Network Architectures
and Services
NET 2015-07-1

Dissertation

End-to-End Flow-Level Quality-of-Service Guarantees for Switched Networks

Fabien Clément Geyer

Department of Informatics

Technische Universität München

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

End-to-End Flow-Level Quality-of-Service Guarantees for Switched Networks

Fabien Clément Geyer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Johann Schlichter
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Georg Carle
	2.	Prof. Jörg Liebeherr, Ph.D. (University of Toronto, Kanada)

Die Dissertation wurde am 12.05.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 02.07.2015 angenommen.

Cataloging-in-Publication Data
Fabien Clément Geyer
End-to-End Flow-Level Quality-of-Service Guarantees for Switched Networks
Dissertation, July 2015
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN 978-3-937201-49-8
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
DOI 10.2313/NET-2015-07-1
Network Architectures and Services NET-2015-07-1
Series Editor: Georg Carle, Technische Universität München, Germany

ABSTRACT

Industrial networks and communications are seeing today a shift from high cost and proprietary technologies to low cost and open solutions based on Ethernet. While Ethernet is already largely used for interconnecting traditional personal computers, packets and flows are generally transferred in a best-effort manner on such networks without offering any guarantees. Such property is insufficient for industrial applications in need of real-time communications. Various solutions have been proposed to address this issue, but they are mostly focusing on Quality-of-Service (QoS) for individual packets and are generally pessimistic and rigid. While some applications can benefit from per-packet guarantees, it does not apply to applications where bandwidths or transfer durations are key properties.

We investigate in this thesis the question of QoS at flow level and work toward the challenge of guarantees for communications using the *Transmission Control Protocol*, or TCP. By effectively looking at the flow level of a network, we are able to have a more abstract view of the network. TCP has already been largely investigated since the late 90's, but the question of guarantees for TCP flows still remains mostly unanswered.

In this thesis, we are interested in the mathematical modeling of network flows in Ethernet networks. We look at the different levels of the network stack, from packet scheduling to application layer. We first investigate packet scheduling in Ethernet networks which enables us to mix time-critical traffic with other less demanding flows such as TCP based transfer applications. We then look at mathematical modeling of TCP based flows. The main contribution of this thesis is the extension of the so-called *flow-level network modeling*. This mathematical framework was initially designed for performance evaluation of infinite TCP flows on single bottleneck networks. This framework has since then be extended to performance evaluation of any topology. We first present in this thesis how to apply it to Ethernet networks. We then extend the framework in order to include the influence of cross-traffic on performances of infinite TCP flows. By considering TCP flows as bidirectional, we effectively increase the precision of the model. We then refine this mathematical framework by including the evaluation of non-permanent, or short, TCP flows. We investigate here ON/OFF TCP flows alternating between transfer and idle periods. By correctly accounting for the statistical bandwidth sharing property of TCP as well as its slow-start phase, we extend the usability of the framework of flow-level network modeling. Next, we use our model in order to give guarantees on short TCP flows. We develop a stochastic bound on the transfer time of TCP flows, which enables us to dimension a network based on stochastic time requirements. Each step is validated against discrete event simulations. Finally, we also investigate how to model higher layer applications working on top of TCP and UDP, and propose a way to have realistic models of those using statistical analysis of network captures.

ZUSAMMENFASSUNG

Industrielle Netzwerke und Kommunikationssysteme erleben heutzutage eine Veränderung von kostenintensiven und proprietären Technologien zu preisgünstigen und offenen Ethernet basierten Lösungen. Während Ethernet bereits für traditionelle „Personal Computer“ Vernetzung verwendet wird, werden Pakete und Datenströme generell in solchen Netzwerken nach dem „Best-Effort“ Prinzip und ohne Garantien übertragen. Solch eine Eigenschaft ist nicht ausreichend für industrielle Anwendungen, die Echtzeitkommunikation benötigen. Verschiedene Lösungen wurden vorgeschlagen, um sich diesem Problem zu widmen, die sich aber meistens auf „Quality-of-Service“ (QoS) für individuelle Pakete fokussieren und generell pessimistisch und unflexibel sind. Während einige Anwendungen von Paket-basierten Garantien profitieren können, trifft dies nicht auf Anwendungen zu, bei denen Bandbreite oder Gesamtübertragungsdauer die Haupteigenschaften sind.

Wir untersuchen in dieser Dissertation die Frage von QoS in der Datenflussschicht, und arbeiten an der Herausforderung von Garantien für Kommunikation mit dem „Transmission Control Protocol“, bzw. TCP. Durch eine effektive Betrachtung des Netzwerks in der Datenflussschicht erhalten wir eine abstraktere Sicht auf das Netzwerk. TCP wurde bereits umfangreich seit den späten 90er Jahren untersucht, aber die Frage von Garantien für TCP Datenströme bleibt weitestgehend unbeantwortet.

In diese Dissertation sind wir an der mathematischen Modellierung von Netzwerkflüssen in Ethernet-Netzwerken interessiert. Wir betrachten die verschiedenen Schichten des Netzwerkstacks, vom Paket-Scheduling bis zur Anwendungsschicht. Wir untersuchen Paket-Scheduling in Ethernet Netzwerken, das uns Zeitkritische Flüsse mit anderen weniger anspruchsvollen sowie TCP-basierten Übertragungsanwendungen zu mischen ermöglicht. Danach betrachten wir die mathematische Modellierung von TCP-basierten Netzwerkflüssen. Der Hauptbeitrag von dieser Dissertation ist die Erweiterung des sogenannten „Flow-level Network Modeling“. Dieses mathematische Framework wurde anfangs zur Leistungsevaluierung von infiniten TCP Datenflüssen in Netzwerken mit einem Einzelflaschenhals benutzt. Dieses Framework wurde seitdem zur Leistungsevaluierung von beliebigen Topologien erweitert. Wir stellen zunächst in diese Dissertation vor, wie man es auf Ethernet Netzwerke anwendet. Daraufhin erweitern wir dieses Framework, um den Einfluss von kreuzendem Datenverkehr auf die Performanz von infiniten TCP Datenflüssen mit einzubeziehen. Durch die Berücksichtigung der bidirektionalen Eigenschaft von TCP Datenflüssen, verbessern wir die Genauigkeit des Modells. Wir verfeinern dann dieses mathematische Framework durch die Einbeziehung von finiten oder kurzen TCP Datenflüssen. Wir untersuchen hier ON/OFF TCP Datenflüsse, die zwischen Übertragungs- und Ruhezeiten wechseln. Durch die Berücksichtigung der statistischen Bandbreitenteilungs-Eigenschaft von TCP, sowie seiner Slow-Start Phase, erweitern wir die Anwendbarkeit des „Flow-level Network Modeling“ Frameworks. Dann benutzen wir unser Modell um Garantien für kurze TCP Datenflüsse zu erzeugen. Wir entwickeln eine stochastische Grenze für die Übertragungsdauer von TCP Datenflüssen, die es uns ermöglicht, ein Netzwerk zu dimensionieren basierend auf stochastischen Zeitanforderungen. Jeder Schritt wurde gegen ereignisorientierte Simulation validiert. Abschließend untersuchen wir wie wir höheren UDP- und TCP-basierten Anwendungsschicht modellieren können, und schlagen eine Lösung vor, mittels statistischer Analyse von Netzwerk-Aufzeichnungen realistische Modelle von diesen Anwendungsschichten zu erhalten.

ACKNOWLEDGMENTS

I'm grateful for the support and encouragements I received from certain people without whom this thesis would not have been possible.

First of all, I would like to thank Prof. Georg Carle for supervising this dissertation and the various advices he gave me along the way. I would also like to thank Prof. Jörg Liebeherr for his feedback on my work and being my second assessor and Prof. Johann Schlichter for chairing the examination committee.

As this work was mostly performed during my time at Airbus Group Innovations in Munich, I would like to thank Josef Schalk and Stefan Schneele for the job opportunity and their support. I would like to thank all the colleagues and fellow PhD candidates I had the pleasure to meet, share an office with or collaborate with during my time there. A particular thanks goes to Nils Tobeck for helping me with my poor German grammar when writing the "Zusammenfassung" of this dissertation. I would also like to thank the various researchers I had the opportunity to meet during conferences, who often sparked some interesting ideas worth investigating, as well as the anonymous reviewers who took time to evaluate and provide feedback on my work.

I'm also grateful for the few longtime friends I had during this time away from my home country, who kept me sane, distracted me from work when necessary and shared a few long bicycle rides with me. Finally, my biggest gratitude goes to my parents and family for their kind support throughout this work and all the preceding years. *Un grand merci à vous.*

Munich, July 30, 2015

Fabien Geyer

CONTENTS

Part I

INTRODUCTION AND BACKGROUND

1. INTRODUCTION

In the last few decades, analog functions and isolated processing devices in industrial environments have been increasingly replaced by digital and interconnected devices. In order to ensure a proper and safe operation of those networked devices, a fundamental requirement for the network is to provide performance guarantees – or Quality-of-Service (QoS) – for communications. Methods and mathematical models for performance evaluation of networks are still an important area of research in computer science, due to the wide range of solutions, protocols and use-cases.

Various networking solutions used for interconnecting industrial devices have been proposed throughout the years, ranging from simple point-to-point connections to complex networks with thousands of nodes. Among the various propositions, Ethernet has emerged as a dominant solution and is used in various industrial sectors (automotive, aviation, robotics, assembly lines, ...), mainly because it is cost effective due to its wide use in traditional computer networks, provides high bandwidths, and is open and interoperable. The main current drawback of standard Ethernet is that it mainly works on a best-effort basis and does not currently offer a standard way of providing and enforcing performance guarantees.

As performance guarantees play a key role in the use – and also often certification – of industrial networks, methods to engineer and enforce QoS in Ethernet networks are needed. Due to the historical development of those industrial networks, the methods which are used are still mainly based on old paradigms where the focus is to ensure guarantees on individual packets. Performances measures customarily used are per-packet maximum end-to-end latency and jitter, as well as insurance that packets are not dropped. Focusing on the previously mentioned performance measures leads to the following challenges. First, while such requirements are useful for certain real-time applications such as actuators and sensor interactions, they are not adapted to other types of applications such as elastic traffic for instance, which is able to adapt its bandwidth to available network resources. Secondly, methods currently used to ensure those performances are often too inflexible and generally lead to over-provisioning.

In this thesis, we look at current and new paradigms for providing guarantees in Ethernet networks, from concrete solutions investigated in the aeronautic industry to mathematical models of protocols. We first evaluate current and newly proposed solutions for packet scheduling in avionic scenarios and propose a solution to reduce packet latency in Ethernet networks. We then concentrate on the core of this thesis, which is mathematical modeling of the performances of Ethernet networks with elastic traffic, such as protocols based on the Transmission Control Protocol (or TCP). Instead of focusing on packet-level guarantees, we propose to look at higher levels performances, namely flow-level and application-level performances. Those models are an important step towards more flexible and more practical network provisioning as well as better utilization of network resources.

1.1 Problem statement and research objectives

Current industrial uses of Ethernet networks provide engineers with affordable and open communications solutions, but in order to ensure real-time communications and per-packet performances guarantees, inflexible and pessimistic Quality-of-Service policies are often used, generally leading to over-provisioned networks. The general goals of this thesis are to provide solutions to reduce over-provisioning, and open the door to new applications, such as networks with mixed-criticality and guaranteed elastic communications, by developing mathematical models, algorithms and tools to enable more flexible analysis, planning and modeling of industrial Ethernet networks.

This thesis is structured around the two following objectives.

O1: Evaluate mechanisms for Ethernet networks with mixed-criticality

The first objective of this thesis is to evaluate solutions which enable Ethernet networks to host a mix of guaranteed real-time and elastic traffic, or simply best-effort traffic. We aim here at providing new services on top of existing legacy networks, where the influence of new traffic on the performances of the legacy applications should be minimized.

In particular, we analyze here different packet scheduling algorithms for Ethernet networks, namely well-established algorithms and architectures proposed in the literature, a new algorithm recently proposed by the industry which targets a similar research objective, and a novel algorithm optimizing the performances of one type of traffic.

This research objective is divided into three sub-research objectives:

O1.1 *Investigate the impact of packet scheduling algorithms in the context of avionic networks and identify the ones enabling the mix of real-time with best-effort traffic.*

We evaluate and compare priority based and fair bandwidth sharing algorithms using discrete event simulations in Chapters 3 and 4. We show that a good compromise is to use a hybrid priority/fair queuing scheduler, where flows with low-latency requirements are treated with the priority based part, while the best-effort flows use the fair queuing part.

O1.2 *Evaluate the benefits of the newly proposed scheduling architecture from the IEEE which address the challenge of mixing audio and video flows with best-effort traffic.*

We evaluate the IEEE 802.1Qav scheduling architecture in Chapter 3. We show that in term of pure performance (latency and jitter), it produces slightly higher values compared to traditional work-conserving algorithms, but within requirement ranges. Thanks to its shaping functionality and provable worst-case latency bound, this new algorithm is a good candidate for future avionic architectures.

O1.3 *Investigate a new packet-forwarding and scheduling architecture for enhancing the performance of periodic real-time traffic mixed with best-effort traffic.*

We propose an architecture based on the Time Division Multiple Access (TDMA) principle in Chapter 5. TDMA is used in combination with the novel algorithm called Time-Aware Deficit Round Robin (TADRR), which is a hybrid scheduler preventing queuing delay for real-time packets while having a fair bandwidth sharing property for best-effort traffic.

O2: Develop mathematical models for network engineering with elastic traffic

The second objective of this thesis is to investigate mathematical frameworks and methods for performance evaluation of inelastic and elastic flows in Ethernet networks, and evaluate how those methods can be used for network engineering in order to provide flow-level and application-level guarantees.

In particular, we first evaluate the state-of-the-art in mathematical models for performance evaluation of Ethernet networks in order to identify potential candidates for further development. Based on this analysis, we then extend one particular approach in order to be able to analyze Ethernet networks with low-latencies with a mix of inelastic and elastic flows. We then build on this work to include statistical properties of the traffic in order to give a finer performance measures with the goal of diminishing network over-provisioning and provide new Quality-of-Service attributes. Finally, we investigate how application layer can be modeled.

This research objective is divided into five sub-research objectives:

O2.1 *Identify performance evaluation frameworks as well as Quality-of-Service attributes which are relevant for Ethernet networks with elastic traffic such as TCP based flows.*

We establish in Chapter 6 that from a flow-layer perspective, packet end-to-end performances are less critical for elastic traffic compared to real-time one. One key performance indicator is the time to transfer an application-layer message, which can be decomposed in multiple packets.

O2.2 *Propose an analytical framework which is relevant for the evaluation of networks with UDP and TCP flows at the network layer.*

We compare various frameworks in Chapter 6 and decide to focus on the so-called flow-level modeling approach due to its realistic models and flexibility. We first formalize it in Chapter 7, apply it to Ethernet networks where queuing delay are non-negligible, and extend it to take into account one non-intuitive behavior of TCP.

O2.3 *Extend this framework to the evaluation of mean performances of short flows.*

We extend the flow-level modeling approach proposed in Research Objective **O2.2** to the study of short flows in Chapter 8. We use a simple ON/OFF model of flow behavior and we follow an approach similar to the Engset model to give mean performances on transfer duration for TCP flows.

O2.4 *Extend this approach to give statistical guarantees on tail performances.*

We establish a new stochastic bound on fair-bandwidth scheduling using stochastic network calculus in Chapter 10. Using this bound and the results from Research Objectives **O2.2** and **O2.3**, we give tail performances on the duration of short ON/OFF TCP flows.

O2.5 *Propose a method to give realistic characteristics to short flows in order to map application layer performances.*

In order to give a more realistic application-layer model than the simple ON/OFF one used in Research Objectives **O2.3** and **O2.4**, we propose a model based on four layers: messages, flows, sessions and user. In order to give realistic values this model, we use a statistical analysis of network captures and develop a tool for automatically performing this analysis.

1.1.1 Positioning and goals

Research on performances of networks can be roughly divided into three research fields. The first category focuses on research on end-to-end communications performances, or in other words, network protocols performances. These works generally propose enhancement of existing protocols or – in smaller occasions – the development of new ones. With its almost universal adoption in a large variety of existing network conditions – wired or wireless Local Area Networks (LANs), Wide Area Networks (WANs), the Internet, satellite communications, etc. – TCP in particular has attracted a lot of attention as a one-size-fits-all congestion-control algorithm is yet to be found.

The second field of research focuses on switching and forwarding elements of networks by looking at measures and architectures to achieve Quality-of-Service, such as queuing management, packet scheduling algorithms and network engineering. By directly acting on elements which are in charge of forwarding packets in a network, those elements are able to drive the performances of network protocols according to required performance measures.

Finally the third field is interested with mathematical modeling of networks and it builds often bridges between the two previous fields of research. These works concentrate on understanding – with help of abstraction and mathematical frameworks – how network performances evolve according to perturbations or evolutions of network properties. Theories and concepts developed in those works are often the drive for changes and new propositions in the two previous fields.

This thesis and its contributions can be assigned to the second field of research for Part II, and to the third field for Parts III to IV.

