-B(7):2232–2244, 2012.

[LLS07] Yee-Ting Li, D. Leith, and R.N. Shorten. Experimental Evaluation of TCP Protocols for High-Speed Networks. *Networking, IEEE/ACM Transactions on*, 15(5):1109–1122, Oct 2007.

[LSM07] D. Leith, R. Shorten, and G. McCullagh. Experimental evaluation of Cubic-TCP. *PFLDNet*, 2007.

[LYKZ10] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 1–14, New York, NY, USA, 2010. ACM.

[Mar10] Luis MartinGarcia. TCPDUMP & LIBPCAP. http://www.tcpdump.org/, 2010.

[MB13] M. Thomson (ed.) A. Melnikov (ed.) M. Belshe, R. Peon. Hypertext Transfer Protocol version 2.0. IETF draft draft-ietf-httpbis-http2-09, July 2013.

[MBT12] R. Peon M. Belshe Twist. SPDY: An experimental protocol for a faster web. RFC 2012, August 2012.

[NJ12a] Kathleen Nichols and Van Jacobson. A Modern AQM is just one piece of the solution to bufferbloat. *Queue*, 10(5):20:20–20:34, may 2012.

[NJ12b] Kathleen M. Nichols and Van Jacobson. Controlling Queue Delay. *Commun. ACM*, 55(7):42–50, 2012.

[NK13] M. Welzl N. Khademi, D. Ros. The New AQM Kids on the Block: Much Ado About Nothing? Technical report, University of Oslo, 2013.

[Ost94] Shawn Ostermann. tcptrace. http://www.tcptrace.org/, 1994.

[Pao12] Jean Paoli. Speed and Mobility: An Approach for HTTP 2.0 to Make Mobile Apps and the Web Faster. MSDN blog Interoperability @ Microsoft, March 2012.

[PFTK00] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose. Modeling TCP Reno performance: a simple model and its empirical validation. *IEEE/ACM Trans. on Netw.*, 8(2), 2000.

[PS11] S. Poojary and V. Sharma. Analytical model for congestion control and throughput with tcp CUBIC connections. In *IEEE Globecom*, 2011.

[qui13] Experimenting with QUIC. http://blog.chromium.org/2013/06/experimenting-with-quic.html, June 2013. Chromium Official Blog.

[Rai05] *Buffer sizes for large multiplexers: TCP queuing theory and instability analysis*, Rome, Italy, April 2005.

[Rom10] Roman. How to use 'dd' to benchmark your disk or CPU? https://romanrm.net/dd-benchmark, 2010.

[RPC+13] R.Pan, P.Natarajan, C.Piglione, M.S.Prabhu, V.Subramanian, F.Baker, and B.VerSteeg. PIE: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing*, pages 148–155, July 2013.

[TIIN10] Omesh Tickoo, Ravi Iyer, Ramesh Illikkal, and Don Newell. Modeling Virtual Machine Performance: Challenges and Approaches. *SIGMETRICS Perform. Eval. Rev.*, 37(3):55–60, January 2010.

[TSZS06] Kun Tan, Jingmin Song, Qian Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *IEEE Infocom*, 2006.

[Ubua] Ubuntu. dd. http://doc.ubuntu-fr.org/dd.

[Ubub] Ubuntu. hdparm. http://doc.ubuntu-fr.org/hdparm.

[Uni80] Unix. Socket. http://fr.wikipedia.org/wiki/Socket, 1980.

[Val10] Juan Valencia. How to create a reverse SSH tunnel. http://www.jveweb.net/en/archives/2010/09/how-to-create-a-reverse-ssh-tunnel.html, 2010.

[VHV12] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter TCP (D2TCP). In *ACM Sigcomm*, 2012.

[VPS+09a] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. *SIGCOMM Comput. Commun. Rev.*, 39(4):303–314, aug 2009.

[VPS+09b] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *Proc. ACM SIGCOMM*, Barcelona, Spain, aug 2009.

[VS94] Curtis Villamizar and Cheng Song. High Performance TCP in ANSNET. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, oct 1994.

[Wal87] Larry Wall. The Perl Programming Language. http://fr.wikipedia.org/wiki/Socket, 1987.

[Whi13] G. White. A Simulation Study of Codel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks. Technical report, CableLabs Technical Report, 2013.

[wik] wikipedia. Tunneling protocol. http://en.wikipedia.org/wiki/Tunneling_protocol.

[WM05] Damon Wischik and Nick McKeown. Part I: buffer sizes for core routers. *SIGCOMM Comput. Commun. Rev.*, 35(3), jul 2005.

[WN10] Guohui Wang and T. S. Eugene Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 1163–1171, Piscataway, NJ, USA, 2010. IEEE Press.

[WRGH11] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, Implementation and Evaluation of Congestion Control for Multipath tcp. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 99–112, Berkeley, CA, USA, 2011. USENIX Association.

[XHR04a] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE Infocom*, 2004.

[XHR04b] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *INFOCOM*, 2004.

[YGM+11] Minlan Yu, Albert Greenberg, Dave Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. Profiling Network Performance

for Multi-tier Data Center Applications. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 57–70, Berkeley, CA, USA, 2011. USENIX Association.

[YLX+11] Peng Yang, Wen Luo, Lisong Xu, Jitender Deogun, and Ying Lu. TCP congestion avoidance algorithm identification. In *ICDCS*, 2011.