1.2 Contributions and document structure

The contributions of this thesis are:

- The study of existing packet scheduling algorithms in two networks architectures currently used in aircrafts or under development (Chapters 3 and 4);
- A proposition of a new packet scheduling algorithm, allowing a mix of real-time traffic with ultra-low latencies requirements with non-critical elastic traffic (Chapter 5);
- A mathematical framework for studying performances of TCP flows in Ethernet networks, by extending well-studied models to particular effects of small LANs (Chapters 6 and 7);
- Methods to provide guarantees for elastic flows (Chapters 7, 8 and 10);

– Tools implementing part of the analytical results developed in this thesis (Chapters 7 and 10).

Table 1.1 shows the individual contributions and methodologies used for their evaluation, with their relationship to the research objectives noted in Section 1.1.

Chapter	Contributions and research objectives	Model	Simulation	Tool
Part II: Scheduling for Mixed Criticality – O1.1 to O1.3				
3	Evaluation of IEEE 802.1Qav in an avionic scenario		•	
4	Comparison of existing and new packet scheduling algorithms		•	
5	Proposition of a novel packet scheduling algorithm		•	
Part III: Flow-Level Network Modeling – O2.1 to O2.3				
6	Survey of performance evaluation frameworks for TCP flows			
7	Extension of flow-level modeling to Ethernet and cross-traffic	•	•	
	Tool for evaluating Ethernet topologies with flow-level modeling		•	•
Part IV: Short Flows: Sessions and Application Layer Modeling – O2.3 to O2.5				
8	Extension of flow-level modeling to support ON/OFF traffic	•	•	
9	Model for realistic characterization of higher layer protocol	•	•	
	Tool for automatically extracting a model from a trace		•	•
10	Extension of stochastic network calculus	•		
	Application to engineering of Ethernet networks	•	•	

Tab. 1.1: Contributions and structure of this thesis

This thesis is structured into five parts: I) Introduction and Background, II) Scheduling for Mixed Criticality, III) Flow-Level Network Modeling, IV) Short Flows: Sessions and Application Layer Modeling, and V) Summary and Conclusion.

Part I gives a broad introduction about this thesis and the general challenges we are trying to address. **Chapter 1** is the comprehensive introduction to this thesis. The chapter introduces the motivation of this work, the research questions, and the contributions of this thesis. **Chapter 2** introduces the problem area, namely the general properties and requirements (quantitative and qualitative) of Ethernet networks currently used by the aeronautic industry as well as future ones. We also give a survey of current mathematical frameworks proposed in the literature and used by the industry to give guarantees in those networks.

Part II gives an overview of the networks we address in the thesis as well as various packet scheduling algorithms in the context of mixing real-time traffic with best-effort one. **Chapter 3** evaluates a new scheduling algorithm proposed by the IEEE for mixing multimedia traffic with normal best-effort Ethernet traffic. Via discrete event simulations, we compare its performance with existing scheduling algorithms in the context of an avionic network in order to evaluate which scheduling algorithm is the best suited for our application (Research Objectives **O1.1** and **O1.2**). **Chapter 4** introduces a new network architecture used for aircraft cabins and evaluates an algorithm combining priority and fair queuing. We investigate the robustness of the scheduling scheme and how the performances of real-time flows are affected in case of network overload (Research Objective **O1.1**). **Chapter 5** proposes a new packet scheduling algorithm enabling ultra-low-latency real-time traffic. This algorithm is an extension of a well-known fair-queuing algorithm and it takes advantage of knowledge of characteristics of the real-time traffic (Research Objectives **O1.1** and **O1.3**).

Part III of this thesis introduces the question of performance evaluation of elastic and TCP flows. **Chapter 6** gives an overview of TCP and its different variants and surveys different approaches and analytical methods which have been proposed in the last two decades for studying its performances (Research Objectives **O2.1** and **O2.2**). **Chapter 7** formalizes and extends the so-called flow-level network modeling mathematical framework in order to use it on Ethernet networks with low latencies and take into account some non-intuitive behavior of TCP (Research Objectives **O2.2** and **O2.3**). We focus here on infinite flows, namely flows with an unlimited amount of data to transmit. We also propose a tool implementing the proposed mathematical results and compare it to another tool based on a similar model.

Part IV proposes various extensions of the mathematical framework proposed in Chapter 7 in order to evaluate network flows at the application level. **Chapter 8** is a direct extension of the framework presented in Chapter 7 where we propose to study short elastic and inelastic flows with a simple unidirectional ON/OFF behavior (Research Objectives **O2.1** and **O2.3**). Variations of the results are proposed in order to take into account specificities of TCP flows (namely slow-start) as well as bidirectional ON/OFF flows. In **Chapter 9**, we investigate realistic application-level behavior and how it may be modeled based on the analysis of real traffic captures (Research Objective **O2.4**). Tools and methods are introduced in order to extract meaningful characteristics from existing captures made on real network and reproduce those characteristics into discrete-event simulations. **Chapter 10** considers the use of stochastic network calculus to characterize tail performances – also sometimes referred as rare events performances – of short flows instead of just mean performances (Research Objective **O2.5**).

Part V concludes this thesis. **Chapter 11** gives a global overview over the different results presented in this thesis in the general context of network engineering, and compares the contributions with the state of the art. Finally, **Chapter 12** concludes this thesis by summarizing the contributions and discussing future work and open research directions.

1.3 Applicability

Although we target in this thesis existing and future Ethernet architectures used in the aeronautic industry (*i.e.* Ethernet networks with low latency and relatively small – around 10 switches and 100 nodes – as presented in more details in Chapter 2), the methods, analytical frameworks and tools developed in this thesis can be used directly or adapted to other types of networks such as networks with larger delays or larger number of nodes. One global class of network where our contributions may be used is packet switched networks.

1.4 Remarks about this document

Notice

Chapters based on previous publications are indicated so. Some details about the industrial topologies used and evaluated in this work have been intentionally left out due their industrial use.

Abbreviations and notations

The various abbreviations and notations used throughout this thesis are listed in Appendix D. The digital version of this document also contains clickable links which are marked in gray (ex: `http://example.com`). Some notes regarding the origin of the content of this thesis appear in the text in the following form:

Note This is a note

2. AVIONIC NETWORKS: CURRENT AND FUTURE CHALLENGES

This chapter serves as a presentation of the problem area addressed throughout this thesis, namely current and future Ethernet networks used in the aeronautic industry. We first describe how Ethernet packets are handled in switches. We then characterize current and future architectures, requirements and properties of avionic networks. Finally, we have a look at possible future uses of those networks and their associated challenges, which also serves as an additional motivation for the work presented in this thesis.

Structure of this chapter

In Section 2.1, we first give an overview of how general switched Ethernet networks operate and how packets are handled. Secondly, we describe in Section 2.2 the two network architectures which served as a basis for this thesis and their various requirements. Then in Section 2.3, we survey the state of the art of analytical methods for guaranteed communications which have been proposed for our use-case. Finally, we outline in Section 2.4 the type of network and its associated properties we are trying to establish it this thesis.

2.1 Life of a frame inside an Ethernet switch

An Ethernet switch is a device used to link Ethernet devices together (computers or other switches). Its role is to receive frames (or packets) from input ports (in the so-called *ingress* part), and forward them to the correct output ports (in the so-called *egress* part) depending on forwarding or routing rules. Because a switch is shared by multiple devices, it may happen that packets have to wait before being sent due to the transmission of another packet. In this case the switch has to store packet in a temporary buffer until the output port is free to send again. This is the traditional queuing principle.

When multiple packets are waiting for a same output port, a switch has to decide which packet should be next. This process is called *packet scheduling*. The most commonly used scheduling algorithm is the First-Come First-Served (FCFS) service policy. In FCFS, packets are sent following their order of arrival, like in a First-In First-Out queue.

Using switched Ethernet enables also to do Quality-of-Service, namely preferential service can be given to certain packets and thus enabling them to "cut the line" and not wait for their turn in the queue. In order to do so, those packets first need to be identified via specific rules, based on one or more of its fields, such as the IEEE 802.1Q identifier or priority field [5] (also called VLAN ID or VLAN priority), or the MAC source or destination addresses. Most commercial switches follow the Quality-of-Service architecture defined in the IEEE 802.1Q standard [5] which is akin to the DiffServ principle presented in RFC 2475 [62]. This architecture is presented in Figure 2.1, where we differentiate three parts:

Ingress This is where packets arrive in a switch via ingress Ethernet physical interfaces (PHY). At this point, packets will be classified in order to determine their *category*, *i.e.* which kind of Quality-of-Service they are associated with. A category of packets is also called a *class of service*. Once packets are classified, a first ingress scheduling can occur, in order to change some packet fields (*packet marking*), or reduce their bandwidth (via *shaping* or *dropping*). Packets will then be passed to the packet processing unit.

Packet processing This is the central element of a switch where packets are processed in order to determine their egress port(s) via some forwarding rules. In case of Ethernet, the MAC address learning algorithm along a forwarding table lookup is generally used. Other process may also occur in the processing unit, such as the Spanning Tree Protocol (STP) as defined in the IEEE 802.1D [1] standard, or IEEE 802.1X port-based network access control [4].

Egress In this part, packets will first be classified in order to determine their category and placed in the according queue (often also classed *output queue*). There are generally four or eight queues in traditional switches functioning on the First-In-First-Out (FIFO) principle. An output scheduler will then take serve the different queues each time the egress physical interface (PHY) has completed the transmission of a frame or a new frame arrives and the PHY is idle. Common packet schedulers found in Ethernet switches will be described in Section 3.3.

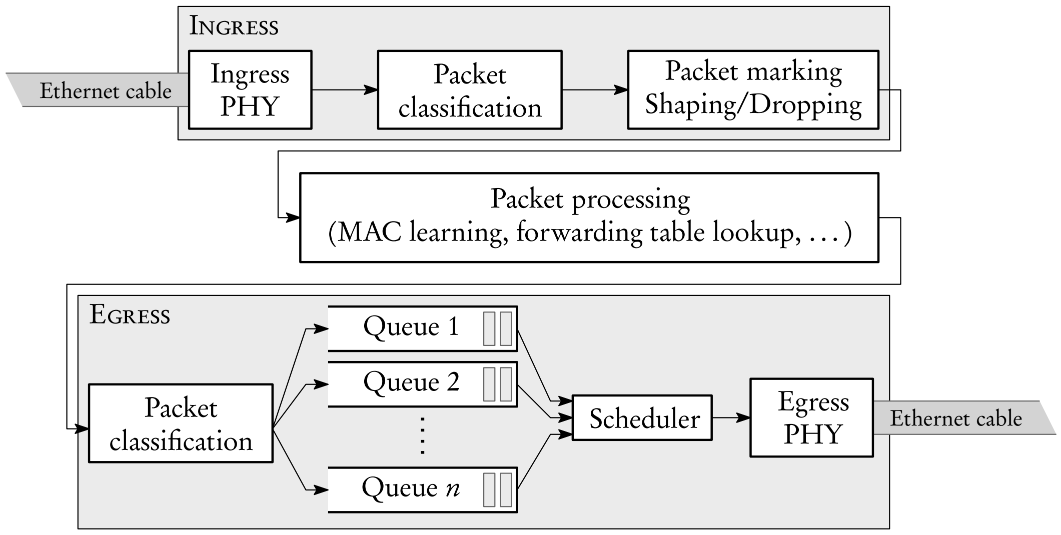


Fig. 2.1: Common Quality-of-Service architecture in Ethernet switches

2.2 Studied networks and their challenges

As demonstrated by surveys on Ethernet and its field of applications from Felser in [107], or more recently from Sommer *et al.* in [224], the usages of Ethernet are quite versatile, without any common denominator regarding size, type of transported traffic, requirements or Quality-of-Service policies. Hence, we shall first specify in this section what we mean by *Ethernet network* in the context of this thesis.

Aircraft manufacturers have to follow various standards and recommendations in order to certify and be allowed to deliver those aircrafts to airlines. As for any other elements of an aircraft, this also applies to onboard networks, and in particular the Ethernet networks we are studying and target in this thesis. The ARINC 636 and 646 standards [20, 21] are good examples of standards aiming to modify existing Ethernet technologies in order for them to be included in aircrafts.

In the aeronautic industry, the different networks and applications they support are classified into four domains, which are sets of devices and related networks which share common safety and security aspects. This classification is detailed in the ARINC 664 Part 5 [22] standard, and it defines the four following aircraft domains:

Aircraft Control Domain (ACD) This domain consists of systems and networks whose primary functions support the safe operation of the aircraft. Example of systems in this domain include: cockpit displays, flight and environmental controls, or electrical and propulsion systems. This domain is generally divided into two sub-domains:

- **Flight and Embedded Control System Sub-domain** This sub-domain is often noted as the *cockpit part*, and it concerns functions related to the aircraft control.
- **Cabin Core Sub-domain (CCS)** This sub-domain provides environmental functions dedicated to cabin operations, such as environmental control, passenger address, smoke detection, etc.

Airline Information Services Domain (AISD) The purpose of this domain is to operate the aircraft and airline administrative information for the cabin and flight-crew. Example of services in this domain include: airborne data loader, maintenance access, or cabin crew information access.

Passenger Information and Entertainment Services Domain (PIESD) This domain provides passenger entertainment and network services with devices already available in the aircraft. Example of services in this domain include: audio and video streaming for in-flight entertainment (IFE), passenger Internet surfing, or passenger flight information system.

Passenger Owned Devices Domain (PODD) This domain concerns devices that passengers may bring on board and which may connect to the airplane network, such as laptops, tablet computers or smartphones. Example of services in this domain include: Internet access, or audio and video streaming.

Each domain hosts various applications, each with their own requirements. Throughout this thesis, we will only look at requirements regarding the time to deliver individual packets, messages (multiple packets forming a logical group), or message exchanges. A brief and non-exhaustive summary of typical requirements and resource needs seen in the industry is presented in Tables 2.1 and 2.2. Although present and important for the safe operation of an aircraft, we will not address other requirements such as security, availability, or hardware-specific needs (electrical noise, vibration and temperature durability, etc.), which are entire fields of research of their own. The *real-time* column in Table 2.1 represents the type of service that is required, and is analogous to the common definitions used in real-time computing, namely:

Hard A packet not respecting the requirements results in a total system failure;

Firm Infrequent requirements violations are tolerable but they may degrade the system's quality of service. If a packet is late, it will be disregarded by the application;

Soft Late packets may still be used, but it will result in a degraded quality of service;

Best effort No requirements regarding Quality-of-Service.

We note that for the rest of this document if we only specify *real-time* without its exact descriptive (*i.e.* hard, firm or soft), we mean real-time in its general sense (*i.e.* it can be hard, firm or soft).

Domain	Application	Real-time	Delay Req.	Jitter Req.
ACD	Flight control	Hard	10 – 100 ms	10 – 100 ms
	Cabin control	Firm	10 – 100 ms	10 – 100 ms
	Data loading	Soft	100 ms – 1 s	100 ms – 1 s
	Interactive audio	Soft	10 ms	10 ms
AISD	Cabin management	Firm	100 ms	100 ms
	Business functions	Soft	100 ms	100 ms
PIESD	Flight information	Soft	100 ms	100 ms
	Audio streaming	Soft	10 ms	10 ms
	Video streaming	Soft	10 ms	10 ms
PODD	Video streaming	Soft	100 ms	100 ms
	Internet access	Best-effort	≥ 100 ms	≥ 100 ms

Tab. 2.1: The four aircraft domains and their typical network requirements

As noted in the description of the different aircraft domains, an aircraft can be split into two parts from a functional point of view:

The cockpit where the primary objective is to control and fly the airplane, which corresponds to the cockpit sub-domain of the ACD where most requirements fall in the low-latency hard real-time category.

The cabin where the primary objective is to service passengers, which corresponds to the cabin sub-domain of the ACD, and the three other domains (AISD, PIESD and PODD) where most requirements fall between best-effort to firm real-time.

Domain	Application	Bandwidth Req. / Flow	Payload Size / Flow
ACD	Flight control	10 kbit/s – 1 Mbit/s	100 – 1000 B
	Cabin control	100 kbit/s – 1 Mbit/s	100 – 1000 B
	Data loading	10 Mbit/s	100 kB – 10 MB
	Interactive audio	10 Mbit/s	100 – 1000 B
AISD	Cabin management	10 Mbit/s	1 – 100 kB
	Business functions	100 kbit/s	1 – 100 kB
PIESD	Flight information	10 – 100 kbit/s	10 – 100 kB
	Audio streaming	1 – 10 Mbit/s	10 – 100 MB
	Video streaming	10 Mbit/s – 1 Gbit/s	10 MB – 1 GB
PODD	Video streaming	10 Mbit/s – 1 Gbit/s	10 MB – 1 GB
	Internet access	100 kbit/s – 10 Mbit/s	10 – 100 MB

Tab. 2.2: The four aircraft domains and their typical network usage per flow

This dichotomy has led to the use and design of two separate networking solutions:

For the cockpit a derivative of Ethernet defined in the ARINC 664 part 7 standard [24], sometimes labeled as *avionic Ethernet*, which will be discussed in more details in Section 2.2.1;

For the cabin a more traditional Ethernet network, which will be discussed in more details in Section 2.2.2.

For simplification purpose, we consider that each part described before is only one single isolated Ethernet network. Although those two networks have some differences as shown later in Sections 2.2.1 and 2.2.2, they share some common attributes, which will define our abstract *typical avionic Ethernet network* throughout this thesis:

- The network is based on Ethernet, as defined in the IEEE 802.3 standard [6]. Frame format follow the standard, as well as their sizes, from 64 B to 1518 B, and are sent with a spacing of 20 B regardless of link speed, corresponding to the preamble, the start of frame delimiter (SFD) and the inter-frame gap (IFG).
- The network is composed of computers connected via switches with full-duplex links. This way we eliminate the collision detection of Ethernet (also known as CSMA/CD – Carrier Sense Multiple Access with Collision Detection) and its potential random delivery of frames, which would prevent the use of Ethernet for real-time data. We consider networks ranging from 10 to 100 switches, with 10 to 1000+ devices connected to them.
- Connections are made via Ethernet cables, with a typical length of 10 m, with a propagation speed of 2×10^8 m/s, with no consideration on the technology (copper or optical fiber). We consider links of the following bandwidths: 10 Mbit/s, 100 Mbit/s and 1000 Mbit/s.

- We consider that there is a unique path from one computer to the other, known by the switches, either via static or dynamic configuration, or automatic discovery.
- Unless specified otherwise, switches work on the *store-and-forward* principle, meaning that in order to forward an Ethernet frame, switches first have to receive the complete Ethernet frame from the sender. Switches may have a so-called *processing delay* before forwarding a frame which typically ranges from $0\,\mu s$ to $100\,\mu s$.
- No particular requirement is imposed on network topology, but we consider that the network is designed or configured in such a way that we do not have loops.
- Current typical networking requirements are presented in Table 2.1. A single network may host one or more avionic domains.

Note that the properties listed here and in Sections 2.2.1 and 2.2.2 have been simplified compared to the reality, but they give a faithful enough description of the Ethernet networks used in today's or in currently developed aircrafts.

2.2.1 AFDX – Avionic Full-Duplex Switched Ethernet

We will now address the solution currently used for the Flight and Embedded Control System Sub-domain which hosts applications mainly related to the aircraft control (*i.e.* functions for the cockpit). As noted earlier, this network has to support the following requirements:

1. The majority of the traffic is considered to be hard real-time, meaning that if a packet is not delivered in time, or dropped in the network, it could lead to a system failure (with possible fatal consequences).
2. Requirements on delay, jitter and packet drop have to be proven with a formal method for certification purpose.
3. In case of failure of one system in the network, it should not disturb the other systems.
4. Ethernet frames end-to-end delay and jitter requirements are in the order of 10 ms to 100 ms.
5. Individual flow bandwidths are relatively small, in the order of 10 kbit/s to 1 Mbit/s.

Because of the first three points, we see that standard Ethernet cannot be used, due its best-effort packet delivery scheme. Note that while other existing technologies than Ethernet have been used in the past (CAN bus – Controller Area Network, ARINC 429, RS-232, FlexRay, etc.), they often do not offer enough bandwidth, are costly due their narrow use or proprietary technology, or lead to large and heavy networks. We refer to the review from Muñoz-Castañer *et al.* [193] for a larger survey and history of the different data buses used in aircrafts. This need for a high-speed low-latency safe network has led to the development in the earlier 2000's of the ARINC 664 standard [24] – also more commonly named Avionic Full-Duplex Switched Ethernet or AFDX – a modified Ethernet tailored to the requirements listed earlier.

One important property hinted in the requirements is that AFDX has to be *deterministic*. In the avionic world, a network is called deterministic if it fulfills the three following properties[1]:

Performance bounds Packets end-to-end delay and jitter have to admit a formal upper bound.

Loss prevention Formal verification that packets are not dropped due to buffer overflow.

Enforcement The network has to enforce that those properties are respected.

The solution which was proposed in AFDX to achieve those properties was to introduce the concept of *Virtual Link* or VL. A VL effectively depicts a tunnel where Ethernet frames have to be sent according to the following parameters:

1. Frames are sent by a single sender (unidirectional communication).
2. Frames are received by one or multiple receivers (multicast principle).
3. The path of the frames between emitter and receiver is predefined and static.
4. Frame sizes have to be between two extrema, often noted s_{min} and s_{max}.
5. The minimum time gap between the transmission of two consecutive frames cannot be smaller than a prescribed time, often referred as *Bandwidth Allocation Gap* or BAG, which is a power of 2 between 1 ms and 128 ms.

Using this concept of VL, various properties can be derived. First, by encapsulating every communication of the network in VLs, the formal verification of packet performances is made easier. The formal method which is most commonly used is the so-called *deterministic network calculus*, which will be described in Section 2.3.1. Secondly, switches can effectively verify that packets respect their specified VL parameters and drop them in case they do not comply, in order to prevent well-behaved traffic from being disturbed. Finally, because this relatively simple encapsulation has been shown to be sufficient regarding latency requirements, no other mechanism such as clock synchronization is necessary.

Note that due to the scope of this thesis, which is the Quality-of-Service of Ethernet frames or higher layer messages, we do not delve longer on the ARINC 664 standard. Apart from the concept of Virtual Link, the main notable differences with standard Ethernet which can be noted are a specific redundancy mechanism and special physical connectors for airborne applications.

Example of industrial AFDX architectures

While actual AFDX topologies and architectures are not openly available, some information can still be found in the literature, especially in the works from Grieu in [128], Charara *et al.* in [86], Scharbarg *et al.* in [215] or Bauer in [51]. From those publications and our own experience with AFDX networks, we note that AFDX networks are generally small networks

[1] Note that there are other definitions of *deterministic* used in other industries which can be more or less strict than the one noted here.

with around 10 switches organized in a partially meshed topology with about 100 end-systems communicating via around 1000 Virtual Links. We present in Figure 2.2 a short summary of the reference AFDX architecture presented by Bauer in [51, Section 2.2.4].

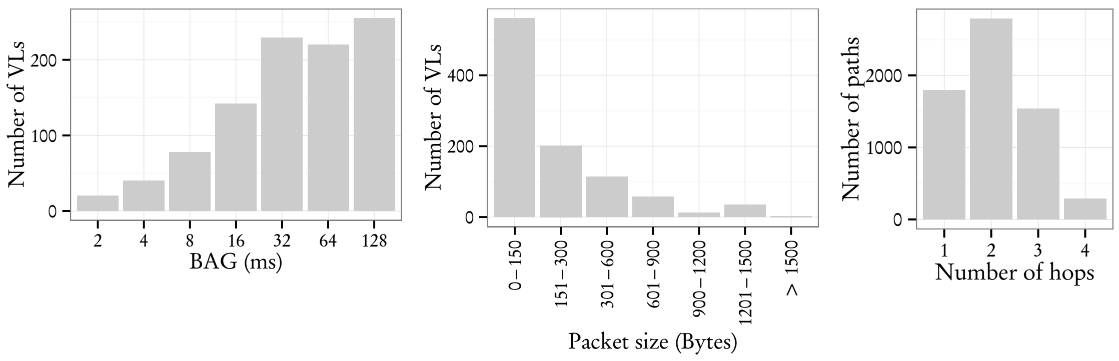


Fig. 2.2: Virtual Links reference values for an industrial AFDX network based on data from [51, Section 2.2.4]

Additional details about this topology and practical use-cases will be given in Chapter 3.

2.2.2 Ethernet-based cabin networks

We will now address the network solution that is currently being developed for the cabin. This network has to support applications from the Cabin Core Sub-domain (CCS) and the three domains related to passenger applications (AISD, PIESD, PODD). This network we present in this section is a possible future extension of networks currently based on ARINC 628 and 746 standards [23, 25]. As noted earlier, this network has to support the following requirements for the different domains:

CCS For the Cabin Core Sub-domain, which corresponds to safety applications of passengers (passenger audio announcements, smoke detection, lighted signs, ...), we have the following requirements: *(i)* this traffic is considered to be firm or soft real-time, meaning that if a packet is not delivered in time or dropped in the network, it will only degrade the system's quality of service; *(ii)* some requirements on delay, jitter and packet drop have to be formally proven; *(iii)* Ethernet frames end-to-end delay and jitter requirements are in the order of 10 ms to 100 ms, especially for interactive audio traffic; *(iv)* individual flow bandwidths are relatively small, in the order of 10 kbit/s to 1 Mbit/s.

AISD For the airline domain, we have the following requirements: *(i)* the traffic is considered soft real-time or best-effort depending on the application; *(ii)* end-to-end delay and jitter requirements are in the order of 100 ms; *(iii)* individual flow bandwidths are relatively small, in the order of 1 kbit/s to 100 kbit/s.

PIESD For the passenger infotainment domain, corresponding to flight information display, audio and video streaming, we have the following requirements: *(i)* no requirements regarding real-time behavior; *(ii)* end-to-end delay and jitter requirements are in the order of 100 ms; *(iii)* individual flow bandwidths ranging from relatively small (1 kbit/s to 100 kbit/s) for flight information display, to very large (10 Mbit/s to 1 Gbit/s) for audio and video streaming depending on the quality requested.

PODD For the passenger owned devices, generally corresponding to Internet browsing and sometimes audio or video streaming, we consider that it works on a best-effort basis, although we might want to upgrade this to better Quality-of-Service if users are paying for this service.

A proposal for a new cabin architecture

Note This section is based on our previous work [122] published in *Proceedings of the 4th International Workshop on Aircraft System Technologies*, 2013.

Due to the range of applications, their mixed requirements, and mixed types of flows that are covered by the cabin, each domain (or even application) used to have its own dedicated network with its own harness and network technology. This engineering approach – called Federated Avionics – used to be popular, but has now reached its limits, because of its price, weight, often proprietary and outdated solutions, as well as difficulties to interface the different applications with each other.

This has led to recent developments in the last years of the concept of a single large Ethernet network supporting all functions and domains of the cabin. While going from multiple isolated networks to a single unified one has its advantages in term of price and weight, it also comes with some drawbacks. Indeed, all the requirements previously cited still have to be fulfilled. Some important challenge is that the network still has to ensure that safety functions work property while the network is also used by devices which are not under the control of the network architect, namely passengers owned devices.

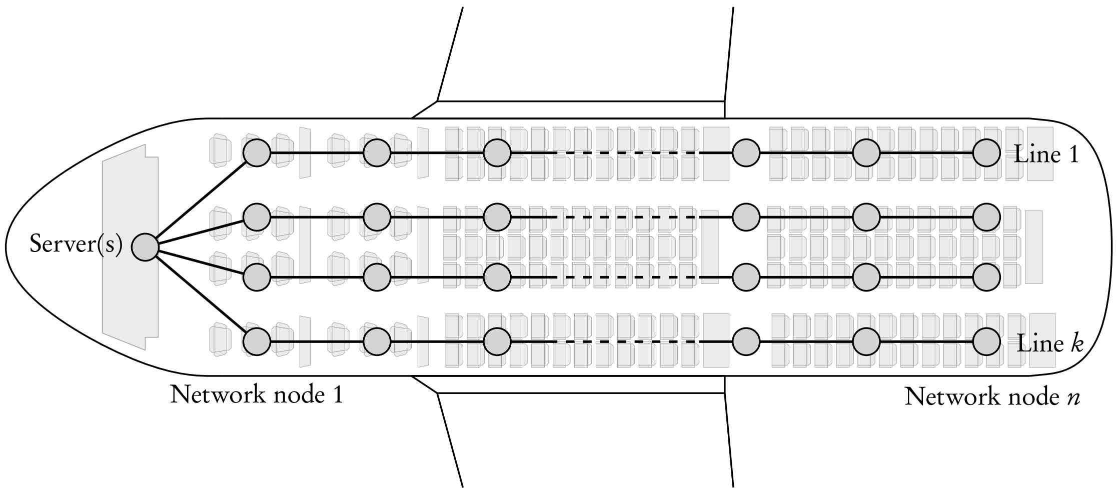


Fig. 2.3: Star/daisy chain cabin network topology

Regarding some characteristics of the network, we note that due to the physical layout of a cabin, a star/daisy chain topology as presented in Figure 2.3 is used. In this cabin topology, servers are located at the front of the airplane, and connected to multiple Ethernet daisy chains (generally in the order of 10) – referred as *lines* – which cover the cabin from front to back. The daisy chain is made of specific Ethernet switches where the cabin devices can be connected to, as presented in Figure 2.4. Additional details about this topology and practical use-cases will be given in Chapters 4 and 5.

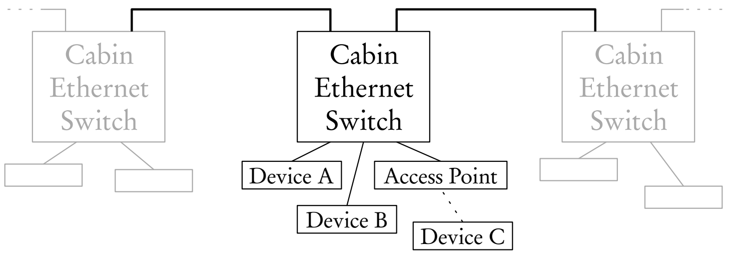


Fig. 2.4: Daisy chain cabin network topology

2.3 State of the art on performance bounds in avionic networks

As noted in Sections 2.2.1 and 2.2.2, some aircraft applications are hard real-time, where the timing properties of the network have to be demonstrated via formal methods. We address in this section various mathematical formalisms which have been proposed in the literature – and sometimes adopted by the industry – for studying avionic networks and determining end-to-end delay performances bounds.

Figure 2.5 presents the basic notions regarding end-to-end delay analysis, where we describe the three following performance indicators:

The maximal observed delay corresponds to the maximal delay which is measured on the real network during its normal operation, and can also be approximated using simulations of the network. We note that this delay depends on a lot of factors, such as the measurement duration or initial state of the network. This delay does not constitute a bound as there is no mathematical proof of its value.

The exact worst-case corresponds to the theoretical worst case delay which can actually occur in case the elements of the network behave within their limits, but in a very specific pattern leading to this worst-case. As showed in Figure 2.5, the actual worst-case generally occurs with a really small probability, which explains the gap with the maximal observed delay. Analytically determining this worst-case is generally challenging, mainly due to the state-space explosion.

The upper bound corresponds to the bound calculated by an analytical model, which is generally larger than the actual worst case due to approximations, simplifications or shortcomings of the formal method. Although this upper bound usually gives an overestimation of the actual worst-case, it is typically easier and more practical to determine this bound than the worst-case. Upper bounds and their associated analytical model are usually characterized by their *tightness*, *looseness* or *pessimism*, which corresponds to the size of the gap between the upper bound and the actual worst-case. A method giving small gaps is called *tight*, while one with a large gap is called *loose* or *pessimistic*.

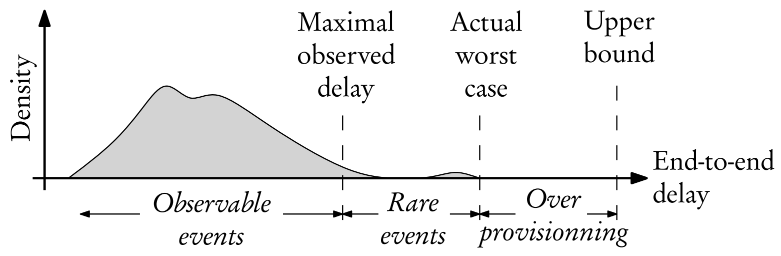


Fig. 2.5: Basic notions regarding end-to-end delay analysis

2.3.1 Deterministic guarantees with network calculus

Deterministic network calculus (DNC) – or often simply called network calculus – is a mathematical framework for analyzing performance guarantees of traffic flows in queuing networks. This formalism was initially developed in the early 1990's by Cruz [95, 96], with the so-called (σ, ρ)-calculus. Its current predominant application is computer networks. We will describe here some of the main mathematical results of this framework, and make a parallel with its current application in AFDX. For a more thorough description of network calculus, we refer to the books from Chang [83] or from Le Boudec and Thiran [168]. In this framework, flows are described as their cumulative arrival of data per unit of time. This is modeled by a non-decreasing function of time (noted t) into the set of monotonically-increasing and strictly positive functions $\mathcal{F}$, more formally defined as:

$$\mathcal{F} = \{f : \mathbb{R}^+ \to \mathbb{R}^+ \mid \forall 0 \le t \le s : f(t) \le f(s), f(0) = 0\} \tag{2.1}$$

This function is called its *cumulative arrival function*. A flow with cumulative arrival function R, or more simply a flow R, is said to be (σ, ρ)-upper constrained if ([83, Definition 1.1.1]):

$$R(t) - R(s) \le \rho \cdot (t - s) + \sigma, \text{ for all } 0 \le s \le t, \text{with } \rho, \sigma \text{ constant values.} \tag{2.2}$$

In Equation (2.2), σ is called the burstiness parameter, and ρ the upper bound on the long-term average rate of the traffic flow. This curve is illustrated in Figure 2.6 with the gray curve. In a real network, a flow can be forced to follow a prescribed (σ, ρ) constraint using a so-called *shaper*.

More generally, a flow R is said to have a deterministic arrival curve $\alpha \in \mathcal{F}$ if its cumulative arrival function R satisfies for all s and t such that for all $0 \le s \le t$ ([168, Definition 1.2.1]):

$$R(t) - R(s) \le \alpha(t - s) \tag{2.3}$$

In non-mathematical words, it defines the worst-case behavior of a flow by a well-known function. The AFDX Virtual Link principle is a direct application of this, such that α can be defined as:

$$\alpha(t) = s_{max} + \frac{s_{max}}{BAG} \cdot t, \text{ for all } t \ge 0. \tag{2.4}$$

Network elements representing queues or switches, often called *servers* in network calculus terms, impose a service curve $\beta \in \mathcal{F}$ on an input flow R, such that the output flow R^* is defined by ([168, Definition 1.3.1]):

$$R^*(t) \ge \inf_{0 \le s \le t} \{R(t - s) + \beta(s)\} \tag{2.5}$$

The operation on the right hand-side of the inequality is known as the min-plus convolution, and is part of the min-plus algebra used in network calculus. The min-plus convolution is noted as $\otimes$, such that $R^*(t) \geq (R \otimes \beta)(t)$ ([168, Definition 3.1.10]). The second operation of the min-plus algebra is the deconvolution, noted $\oslash$, which is defined as ([168, Definition 3.1.13]):

$$(f \oslash g)(t) = \sup_{0 \leq s}\{f(t+s) - g(s)\} \tag{2.6}$$

In case of an Ethernet interface with link speed C and delay δ, the service curve β can be expressed as:

$$\beta(t) = C[t-\delta]^+, \text{where } [x]^+ = \max(0,x) \tag{2.7}$$

This particular affine curve is called a rate-latency service curve, and it is illustrated in Figure 2.6 with the black curve.

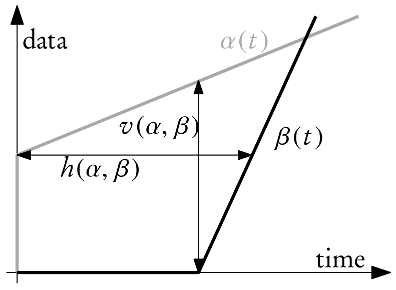


Fig. 2.6: Latency ($h(\alpha,\beta)$) and buffer ($v(\alpha,\beta)$) bounds in deterministic network calculus

Using this formalism, two performance bounds can be derived, as presented in Figure 2.6:

The backlog or queue size bound, which corresponds to the maximal vertical deviation between the arrival and service curve $v(\alpha,\beta)$, or in mathematical terms ([168, Theorem 1.4.1]):

$$R(t) - R^*(t) \leq \sup_{s\geq 0}\{\alpha(s) - \beta(s)\} = (\alpha \oslash \beta)(0) = v(\alpha,\beta) \tag{2.8}$$

The delay bound which corresponds to the maximal horizontal deviation between the arrival and service curves $h(\alpha,\beta)$, or in mathematical terms ([168, Theorem 1.4.2]):

$$R^*(t) - R^*(t-s) \leq \sup_{t\geq 0}\left\{\inf_{s\geq 0}\{\alpha(t) \leq \beta(t+s)\}\right\} = h(\alpha,\beta) \tag{2.9}$$

One strong property of network calculus is the so-called *concatenation*, where a large network of servers can be simplified to a single server using the min-plus convolution. A simple example is a flow traversing two servers with respective service curves β_1 and β_2. It can be shown (see [168, Theorem 1.4.6]) that this is equivalent to a flow traversing a single server with service curve $\beta_C = \beta_1 \otimes \beta_2$, and the resulting bounds will be tighter than an analysis done with β_1 and β_2 separately.

Regarding packet scheduling as presented later in Section 3.3, network calculus can be used on networks with various algorithms such as priority-based scheduler (see the book from Le Boudec and Thiran [168, Chapters 2.4, 6 and 7]) or fair queuing (see the work from Stiliadis and Varma in [228]).

One of the pitfalls of network calculus is its looseness, which is generally attributed to a loose handling of flow multiplexing. Various methods have been proposed to address this issue, such as the *Pay Burst Only Once* (see book from Le Boudec and Thiran [168]) and *Pay Multiplexing Only Once* (see work from Schmitt *et al.* [218]).

We also note that due to the way flows are modeled, elastic protocols such as TCP are hard in practice to use with this framework. Various propositions were made regarding using network calculus on feedback-based protocols such as TCP, such as for the instance the works from Chang [82], from Baccelli and Hong [38], or from Agrawal *et al.* [27]. We note that those previous work are either limited to the study of a single flow, or use an idealized version of TCP.

Details about how network calculus can be applied to Ethernet networks and how it has been used during the development of the A380 in the early 2000's are presented by Grieu in [128]. This work has led to the definition of the AFDX standard [24].

Various tools are available for the performance evaluation of network with network calculus, open-source ones such as the DISCO Network Calculator from Schmitt and Zdarsky [217], CyNC from Schioler *et al.* [216], COINC from Bouillard *et al.* in [65], or NC-maude from Boyer [68]; or closed-source and commercial ones targeted at the industry [69].

2.3.2 Stochastic guarantees with stochastic network calculus

We have seen earlier that the theory of network calculus is able to give deterministic guarantees on delays experienced by frames. While this is a useful property for hard real-time flow such as AFDX traffic, it is not always required for other types of flows such as audio or video streaming where a few late or lost packets can be tolerated. For such flows, stochastic guarantees can be useful as they lead to a better network utilization. Those stochastic guarantees can be expressed as:

$$\Pr(\mathit{delay} \geq \mathit{bound}) \leq \epsilon, \tag{2.10}$$

$$\Pr(\mathit{queue\ size} \geq \mathit{bound}) \leq \epsilon, \tag{2.11}$$

with ϵ the probability that the bound is violated.

The idea of giving stochastic guarantees is not incompatible with an aeronautic usage and its requirements, as demonstrated by the safety level classification defined in the ARP-4754A recommendations [210], which defines so-called *Design Assurance Level* or DAL. Those DALs characterize the severity of the consequence of a failed application, as well as the acceptable probability of occurrence per flight hour. The different DAL are presented in Table 2.3. Similar levels have been standardized in other industries (automotive, nuclear, industrial process, ...) in the IEC 61508 standard [232]. From Table 2.3, we determine then that we should target values of ϵ in Equations (2.10) and (2.11) in a range between 10^{-3} and 10^{-9}, depending on the level of guarantees wanted.

The idea illustrated in Equations (2.10) and (2.11) is where stochastic network calculus (SNC) enters into play. Using analogous formalisms than its deterministic cousin, stochastic guarantees can be given on networks. Due to its relative youth and as it is still an active field of research, various versions and iterations of stochastic network calculus have been proposed. We can cite the work from Liu *et al.* in [176], Starobinski and Sidi in [227], Chang in [83],

DAL	Probability	Severity in case of failure
A	$\leq 10^{-9}$	*Catastrophic*: Failure may cause a crash. Error or loss of critical function required to safely fly and land aircraft.
B	10^{-7} to 10^{-9}	*Hazardous*: Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.
C	10^{-5} to 10^{-7}	*Major*: Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries) or significantly increases crew workload.
D	10^{-1} to 10^{-5}	*Minor*: Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change).
E	$\geq 10^{-1}$	*None:* No effect on aircraft operational capability or pilot workload.

Tab. 2.3: Design Assurance Level definitions and their associated acceptable probability of occurrence per flight hour (adapted from [210])

Vojnović and Le Boudec in [236], Fidler in [108] or also Jiang and Liu in [152] for various iterations and formalization of SNC. We will present in the rest of this section the so-called $(\sigma(\theta), \rho(\theta))$-calculus, as presented by Chang in [83].

In stochastic network calculus, a flow is represented by a sequence of non-negative, real random variables $(a_n)_{n\in\mathbb{N}}$. As for deterministic network calculus, we look at the cumulative arrival of data, which for up to the (discrete) time n is represented by:

$$A(n) = \sum_{i=1}^{n} a(i) \tag{2.12}$$

Stochastic network calculus makes use of the notion of *moment-generating function* of a random variable X, which is defined as:

$$M_X(\theta) = \mathbb{E}\left[e^{\theta X}\right], \forall \theta \in \mathbb{R} \tag{2.13}$$

A flow is said to be $(\sigma(\theta), \rho(\theta))$-constrained for some θ, if the following moment-generating function exists and is bounded by ([83, Definition 7.2.1]):

$$\frac{1}{\theta} \log \mathbb{E}\left[e^{\theta(A(n)-A(m))}\right] \leq \rho(\theta)(n-m) + \sigma(\theta), \text{for all } 0 \leq m \leq n \tag{2.14}$$

This is stochastic variant of Equation (2.2) in deterministic network calculus.

A server can either be represented as work conserving links with constant rate C, or as a stochastic process similarly to Equation (2.14). If a flow A traverses a server with constant rate C, the queue length will then have the following probabilistic bound as in Equation (2.11) ([83, Lemma 7.4.1]):

$$\Pr(Q(t) \geq x) \leq \beta(\theta) e^{-\theta x} \tag{2.15}$$

where

$$\beta(\theta) = \frac{e^{\theta\sigma(\theta)}}{1 - e^{\theta(\rho(\theta)-C)}} \tag{2.16}$$

The expression of the bound on the delay $D(t)$ can then be derived using Equation (2.15) and the relation (see [154]):

$$D(t) = \left\lceil \frac{Q(t)}{C} \right\rceil \tag{2.17}$$

for a single server with constant rate C.

As for deterministic network calculus, stochastic network calculus also offers a concatenation property which enables the reduction of large networks to a single node, as highlighted by Ciucu *et al.* in [91]. We refer to the books from Chang [83] and Jiang and Liu [152] for a more complete overview of stochastic network calculus, as well as the work from Mao and Panwar in [180] and from Fidler in [109] for a survey on envelopes and service curves which have been proposed in the literature.

Regarding packet scheduling algorithm as presented later in Section 3.3, stochastic network calculus has also been used to study various algorithms. We refer to the work from Jiang *et al.* in [154] for a study of priority and fair-queuing, or the work from Liebeherr *et al.* in [172] for a more general class of schedulers.

Among the pitfalls of stochastic network calculus, is its relative complexity compared to deterministic network calculus especially for numerical evaluations. Secondly, depending on which version of the calculus is used, one initial assumption of the formalism is the independence property of the random variables used, which is hard to prove in practice due to similar hardware or protocol synchronization for instance. Finally, modeling of feedback based protocols such as TCP using stochastic network calculus is still an open research point, as highlighted by Jiang and Liu in [152, p. 216].

Regarding an application of this calculus on real networks, Scharbarg *et al.* used this framework on an AFDX network in [215], using the formulation of stochastic network calculus from Vojnović and Le Boudec [236, 237]. They concluded that the bounds calculated with stochastic network calculus, with a probability of violation $\epsilon = 10^{-6}$ on a network with a load under 15 %, was at most about four times the actual upper bound seen in an industrial network. Despite the body of work on this theory, stochastic network calculus has not been adopted yet in the industry, mainly due to the pitfalls aforementioned.

To the best of our knowledge, there is currently only one open-source tool available for the numerical evaluation of stochastic network calculus, the DISCO Stochastic Network Calculator, proposed by Beck and Schmitt in [54], which is based on the formulation from Chang in [83] and Fidler in [108].

2.3.3 Other methods

Two other formal methods have been proposed for guaranteeing latencies in avionic networks. As those methods are less prominent in the aeronautic industry, we detail them in lesser extent than the deterministic and stochastic network calculus.

Model checking with timed-automata

Verification using *model checking* (MC) has been proposed for the analysis of Ethernet networks, where systems are generally represented as finite state automata. This technique, based on timed-automata (see for instance work from Alur and Dill in [34]), was proposed for AFDX networks by Charara *et al.* in [86] and [85]. Although this technique is able to give the exact worst-case, it was showed that it is limited to very small networks because of the combinatorial state-space explosion.

Trajectory approach

A formalism known as *Worst-Case Execution Time* (or WCET), initially developed to give bounds on the execution of programs on a processor, has been adapted give deterministic bounds on network delays. This extension is sometimes denoted *trajectory approach* (TA). We refer to the work from Wilhelm *et al.* [244] for a general survey on the methods and tools for WCET analysis, and to the work from Migge [189] and Martin and Minet [181, 182] for the initial works on the trajectory approach. This method was proposed and extended by Bauer *et al.* in [53] and [51] in the case of AFDX networks, where the bounds were shown to be better in average than the one produced by deterministic network calculus.

2.3.4 Conclusion and outlook on current approaches

We have detailed in Section 2.3 the various methods which have been used to give formal guarantees on packets in avionic networks. Table 2.4 presents a comparison between the different methods, with their various advantages and disadvantages. Going back to Section 2.2, we notice that all the presented methods target primarily the ACD domain, meaning flows with hard real-time requirements, and with a smaller extent, the audio and video streaming applications of the PIESD domain.

Method	Advantages	Disadvantages
DNC	Efficient on large networks Well-tried industrial use Relatively easy use	Can lead to loose bounds Limited types of traffic
SNC	Tighter bounds than DNC Broad range of traffic classes Accounts for statistic multiplexing	More complex theories than DNC Difficulties with numerical evaluation Independence assumptions on flows
MC	Exact worst-case results Detailed modeling of the system	Limited to small networks
TA	Tighter bounds on average than DNC Generation of worst-case scenarios	Limited types of traffic

Tab. 2.4: Short comparison of the different available methods which have been used for the study of packet end-to-end delay guarantees in AFDX networks

We also note that the various methods presented in Section 2.3 generally focus only on per-packet guarantees, *i.e.* the link layer. While such guarantees are required for certain applications, they do not really map to higher layers with other requirements than delay bounds on packets. This is especially the case for elastic traffic, where instead of per-packet guarantees, we might be interested in per-flow guarantees. This realization serves also as an additional motivation for Parts III to IV.

2.4 Towards the ideal Ethernet and its guarantees

We have seen in Sections 2.2.1 and 2.2.2 that we have two quite different network solutions, each with its own sets of requirements, ranging from hard real-time to best-effort, flows with low end-to-end delay requirements and small bandwidth needs, to flows with large bandwidth needs and larger end-to-end delay requirements. Historically, the standard method was to separate all those applications in different physical networks using different open or proprietary protocols, which lead to high weight and complex architectures when those different applications and networks have to be interconnected.

In order to reduce weight, cost and complexity, the current trend is to group those networks, use more widely available standards and technologies, and use Commercially-Off-The-Shelves (COTS) components. For instance performance evaluations of Gigabit COTS Ethernet switches for avionic networks were performed by Meier *et al.* in [187] as well as Jacobs *et al.* in [146]. Both studies concluded that the performances in term of latency and jitter are sufficient for avionic applications. A similar study has been made more recently by Suen *et al.* in [230], where they focused on the ability to complete the interchange of message between nodes in the system. They also conclude that COTS components provide performances within the range of avionic functions. In case those solutions do not fit all requirements, another affordable alternative is to use Field-Programmable Gate Array (FPGA) based solutions, where packet processing is done in hardware. As highlighted in the recent work from Carvajal *et al.* in [78], such solution can achieve better service classification than COTS switches and help reduce the end-to-end latency of real-time traffic.

The broader goals currently investigated by network designers – and also main motivation of this thesis – are then:

- *Could we have a single Ethernet network for the whole aircraft, with mixed traffic requirements regarding Ethernet frames, as well as other types of requirements?*
  Part of this question will be addressed later in Part II.

- *Instead of focusing on requirements on Ethernet frames, could we give guarantees for higher layer protocols?*
  This question will be addressed later in Parts III to IV.

Those questions are not only present in the aeronautic industry, but also in other ones, such as the automotive or automation industries, the professional audio/video industry, or even in data centers. This is demonstrated for instance by the recent proposition from the IEEE organization in 2010 – 2011 of a set of standards dedicated to mix audio and video interactive traffic with best-effort traffic in Ethernet networks. This set of standards is currently marketed under the name *Audio/Video Bridging* (AVB) [12]. We will evaluate part of this set of standard in Chapter 3 to see if it is applicable in an avionic use-case. More recently the

IEEE moved toward a broader scope than audio and video streaming on Ethernet networks, namely general guarantees on network flows with industrial applications. This technology is currently marketed under the name *Time Sensitive Networking* (TSN) [11] and Distinguished Minimum Latency Traffic in a Converged Traffic Environment (DMLT) [10], which both aim at standardizing a way to prove guaranteed latency and bandwidth on Ethernet networks with a mix of best-effort traffic.

Part II

SCHEDULING FOR MIXED CRITICALITY

3. A SOLUTION FROM IEEE FOR MIXING REAL-TIME AND BEST-EFFORT TRAFFIC

Note This chapter is based on our previous publication [119], published in *Proceedings of the 18th IEEE Symposium on Computers and Communications*, 2013. Compared to the original publication, Section 3.3 was extended, and Figures 3.6 and 3.8 and their associated interpretations in Section 3.5 were added.

3.1 Introduction

We described in Section 2.4 an ideal Ethernet network where link capacities should be shared between real-time traffic and best-effort one. We noted that such network is not only of interest for the aeronautic industry, but also for other industries. One notable example and its associated solution is the professional audio/video industry, which contributed to the creation of the IEEE Audio/Video Bridging (AVB) Task Group [12] and its set of standards published in 2011 and 2012. This set of standards describes various building blocks which enable high quality audio and video streaming on an Ethernet network: high accuracy clock synchronization for synchronized stream playback, transport protocol for multimedia streams, devices and streams discovery, automatic resource reservation on the network, and packet scheduling for multimedia streams.

The aim of this chapter is to evaluate if this solution from IEEE could be of benefit for our use-cases. One standard of interest is the IEEE 802.1Qav [2], which describes a packet scheduling policy for mixing audio or video streaming flows with traditional Ethernet best-effort traffic. This standard aims at providing some important properties for multimedia flows, specifically for packets scheduling:

- A low and bounded latency for multimedia packets;
- A separation between multimedia and best-effort flows;
- A high-priority class with bounded bandwidth.

We note from those properties a clear parallel with our initial goals set in Section 2.4.

The contribution of this chapter is our comparison of the IEEE 802.1Qav scheduling architecture with the schedulers presented in Section 3.3 using a performance evaluation of an industrial AFDX network with discrete-event simulation. We evaluate the architecture by adding so-called best-effort Virtual Links with the purpose of increasing the utilization of the network and see how the performance of the initial Virtual Links is impacted. We will try to answer the following questions:

1. Which common packet scheduling algorithm could be used for mixed-criticality traffic? (Research Objective **O1.1**)
2. How does the IEEE 802.1Qav scheduling architecture compare against common schedulers in term of performances? (Research Objective **O1.2**)
3. Could it be a solution for mixing real-time with best-effort traffic? (Research Objective **O1.2**)

Structure of this chapter

In Section 3.2 we present similar research studies. Section 3.3 proposes an overview of the most common packet scheduling algorithms found in COTS switches. Section 3.4 delves more into the algorithms of the IEEE 802.1Qav scheduling architecture. In Section 3.5, we describe our simulation model and the results of our performance evaluation. Finally, Section 3.6 summarizes and concludes this chapter.

3.2 Related work

Previous work was already performed on a possible evolution of AFDX toward a backbone supporting avionics flows as well as other traffic classes. Bauer *et al.* presented in [52] a first comparison on a network supporting two traffic classes. Strict Priority Queuing (SPQ), Weighted Fair Queuing (WFQ), Worst-case Fair Weighted Fair Queuing (WF^{2}Q) and single queue First-In First-Out (FIFO) were compared using discrete-event simulation. They concluded that WFQ and WF^{2}Q were good candidates to prevent the starvation problem from SPQ, and also recommended the use of a hybrid priority/fair-queuing scheduler (as we will investigate later in Chapter 4). Zhang *et al.* proposed a similar study of FIFO, SPQ and WFQ in [253] but used deterministic network calculus as a performance evaluation framework. They concluded that using SPQ in both switches and end-systems brings the best performances in term of latency. Zhang *et al.* presented a study of the Earliest Deadline First algorithm (EDF) applied to AFDX in [251] by using Petri nets for their performance evaluation. They concluded that EDF scheduling satisfies the requirements of an AFDX network, although their results lack a comparison with other scheduling algorithms. From a practical perspective, EDF scheduling also requires more resources and configuration than the more traditional packet scheduling algorithms presented in this chapter. Finally, Hua and Liu proposed in [143] a study on Deficit-Round-Robin (DRR) on three traffic classes (avionic, multimedia and data) in an AFDX network using deterministic network calculus and simulations. They showed that DRR can achieve better end-to-end delay than SPQ in the case of multimedia and data flows but at the cost of slightly larger delays for avionic flows. They also proposed an interesting method of how to derive the quantum parameter used in DRR.

Regarding AVB, Imtiaz *et al.* studied in [144] the performances of the IEEE 802.1Qav forwarding method, and showed that worst-case delays can be computed for traffic classes using this new scheduler. A similar approach is also reported in Annex L of the IEEE 802.1Qav standard. In the context of deterministic network calculus as presented in Section 2.3.1, the scheduling architecture of IEEE 802.1Qav was first studied by Queck in [203] where a simple input curve was proposed and applied to an automotive scenario. This work was later formalized and extended by De Azua and Boyer in [97], where alternative input curves where proposed.

In a more general view of AVB, Imtiaz *et al.* also investigated the usability of those new standards in industrial environments in [145]. They proposed an approach of how to integrate industrial process data communication into AVB transport mechanisms, and provided a proof of concept using on a real implementation. More recently, Lim *et al.* carried out a performance evaluation of the AVB standards in an automotive scenario using discrete-event simulation in [173]. They showed that it is a viable solution for multimedia and driver assistance applications where a high synchronization accuracy is required, but it is not adapted for flow with ultra-low end-to-end requirements needed in the automotive industry (100 μs end-to-end latency requirements).

3.3 Common packet scheduling algorithms

Note This section has been extended compared to the original publication [119]. Because the schedulers described here are also used in the rest of this thesis, we give more thorough descriptions and illustrations of the various algorithms.

Scheduling algorithms has attracted a lot of researchers since the early days of computers architectures and networks. Those algorithms are answers to some optimization problem which arise with application requirements, such as reduced latency or jitter, minimum or maximum bandwidth allocation.

Despite the large body of work on the topic, current COTS Ethernet switches are generally based around two families of algorithms – priority scheduling, fair queuing, or a mix of both – which will be described later. Ethernet packet schedulers are typically *work conserving* (as defined by Kleinrock in [161, Section 3.4]), meaning that they are idle only when all the queues they serve are empty.

3.3.1 Priority scheduling

Priority scheduling – also called *Strict Priority Queuing* (SPQ) – is a work-conserving scheduling algorithm where each queue is assigned a priority. It works as following: all queues are polled in their priority order, until a non-empty queue is found and served. This means that a queue can be served only if all higher priority queues are empty.

The main advantages of this algorithm are: *(i)* thanks to its simplicity, it can be easily modeled and deterministic guarantees can be given to all the classes of service; *(ii)* if the optimization parameters for one class of service is the lowest possible latency and jitter, it is the best algorithm for this task. The main disadvantage is the phenomenon called *starvation*, where lower priority queues might get little to no service if higher priority classes always have packets to send. This problem can occur in case of misconfiguration or misbehaved sources.

3.3.2 Fair queuing family

In the fair queuing (FQ) family of scheduling algorithms, the idea is to distribute the available bandwidth – noted here C – fairly among the different served queues. In a simplistic view, it means that if there are N queues to serve, each queue will receive a bandwidth of $^C/_N$. As this family of algorithms is work-conserving, if one queue uses only a bandwidth of b with $b < {}^C/_N$, the remaining bandwidth will be distributed fairly among the other queues, such that they will receive: $^C/_N + \frac{^C/_N - b}{N-1}$. This process is repeated recursively until all the available bandwidth is distributed. Additionally, weights may be attributed to the queues, noted w_1 to w_N, such that queue i will receive a bandwidth of:

$$\frac{w_i}{\sum_{k \in \Psi(t)} w_k} \cdot C \tag{3.1}$$

with $\Psi(t)$ the set of non-empty queues active at time t. This principle is illustrated in Figure 3.1 and also later in Figure 3.2.

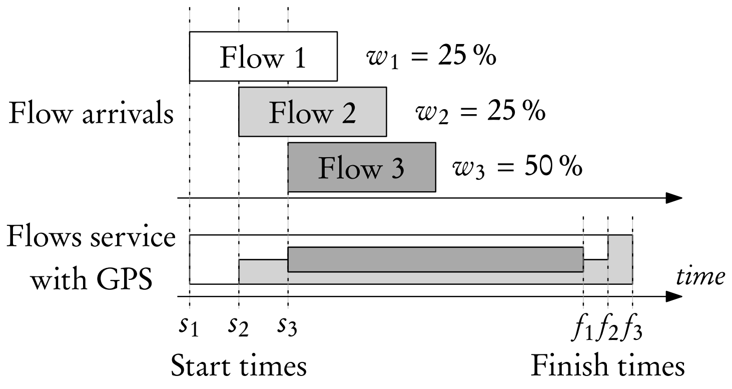


Fig. 3.1: Illustration of Generalized Processor Sharing (GPS)

Various algorithms have been proposed in the literature to implement this idea in an efficient way. The most commonly seen in the literature are: Packetized General Processor Sharing (PGPS) also called Weighted Fair Queuing (WFQ) presented by Demers *et al.* in [99], Worst-Case Fair Weighted Fair Queuing (WF^{2}Q) by Bennett and Zhang [58], Self-Clocked Fair Queuing (SCFQ) by Golestani [127], or Deficit Round Robin (DRR) by Shreedhar and Varghese [222]. The WFQ, WF^{2}Q and DRR algorithms will be presented in more details later in this section.

The main advantages of this family of algorithms are: *(i)* deterministic and stochastic guarantees can be given to all classes of service as shown by Parekh and Gallager in [197] (for deterministic guarantees) and by Zhang *et al.* in [254] (for stochastic ones); *(ii)* the starvation effect noted for priority scheduling cannot occur here as each class of service is guaranteed a minimum bandwidth of $^C/_N$ (or $C \cdot w_i / \sum_{k=1}^{N} w_k$ if weights are used).

The main disadvantages of this family of algorithms are: *(i)* its algorithmic complexity, which is usually between $O(\log N)$ and $O(N)$ work per packet (with N the number of served queues) depending on the algorithm; *(ii)* implementation difficulties due to possible integer overflows. This can be a limiting factor depending on the chosen hardware, algorithm and number of queues. Those disadvantages have triggered a lot of work in the area of scheduling, as demonstrated by the number of publications on the topic and the different variants of

FQ algorithms, where the goal is to have a $O(1)$ work per packet while keeping the fairness property noted in Equation (3.1).

Secondly, those scheduling policies are generally not adapted to applications with low-latencies, because there is no notion of priority, although a good choice of weights may prevent this as for instance shown by Bennett and Zhang in [59]. We will see two alternate solutions to this point in Chapters 3 and 4, where we either choose the weights of the queues in order to still have low-latencies, or use and hybrid priority/fair queuing algorithm.

This family of scheduling algorithm is found less frequently in COTS Ethernet switch due to the disadvantages aforementioned.

Weighted Fair Queuing

Weighted Fair Queuing (WFQ) is a packet-by-packet approximation of Generalized Processor Sharing (GPS). GPS is an idealized scheduling policy implementing the relation described in Equation (3.1) where flows are infinitely divisible, *i.e.* a fluid model is used. Because this fluid model is not directly applicable to packets, WFQ uses the finish time of each head-of-line packet as if it were served by GPS, and serves the queue with the smallest GPS finish time. This concept is presented in Algorithm 3.1, which corresponds to a simplified version of the dequeuing module of WFQ. An illustration of the WFQ service discipline is given later in Figure 3.2.

Algorithm 3.1 Pseudo-code of a (simplified) dequeuing module of WFQ

```
Enqueuing module: input packet $p$, queue $i$
1: $virtualStartTime \leftarrow \max(now(), tailPacket(i).virtualFinishTime)$
2: $p.virtualFinishTime \leftarrow virtualStartTime + \frac{size(p)}{bandwidth} \cdot \frac{w_i}{\sum_{k \in \Psi(t)} w_k}$
3: $Enqueue(i, p)$
Dequeuing module:
1: $bestQueue \leftarrow \varnothing$
2: $bestFinishTime \leftarrow +\infty$
3: for all Queue $i \in \Psi(t)$ do
4:     if $headPacket(i).virtualFinishTime() < bestFinishTime$ then
5:         $bestFinishTime \leftarrow headPacket(i).virtualFinishTime()$
6:         $bestQueue \leftarrow i$
7:     end if
8: end for
9: $Dequeue(bestQueue)$
```

Worst-case Fair Weighted Fair Queuing

Worst-case Fair Weighted Fair Queuing (WF^{2}Q) proposed by Bennett and Zhang in [58] is an improvement of WFQ. While it can be proven that WFQ does not fall behind GPS by more than one maximum-size packet, the criticism addressed in [58] is that it can be far ahead of GPS, which means that it can introduce jitter in the flows.

WF^{2}Q avoids this problem by serving the queue with the smallest GPS finish time (like in WFQ), but only if the corresponding GPS start time is reached. This property means that WF^{2}Q provides almost identical service than GPS, differing by no more than one maximum size packet.

A short summary of both WFQ and WF^{2}Q is presented on Figure 3.2. In this case, we have 11 queues, with a weight of 10 for the first queue, and a weight of 1 for the other queues. We can see here how WFQ only uses the finish time of GPS, while WF^{2}Q uses both finish time and start time to service the packets. Such behavior means that flows should experience less jitter with WF^{2}Q than WFQ.

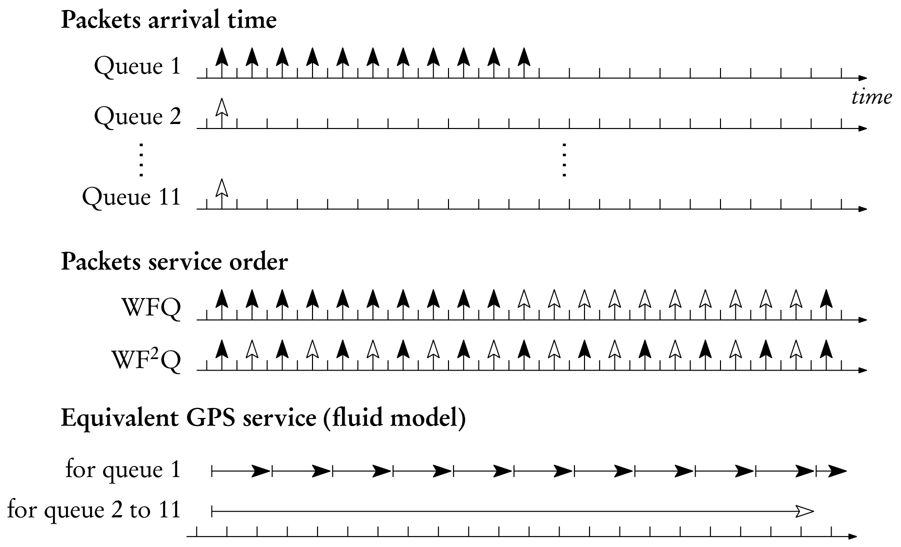


Fig. 3.2: Example of GPS, WFQ and WF^{2}Q service orders. Vertical arrows correspond to packet arrivals or departures. Horizontal arrows correspond to GPS service times, where the base of the arrow is the start service time and its tip is the finish time.

Deficit Round Robin

Deficit Round Robin (DRR) is a variant of Round Robin proposed by Shreedhar and Varghese in [222]. In Round Robin, each non-empty queue is served one after the other, always is the same order. DRR is an approximation of the Fair Queuing principle of fairness in terms of throughput which requires only $O(1)$ work to process a packet. DRR was designed to have fairly straightforward implementation compared to WFQ.

In DRR, each queue has a deficit counter, incremented at each round-robin cycle of the scheduler by a *quantum* of bits when the queue holds packets. For a queue to be served, its head-of-line packet size must be below the current value of the deficit counter. Otherwise, the queue will have to wait for a future round, after the deficit counter has been incremented enough. When a queue is served, its deficit counter is decremented by the size of the packet which has been served. The counter is reset to zero when the queue is empty. The workings of DRR will also be detailed later in Section 5.4.

3.3.3 Other schedulers

While the two families of scheduling algorithms described in Sections 3.3.1 and 3.3.2 are the most frequent ones in COTS switches, one variation often found is to combine both of them to bring their advantages together. This will be discussed in more details in Chapter 4.

Apart from those schedulers, COTS switches might also implement a simple Round-Robin or Weighted Round-Robin (WRR) scheme, where each queue is polled one after the other and if one queue is not empty, its head-of-line packet will be sent. While those algorithms offer a simple algorithm and could be seen as simplified versions of fair queuing, we note that the bandwidth sharing depends on packet size and hence is biased. In other word, a class of traffic with large packet will receive more bandwidth than one with small packets.

We focused in this section only on packet schedulers found in current COTS Ethernet switches. We refer to the work from Zhang in [250] for an early survey of various other schedulers used for guaranteed performances. There are many other algorithms and Quality-of-Service architectures proposed in the literature, which are generally designed to optimize some specific performances of the packets or flows they serve.

3.4 The IEEE 802.1Qav scheduling architecture

The IEEE 802.1Qav standard describes a scheduling architecture which has to be used for AVB multimedia flows. It is based on a traditional SPQ architecture with eight queues, where the two top priority ones are dedicated to audio and video traffic. For those two queues, a special shaper algorithm was developed, called *Credit Based Shaper* or CBS, which gives delay guarantees to the transmission of multimedia traffic while preventing the starvation problem of SPQ. This scheduling architecture is presented in Figure 3.3.

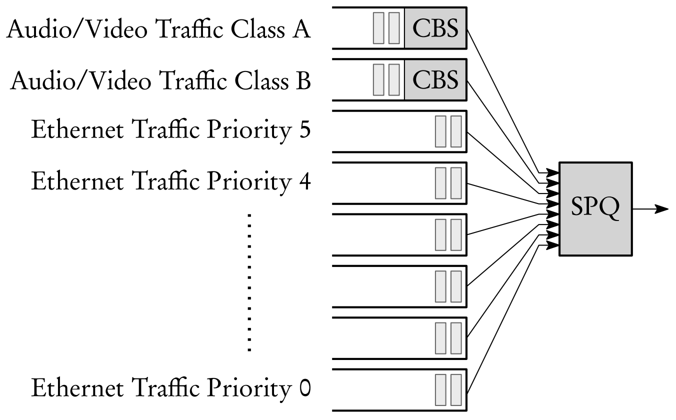

Audio/Video Traffic Class A
CBS
Audio/Video Traffic Class B
CBS
Ethernet Traffic Priority 5
Ethernet Traffic Priority 4
SPQ
Ethernet Traffic Priority 0

Fig. 3.3: IEEE 802.1Qav scheduling architecture

3.4.1 The Credit Based Shaper algorithm

The details of the Credit Based Shaper algorithm are presented in the IEEE 802.1Qav standard [2] or in Annex L of the IEEE 802.1Q standard [5], along with some mathematical properties and their associated proofs. This algorithm was designed with efficiency and simplicity in mind, with the following formulation:

- Each queue is assigned a credit value, noted here as *credit*, initialized to 0, and reset to 0 if the queue is empty and *credit* is above 0.
- When packets are present in the queue and not allowed to be served because of other frames currently being transmitted, *credit* is increased at a rate of *idleSlope*, with a maximum value of *hiCredit*.
- If *credit* is greater or equal to 0, transmission of packets are allowed.
- When packets are served, *credit* is decreased at a rate of *sendSlope*, with a minimum value of *loCredit*.

An example of the different states of the CBS algorithm is presented in Figure 3.4. The *idleSlope* and *sendSlope* values of the algorithm are functions of the fraction of the bandwidth we want to allocate (*bandwidthFr*) and C:

$$idleSlope = bandwidthFr \cdot C \tag{3.2}$$

$$sendSlope = idleSlope - C \tag{3.3}$$

In the normal operation of the AVB stack, *bandwidthFr* is automatically updated with the stream reservation protocol MSRP (Multiple Stream Reservation Protocol) which is part of the IEEE 802.1Qat [3] standard. When audio and videos streams are activated or deactivated on the network, the *bandwidthFr* parameter of the different switches affected by those changes is set automatically according to the new number of available streams (if there is enough bandwidth available). This process is analogous to the IntServ Quality-of-Service architecture and RSVP [70], the main difference being that AVB operates at layer 2 of the OSI model, while IntServ targets layer 3. For this evaluation, we set *bandwidthFr* manually (according to the parameter depicted in Table 3.1 as explained later).

It has been demonstrated in [2, Annex L] that the IEEE 802.1Qav scheduling architecture presented in Figure 3.3 admits an upper bound for the end-to-end delay for two AVB traffic classes. Deterministic network calculus curves have also been proposed for CBS as presented earlier in Section 3.2.

3.5 Evaluation in case of a reference AFDX network

We evaluate and compare here the schedulers presented in Section 3.3 and Section 3.4 in case of a reference AFDX network. We focus here on two end-systems of an industrial AFDX configuration, called here ES-0 and ES-1, and look at the performances of their respective avionic Virtual Links. Those two end-systems have been chosen to represent typical end-systems as the mean number of received Virtual Links by an end-system is around 100 in the studied configuration. The AFDX topology used here is similar to the one presented in Section 2.2.1. In order to evaluate the influence of increased traffic in the network, additional end-systems were placed at different points, emitting so-called *best-effort* VLs. Packet scheduling is performed inside the switches of the network to differentiate between the avionic and best-effort

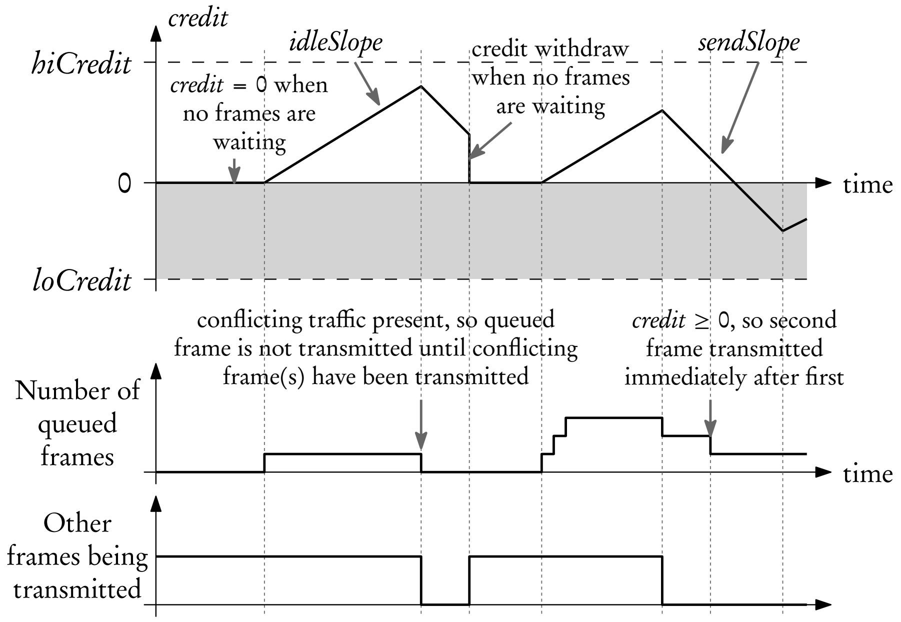

credit
hiCredit
idleSlope
credit withdraw when no frames are waiting
sendSlope
credit = 0 when no frames are waiting
0
time
loCredit
conflicting traffic present, so queued frame is not transmitted until conflicting frame(s) have been transmitted
credit ≥ 0, so second frame transmitted immediately after first
Number of queued frames
time
Other frames being transmitted

Fig. 3.4: Illustration of the Credit Based Shaper algorithm, based on [2, Annex L]

VLs. The methodology used here is to compare the performances of the VLs according to which scheduling algorithm was used.

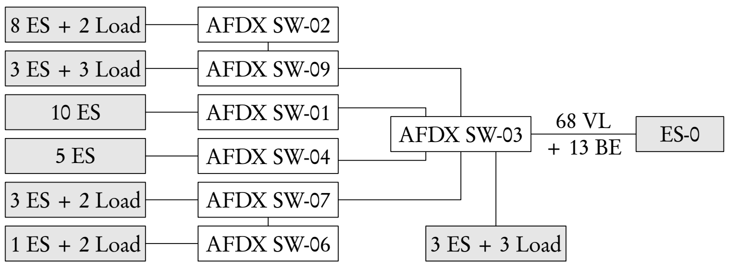
The two end-systems of interest as well as the end-systems communicating with them and their place in the topology are presented in Figure 3.5. The networks elements present on the topologies in the figure are: *(i)* AFDX switches, noted "AFDX SW-*x*"; *(ii)* groups of avionic end-systems noted "*n* ES" with *n* the number of end-systems; *(iii)* groups of load end-systems, noted "*m* Load", with *m* the number of end-systems, sending the best-effort load flows (noted "BE").

The different schedulers compared in this evaluation as well as their respective configuration are given in Table 3.1. As noted in Section 3.3.2, there have been various propositions for algorithms performing fair queuing, which is why we choose to compare the most prominent ones, namely WFQ, WF^{2}Q and DRR. The schedulers have to serve two queues, one for the *avionic* traffic and one for the *best-effort* one. We also have a special case where no scheduling is used with a single FIFO queue where the *avionic* and *best-effort* flows are mixed. The goal is to evaluate the performances of the flows in absence of scheduling.

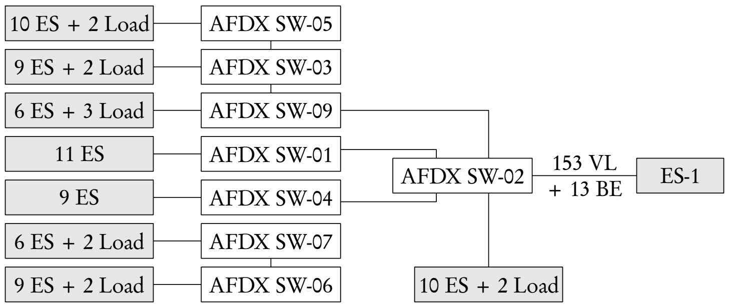
3.5.1 Performance evaluation

Note Figures 3.6 and 3.8 and their associated interpretations have been added compared the original publication [119]. As one point of this chapter was to see the impact of the best-effort load on the real-time performances, those figures were added to help understand which scheduling algorithms are the best suited to isolate real-time traffic from additional load.

The performance evaluation of the network was performed using the discrete-event simulator



(a) First AFDX topology with ES-0



(b) Second AFDX topology with ES-1

Fig. 3.5: The two separate AFDX topologies used for the evaluation

Algorithm	**Configuration**
SPQ	One high priority queue for avionics flows, and one low priority queue for the best-effort flows
WFQ, DRR, WF^{2}Q	$^2/_3$ weight for avionics flows, and $^1/_3$ for best-effort flows
CBS	Avionics flows served with the CBS algorithm with $^2/_3$ of the bandwidth, and best-effort in a separate FIFO queue
FIFO	One common queue for avionics and best-effort flows, without traffic differentiation

Tab. 3.1: Scheduler configurations chosen for this evaluation

OMNeT++ [17] with the framework INET [14]. The complete industrial AFDX reference network was simulated and we focus on the two end-systems described earlier. Virtual Links are simulated as periodic UDP packets generated following the *BAG* and s_{max} parameters of the reference configuration. We followed a Monte-Carlo method by simulating each configuration ten times.

The initial workload received by ES-0 represents 13 % of the link capacity, and 22 % for ES-1, which corresponds only to the avionics flows. In order to generate different levels of load directed toward the two end-systems and see the performance changes, the BAG of the *best-effort* VLs is changed. We generate load up to 96 % to see the performances of the flows in a near overload case.

In order to have a good comparison basis, our reasoning is that the SPQ scheduler will give the best performances in term of latency for the avionic VLs, as they have the highest priority. Our goal then is to compare the performances of the VLs served by other schedulers presented in Table 3.1 to the ones served by SPQ. For this purpose, we use the statistical test described by Law and Kelton in [167, Chapter 10.2] which is a method for comparing two sets of numbers and determine if there is a significant difference between the two sets.

This method described is a two-sample location test of the null hypothesis that the means of two populations are equal. Let $\mathcal{S}$ be the set of reference samples (*i.e.* the performances with SPQ in our case), and $\mathcal{R}$ the set of samples with want to compare it to (*i.e.* the performances of another scheduler). The test uses a corrected number of degrees of freedom ν compared to Student's t-test, which is computed using the Welch-Satterthwaite equation [212, 242]:

$$\nu = \left\lfloor \frac{\left(\frac{s_{\mathcal{R}}^2}{|\mathcal{R}|} + \frac{s_{\mathcal{S}}^2}{|\mathcal{S}|}\right)^2}{\frac{1}{|\mathcal{R}|-1}\left(\frac{s_{\mathcal{R}}^2}{|\mathcal{R}|}\right)^2 + \frac{1}{|\mathcal{S}|-1}\left(\frac{s_{\mathcal{S}}^2}{|\mathcal{S}|}\right)^2} \right\rfloor \tag{3.4}$$

with $|\mathcal{R}|$ and $s_{\mathcal{R}}^2$, respectively $|\mathcal{S}|$ and $s_{\mathcal{S}}^2$, the cardinal and unbiased estimator of the variance of $\mathcal{R}$, respectively of $\mathcal{S}$. The confidence interval of the difference between the two samples is then:

$$(\mu_{\mathcal{R}} - \mu_{\mathcal{S}}) \pm \left(\frac{s_{\mathcal{R}}^2}{|\mathcal{R}|} + \frac{s_{\mathcal{S}}^2}{|\mathcal{S}|}\right) \cdot \mathrm{qt}\left(1 - \frac{\alpha}{2}, \nu\right) \tag{3.5}$$

with $\mathrm{qt}(x, y)$ the quantile function for Student's t-distribution with y degrees of freedom evaluated at probability x, and α the level of significance. For this evaluation we choose a level of significance $\alpha = 0.1$.

Maximal end-to-end latency

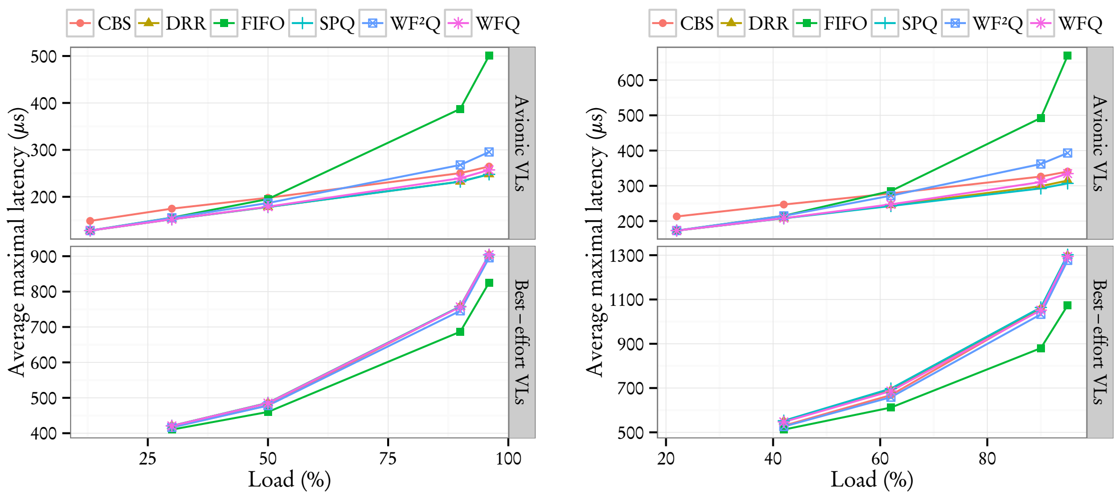
We take the maximal end-to-end latency per Virtual Latency as our first performance indicator, as each Virtual Link has requirements this measure. We define the latency of a frame, as the time between the first bit of the frame is sent on the wire link, and the time the last bit of the frame is captured by the received.

We first characterize the evolution of the latency according to the load in Figure 3.6. As expected, an increase in the load results in a growth in the end-to-end latency. We notice that is also has a smallest impact on the avionic VLs when using SPQ or DRR. The load increment has a smaller impact on the avionic VLs compared to the best-effort ones.

Not surprisingly, the network load has a high influence on the avionics frame latency when using FIFO. This scheduling scheme gives the worst latencies for avionics flows compared to other algorithms, but the best ones for non-avionics flows. It is due to the fact that no packet classification is performed.

Figure 3.7 represents the latency of the VLs received by ES-0 and ES-1, under the different scheduling algorithms, with a comparison with SPQ. We see that the choice of SPQ as our nominal case is good as no other algorithm performs better in the context of avionics flows.

For WFQ, we obtain similar results compared to SPQ as 0 lies in the confidence interval, which can be explained by the weight of the queues. The avionics flow is assigned a 2/3 share of the bandwidth, when in reality it is only using 13 % of the link. On the other hand, the



(a) Results for ES-0 **(b)** Results for ES-1

Fig. 3.6: Evolution of the average of the Virtual Links maximal end-to-end latency

best-effort flows were assigned only 1/3 of the bandwidth, which means that for the 50 % to 90 % load case, this class was always using more than its reserved bandwidth. In practice this results almost always in a higher priority for the avionics frames over the best-effort frames. The same analysis can be made for DRR, were we have almost similar results than SPQ and WFQ. Results for WF^{2}Q are worse than those of WFQ for higher load, since 0 is not within the confidence interval.

Regarding CBS, as its principal function is to shape traffic, we obtain higher latencies than SPQ for avionics flows. One of the benefits of such behavior is that some time slots are available for the lower priority class, resulting in lower latencies for the best-effort flows compared to WFQ and SPQ. Compared to the starvation problem of SPQ described earlier, this is an improvement.

Maximal end-to-end jitter

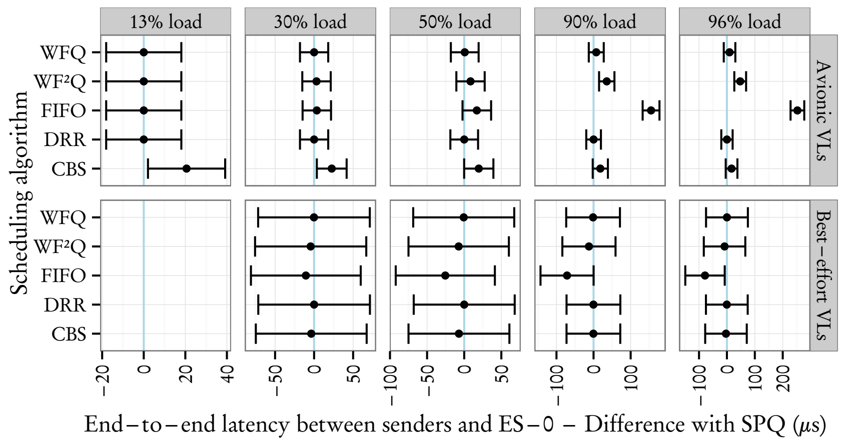
We used the *interarrival jitter* definition from RFC 3550 [220] for our second performance indicator. It is defined as the measure of packet arrival time spacing at the receiver smoothed with an exponential filter with parameter 1/16 (equivalent to a low-pass filter), such that:

$$\textit{time difference}(i) = (\text{Arrival time of packet } i) - (\text{Arrival time of packet } i-1) \tag{3.6}$$

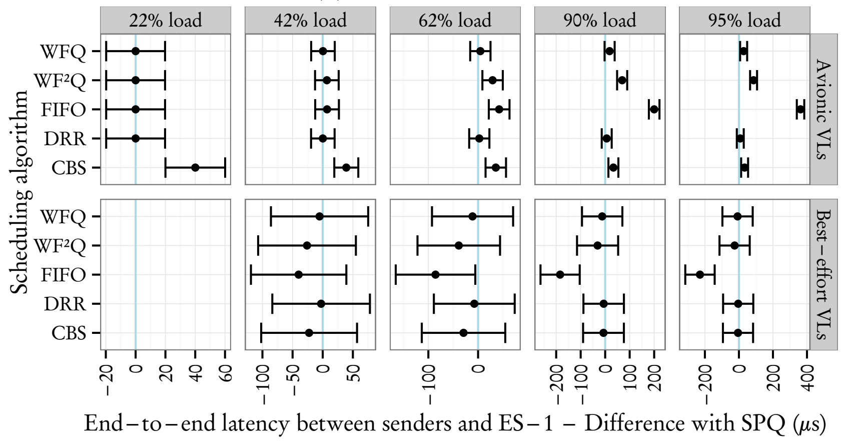
$$\textit{jitter}(i) = \textit{jitter}(i-1) + \frac{1}{16}\left(\textit{time difference}(i) - \textit{jitter}(i-1)\right) \tag{3.7}$$

As for the latency, we first characterize the evolution of the jitter according to the load in Figure 3.8. For low traffic loads, the CBS algorithm gives worse results for jitter than other algorithms. While CBS was mainly developed for fixed sized packets, where it would give better values for jitter, it was used here with heterogeneous packet sizes.

As for the latency analysis, FIFO gives the worst results for avionics VLs and the best results for best-effort VLs. This can again be explained by the fact that no packet classification is done.



(a) Results for ES-0



(b) Results for ES-1

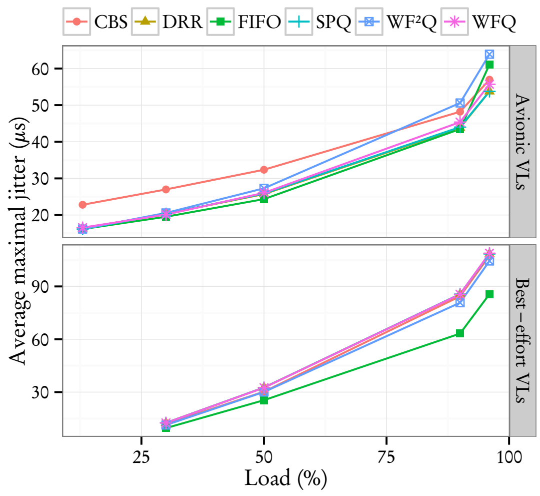
Fig. 3.7: Comparison of maximal end-to-end latency with SPQ according to the method presented in Equation (3.5). The top-left part of each sub-figure corresponds to the nominal case without additional best-effort traffic.

As expected, WF^{2}Q behaves better than WFQ for best-effort VLs in this jitter analysis, but with an impact on avionics VLs with high loads, where we can see a real difference compared to other algorithms. Finally, differences between SPQ, WFQ and DRR are almost not visible here.

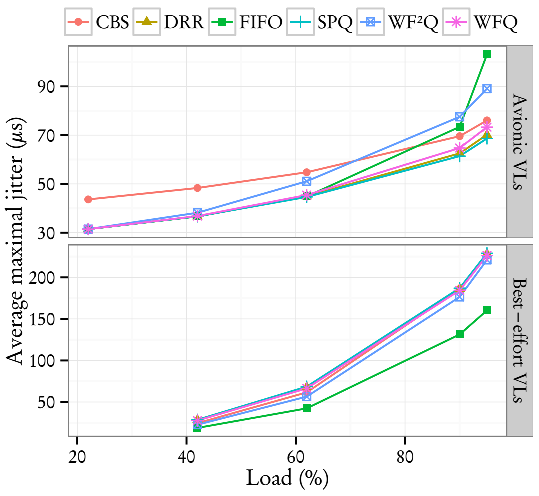
Figure 3.9 represent the maximal jitter experienced by avionics VLs and best-effort VLs by the both studied end-systems under different scheduling algorithms, compared to SPQ. The remarks made for Figure 3.8 also apply here.

3.6 Conclusion on the scheduling architecture of IEEE AVB

We noted previously in Section 2.4 that a network with mixed traffic criticality is also of interest for other industries, which lead to a recent push for new standardized packet scheduling architectures. Notably, a set of IEEE standards was published in 2011 and 2012 for the professional audio/video industry targeted at providing reliable and high quality audio and



(a) Results for ES-0



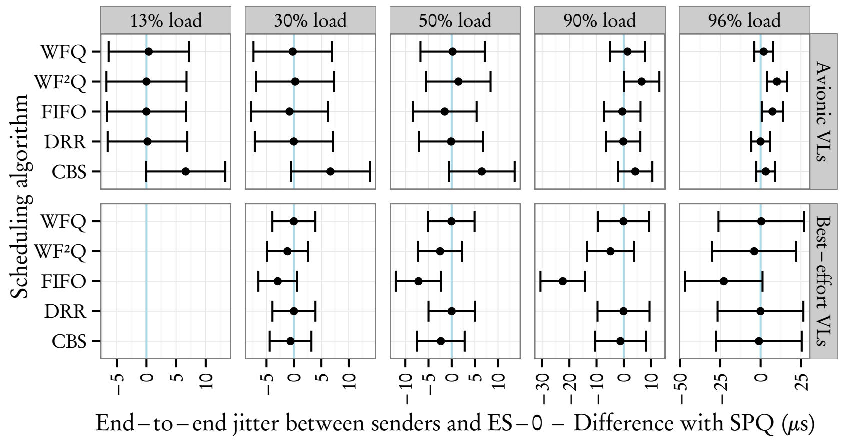
(b) Results for ES-1

Fig. 3.8: Evolution of the average of the Virtual Links maximal end-to-end jitter

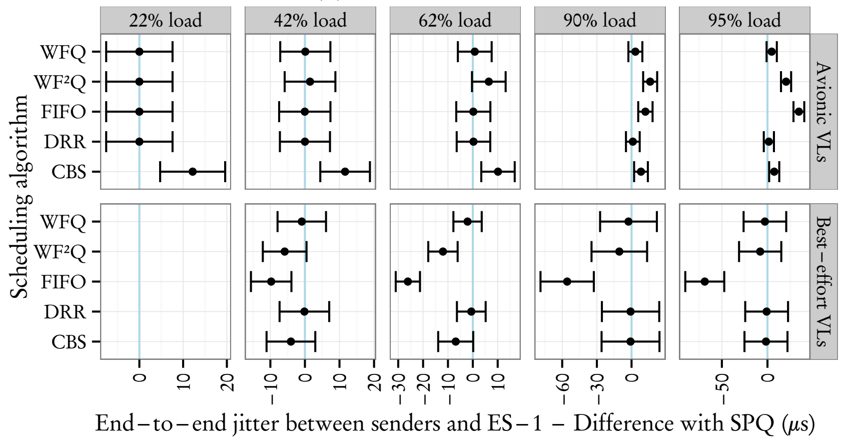
video streaming in Ethernet networks with mixed best-effort traffic. We choose to investigate this solution because of the following two points. First, this solution provides guarantees for critical flows while still allowing good performances for the best-effort flows and preventing the starvation problem of strict priority queuing. This point fits the aim of Research Objective **O1.1**. Secondly, from a more practical point of view, due to its standardization and possible mass-market use in the future both for professional audio/video and automotive markets, using such technology could lead to cost reductions compared to the costly custom-designed solutions currently used in aircrafts.

Those two points lead us to evaluate this solution in this chapter by applying it in case of a reference industrial AFDX network, to see if this solution is viable for an evolution of AFDX toward a mixed critical network. In order to evaluate the benefits of this new scheduling architecture as prescribed earlier in Research Objective **O1.2**, we compared here the scheduling architecture proposed by the IEEE in the IEEE 802.1Qav standard [2] with other well-known algorithms presented in Section 3.3. Discrete-event simulation of an industrial AFDX network was used in order to get meaningful results in the context presented in Section 2.2.

Based on our results, we conclude that the scheduling architecture proposed in IEEE 802.1Qav is a possible candidate for an evolution of AFDX with mixed-criticality. The end-to-end delays and jitter measurements showed that it offers results similar to well-known scheduling algorithms, with the advantage of a simple implementation. Furthermore, using shaping on top of SPQ is a good mechanism against queue starvation, where high priority queue might prevent low-priority from transmitting. Although such function is not necessary for AFDX Virtual Links as there are mechanisms to enforce that they do not overload the network, this function is of benefit for scheduling between different classes of non-AFDX flows. Finally, as a worst-case delay analysis for all traffic classes using CBS has been showed via various methods, it provides a good position for certification processes used in the aeronautic industry. More precisely, due to the current use of network calculus for certifying AFDX network and the recent developments of De Azua and Boyer in [97], it places CBS as a good candidate for an



(a) Results for ES-0



(b) Results for ES-1

Fig. 3.9: Comparison of maximal end-to-end jitter with SPQ according to the method presented in Equation (3.5). The top-left part of each sub-figure corresponds to the nominal case without additional best-effort traffic.

evolution of AFDX with mixed criticality.

Key insights and contributions

Comparison of scheduling algorithms for mixed-criticality We evaluated and compared the impact of scheduling algorithms via discrete-event simulations as prescribed in Research Objective **O1.1**. We showed that when no scheduling is used, the load of additional flows has a large influence on the performances of real-time flows, while using a strict priority or fair-queuing scheme provides some level of isolation.

Evaluation of the scheduling architecture from IEEE We evaluated a new scheduling architecture for mixing real-time with best-effort traffic called CBS. This fits the aim of Research Objective **O1.2**. We showed that due to its non work-conserving principle, there is a small performance drop at small loads for real-time traffic, but the best-effort traffic can benefit from it. In case requirements are still fulfilled using CBS, it might still prove to be a good solution for a future mixed-criticality network.

4. HYBRID SCHEDULING WITH A MIX OF PRIORITY AND FAIR QUEUING

Note This chapter is based on our previous publication [122], published in *Proceedings of the 4th International Workshop on Aircraft System Technologies*, 2013. Section 4.2 has been extended compared to the original publication. Figure 4.6b as well as the evaluation of the simulation results using Equation (4.3) and its associated applications (*i.e.* Figures 4.4b and 4.5b) are extensions of the original publication.

4.1 Introduction

We noted in Chapter 2 that we target networks with a mixed set of requirements. Most notably, those requirements can be roughly split into two groups: *(i)* real-time applications with low end-to-end latency needs and small bandwidths; *(ii)* elastic applications where latency is less import, but bandwidth plays a larger role.

We propose in this chapter to use a Quality-of-Service architecture similar to DiffServ, where instead of treating each flow individually, flows with similar requirements are grouped into classes, and packet schedulers interact on classes instead of flows. This concept introduced by Floyd and Jacobson in [116] is also sometimes referred as Class-Based Queuing (CBQ) and has mainly been introduced to provide some scalability to QoS requirements.

Looking at the schedulers presented in Section 3.3, one solution to address the flow requirements would be take the advantages of both priority and fair queuing policies by using a hybrid scheduler as noted in Section 3.2. This scheduling architecture is presented in Figure 4.1, where we have two groups of queues:

Queues 1 to 4 dedicated to the low-latency traffic: those queues are scheduled using SPQ, with queue 1 having the highest priority and queue 4 the lowest one;

Queues 5 to 8 dedicated to the elastic traffic: those queues can only be served if queues 1 to 4 are empty; the choice of which queue to serve among those will be made by a fair queuing algorithm.

We note that the scheduling architecture presented in Figure 4.1 is not novel, as it has for instance already been presented in RFC 5865 [44]. This policy is also available in some COTS Ethernet switches such as the ones from Cisco or HP, under the name *Low-Latency Queuing* or LLQ, although they generally include only one queue served with priority queuing.

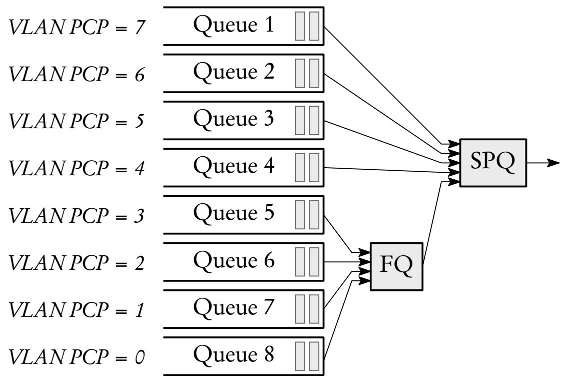


Fig. 4.1: Scheduling architecture with mixed priority and fair queuing policies. "VLAN PCP" corresponds to the value of IEEE 802.1Q Priority Code Pointer field.

We will study here this scheduling architecture in the context of the cabin network presented earlier in Section 2.2.2 using discrete-event simulations. The contributions of this evaluation are threefold: *(i)* we characterize the end-to-end latencies, jitters and buffers usage which are to be expected in this particular star/daisy chain topology; *(ii)* we evaluate the separation between the two types of traffic in term of performances when using hybrid scheduling (Research Objective **O1.1**); *(iii)* we evaluate the behavior of the network in case of overload (Research Objective **O1.1**).

Structure of this chapter

In Section 4.2, we present related work. We will first present in Section 4.3 the network nodes' internal architecture as well as the different classes of traffic used for this evaluation. We will then present in Section 4.4 the performance evaluation of the scheduling with results regarding end-to-end latency of frames, end-to-end jitter and buffer utilization. Finally, Section 4.5 summarizes and concludes this chapter.

4.2 Related work

Note As the original publication serving as a basis for this chapter did not review the related work, this section is an extension of the original publication [122].

Various proposals have been made to provide Quality-of-Service on networks. The most notable ones are the IntServ and DiffServ architectures proposed by the IETF. A survey of those different architectures and mechanisms has been proposed by El-Gendy *et al.* in [103]. They noted that the main advantage of DiffServ were its scalable and its increasing support by network equipment vendors. They noted that traffic aggregation, admission control, and application-level QoS mapping were still unresolved challenges.

One solution proposed to address those challenges is the so-called Class-Based Queuing proposed by Floyd and Jacobson in [116]. Different variants of this principle have been investigated using various analytical frameworks in the literature. The Hierarchical Packet

Fair Queuing (H-PFQ) presented by Bennett and Zhang in [59] investigated and compared different packetized version of General Processor Sharing and their impact of latency. One drawback often noted for this approach is that the delay bound is dependent on the depth of the hierarchical scheduler. Similarly, Stoica *et al.* proposed the Hierarchical Fair Service Curve (H-FSC) link sharing in [229] which is similar the H-PFQ, but with a more flexible resource management and higher resource utilization. They proposed a formal method based on deterministic network calculus for defining scheduling parameters which decouples delay and bandwidth allocation. Similarly, Millet and Mammeri studied the delay bound of a real-time class with a Weighted Fair Queuing algorithm using deterministic network calculus in [190]. While the works previously cited on CBQ were limited to a single node, Kim and Kim proposed in [160] to study DiffServ networks with multiple nodes using deterministic network calculus.

Regarding the impact of traffic aggregation, Guérin and Pla investigated the use of traffic shaping using simulations in [130]. They showed that reshaping flows was an efficient solution to eliminate egress non-conformance in DiffServ networks. Jiang and Yao also studied this problem in [153] using analytical and simulation results. One focus of their study was the impact of burstiness on the end-to-end performances of a real-time class of traffic.

As noted previously in Sections 3.2 and 3.6, one approach often proposed for mixing real-time and best-effort traffic in a DiffServ environment is to use a hybrid scheduler with priority scheduling for real-time traffic, and fair-queuing for the rest of the classes or flows. This approach is has been recommended in RFC 5865 [44] for the DiffServ QoS architecture. In case of avionic networks, this solution has been put forward by Bauer *et al.* in [52] for a multi-criticality AFDX network. Dekeris *et al.* made a performance evaluation similar to our use-case in [98], where they showed the benefits in term of latency for high-priority audio traffic and the trade-off it has on best-effort traffic.

4.3 Presentation of the use-case

The topology evaluated here is a single daisy chain as presented in Figure 2.4, with 1 Gbit/s links between the cabin Ethernet switches. The exact architecture of the cabin Ethernet switches forming the daisy chain is presented in Figure 4.2. We have three switches for the three domains of interest (ACD, AISD and PIESD), connected to a main switch. The scheduling presented in Figure 4.1 will occur inside the main switch.

The network will support eight classes of service, as presented in Table 4.1. We consider a centralized architecture, where communications will always occur between the central server located at the front of the aircraft, and one (or more) device(s) connected to the different lines. Direct communication between two devices is not allowed. Following this set of rules, we see that the packet direction which is the most interesting for this study is from the devices to the server, as packets will have to traverse multiple hops and scheduling will play an important role in the performances.

We study three different topologies, noted here A, B and C, corresponding to three possible aircraft cabin layouts. The number of hops in the daisy chain will vary according to the topology, respectively 8, 12 and 15. This number of hops in the daisy chain directly reflects the physical length of the cabin and its associated number of seats. Table 4.2 summarizes the

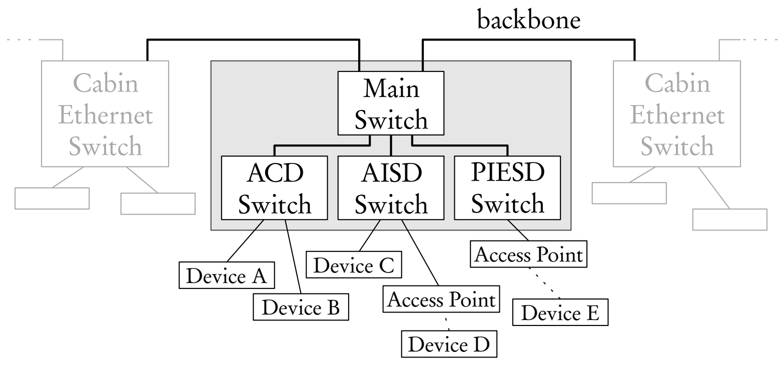


Fig. 4.2: Daisy chain cabin network topology

Class of Service	VLAN PCP	Scheduling	SPQ Priority	WFQ Weight
Interactive Audio	7	Priority Queuing	7	∅
Interactive Video	6	Priority Queuing	6	∅
Signaling, Sensors	5	Priority Queuing	5	∅
Special ACD flows	4	Priority Queuing	4	∅
Video Streaming	3	Fair Queuing	0	7
Network Management	2	Fair Queuing	0	1
Data Loading	1	Fair Queuing	0	4
Best effort	0	Fair Queuing	0	8

Tab. 4.1: Definition of the classes of service

number of devices connected in the different topologies as well as the flows they are sending to the server. The devices are placed at regular intervals on the line, except for the Crew and Passenger WLANs placed at the end of the line in order to simulate an unfavorable scenario where it will interfere with at every hop. Table 4.3 presents the aggregated bandwidth for per priority.

Flows are simulated here as UDP flows sending at constant bit rate. The packet size can be fixed or generated randomly between two values each time a new packet is sent using a uniform pseudo-random number generator.

As noted in the goals of this study, one point of interest is to evaluate the behavior of the network in case of overload. Hence, we devise for our performance evaluation the two following scenarios:

A normal scenario where the devices and flows follow the descriptions given in Table 4.2;

An overload scenario which simulates an excessive use of the WLAN access points, namely the bandwidth usage of the Crew WLAN is increased from 250 Mbit/s to 300 Mbit/s and the Passenger WLAN from 250 Mbit/s to 700 Mbit/s. In this case the network is overloaded.

Using those two scenarios, we are able to see the impact that uncontrolled applications (*i.e.* the passenger and the crew WLANs here) have on the performances of real-time flows (*i.e.*

Device type	**Bandwidth**	**Packet size**	**Priority**	**Domain**	**Devices / Topo**		
	(Mbit/s)	(B)			**A**	**B**	**C**
Phone	2	68	7	ACD	2	12	2
Passenger unit	0.24	68 – 1522	6	ACD	64	96	120
Sensor	1.6	68 – 1522	5	ACD	2	2	3
Light management	0.24	68 – 1522	4	ACD	0	96	0
Camera	16	68 – 1522	3	ACD	4	4	5
Crew HMI	1.2	68 – 1522	2	AISD	4	6	7
Crew WLAN	250	68 – 1522	1	AISD	1	1	1
Passenger WLAN	250	68 – 1522	0	PODD	1	1	1

Tab. 4.2: Flows in normal scenario

Priority	**Aggregated bandwidth** (Mbit/s)		
	Topology A	**Topology B**	**Topology C**
7	4	24	4
6	15.36	23.04	28.80
5	3.2	3.2	4.8
4	0	23.04	0
3	64	64	80
2	4.8	7.2	8.4
1	250	250	250
0	250	250	250
Total	591.36 Mbit/s	644.48 Mbit/s	626 Mbit/s

Tab. 4.3: Aggregate bandwidth usage per priority for each topology

packets of the ACD domain) .

4.4 Performance evaluation using simulation

Note In order to give a more precise quantitative comparison between the two scenarios (normal *vs.* overload), Equation (4.3) – which describes the relative difference between the performances – and its application in Figures 4.4b, 4.5b and 4.6b are an extension of the original publication [122].

The evaluation of this use-case was done using OMNeT++ [17] and its framework INET [14]. We follow a Monte Carlo method of simulating multiple runs, each time with a different seed and different initialization vector. For this evaluation, we perform 10 runs of each use case.

An overview of the global topology in OMNeT++ is presented in Figure 4.3

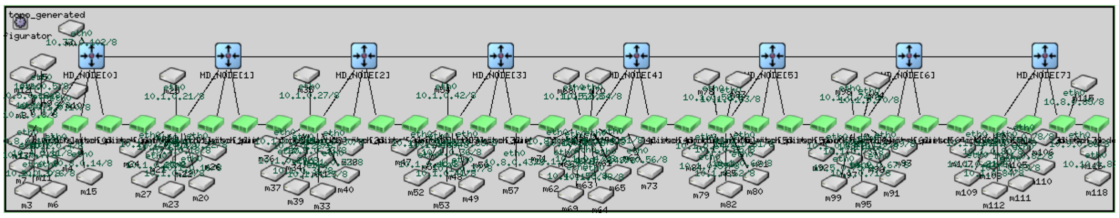
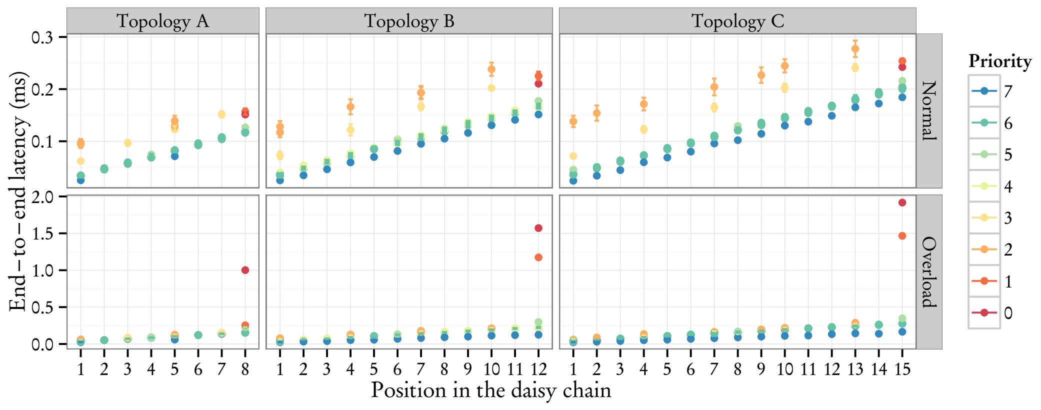


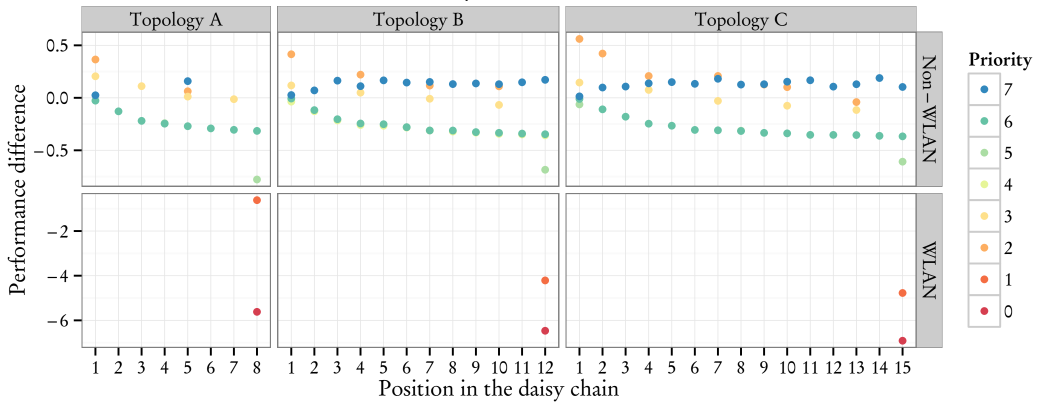
Fig. 4.3: Illustration of the OMNeT++ model with topology A. Blue blocks correspond to main Ethernet switches, green ones to domain Ethernet switches, and gray ones to devices

4.4.1 End-to-end latency measures

We define the end-to-end latency of a packet as the difference between the timestamp when it has been generated in the device, and the timestamp on which the last bit of the packet is received by the server. In this version of the model, Ethernet switches, as well as physical interfaces are considered to have no processing delay. This means that the only contributions to the frame's latency are: propagation, forwarding and queuing delays. Figure 4.4a presents the maximum end-to-end latency for each device measured in the simulation, in two use cases.



(a) Maximum end-to-end latency measure in normal and overload scenarios



(b) Performance difference for the maximal end-to-end latency

Fig. 4.4: End-to-end latency measures

Normal scenario

From the results of the normal scenario, we observe the effect of the daisy chain topology where each hop adds the store-and-forward delay. By looking at the differences between the end-to-end delays at each hop, we see that from priority 4 to 7, the higher the priority, the lower the end-to-end delay gets. This is a direct effect of the SPQ mechanism, where frames of higher priorities are sent before frames of lower priority. For applications such as interactive audio or video, such behavior is desirable.

Within priorities 0 to 3 served here with WFQ, the lower the weight, the higher the end-to-end delay gets. It has been demonstrated that the worst-case delay of a token bucket constrained flow with parameter (ρ, σ) (*i.e.* constrained by Equation (2.2)) traversing a single node following the WFQ service discipline is given by the following expression (see work from Sayenko *et al.* [213, Equations (3) and (4)]):

$$D = \frac{\sigma + L_{max}}{\rho} + \frac{L_{max}}{B} \tag{4.1}$$

$$= \frac{L_{max}}{B} \cdot \left(1 + \frac{N+1}{w_i}\right), \text{choosing } \rho = w_i \cdot B \text{ and } \sigma = N \cdot L_{max} \tag{4.2}$$

with L_{max} the maximum packet size, B the available bandwidth, w_i the normalized queue weight and N the number of flows sending data simultaneously. In other words, the worst-case queuing delay is inversely proportional to the weight of the queue. This property explains the fact that priority 2 (with the smallest weight) experiences larger end-to-end latency than the other queues served with WFQ.

Overload scenario

We consider now the second scenario where both WLANs are overused and overload the network. Results are also presented in Figure 4.4a. The two priorities corresponding to the WLANs (0 and 1) are impacted by this overuse of the network, with an end-to-end latency reaching up to 2 ms on Topology C. The rest of the flows on the other hand are less impacted. When comparing the mean of the maximal end-to-end latencies for priorities 2 to 7, we see an increase of around 20 μs in average over the three topologies, corresponding to an increase of about 20 %.

In order to get a better quantitative assessment regarding the performance differences between the two scenarios, we propose to use here a modified relative error between the maximal end-to-end latencies of the different nodes of the networks such that:

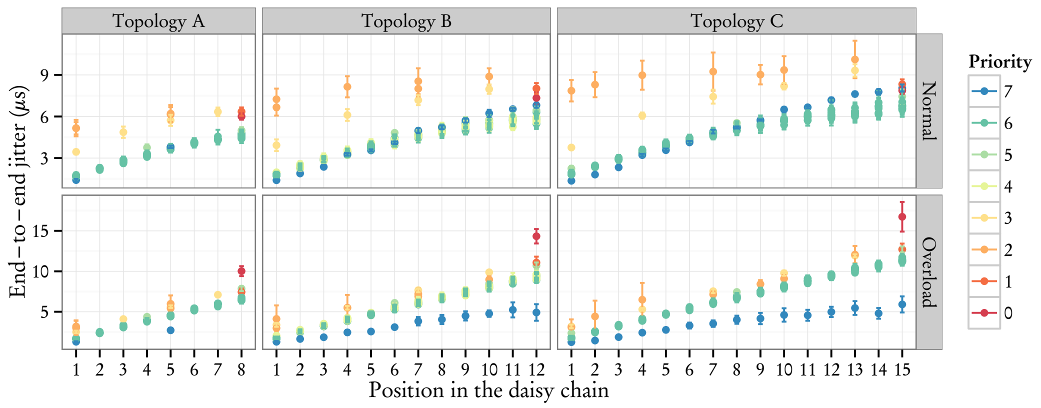
$$\textit{performance difference}(\textit{flow } i) = \frac{\textit{latency}_i^{\textit{overload scenario}} - \textit{latency}_i^{\textit{normal scenario}}}{\textit{latency}_i^{\textit{normal scenario}}} \tag{4.3}$$

Results are presented in Figure 4.4b. We notice that some priority have better performances in case of overload, which is due to the fact that simulations are used, *i.e.* the case of higher latencies during the normal scenario did not happen during the overload scenario. Nevertheless, we note that most of the performance differences of the non-WLAN traffic are in the ±50 % band, indicating that the overload has a rather small impact on the performance of the traffic.

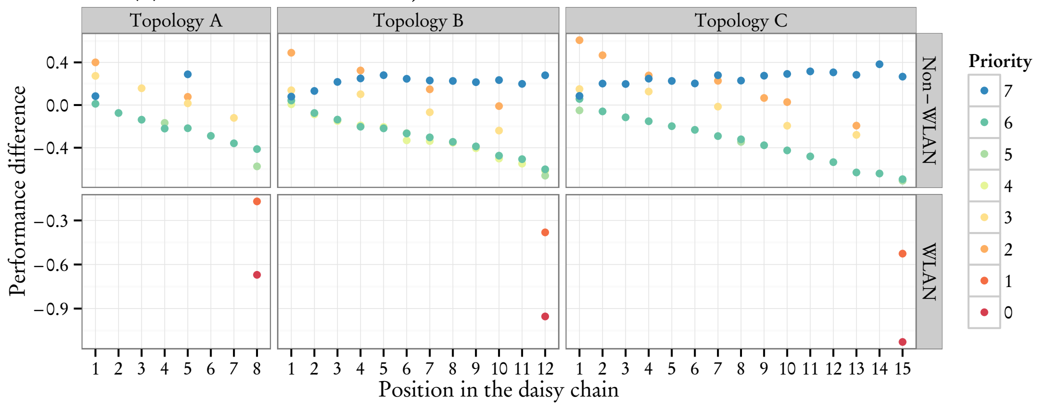
Another interesting point to note on Figure 4.4b is the correlation between performance difference and position on the daisy chain. While there is a clear correlation for the first few nodes of the daisy chain, this correlation is more subtle for the tail of the daisy chain.

4.4.2 End-to-end jitter measures

As in Section 3.5.1, we used the *interarrival jitter* definition from RFC 3550 [220] for the end-to-end jitter measurement. It is defined as the measure of packet arrival time spacing at the receiver smoothed with an exponential filter with parameter $^1/_{16}$ as shown in Equation (3.7). Figure 4.5a presents maximum end-to-end jitter experienced by the different devices.



(a) Maximum end-to-end jitter measure in normal and overload scenarios



(b) Performance difference for the maximal end-to-end latency

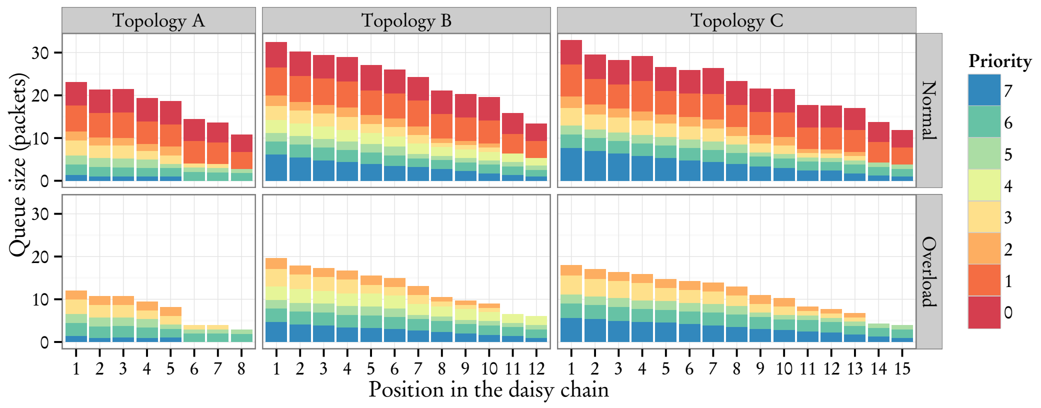
Fig. 4.5: End-to-end jitter measure

Like for the end-to-end delay, devices connected at the end of the daisy chain experience higher jitter than devices connected in front. We see that the overload has a minor effect on the rest of the traffic.

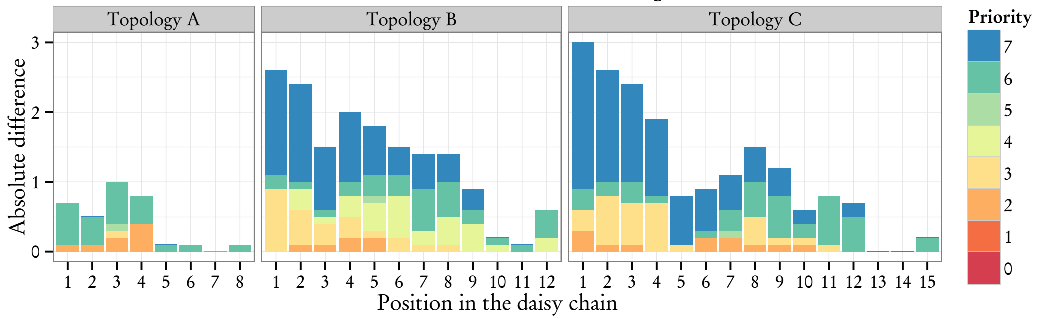
As for the latency, we also measure the performance difference on jitter using an adapted version of Equation (4.3). Results are presented in Figure 4.5b. As for the latency, the performance differences for the non-WLAN traffic are in the ±50 % band. We note here that the position on the daisy chain does have a larger influence on the jitter performances than with the latency.

4.4.3 Buffer usage measures

In order to have an indicator of the memory usage needed for the buffers of the main switch, we also investigated the maximum buffer usage experienced by the queues of the main switch in the simulation. Figure 4.6 shows the maximum number of frames per queue in number of packets. We notice almost no differences between the normal case and the overload case for the flows not overflowing the network (priorities 7 to 2). Flows with priorities 0 and 1 are not represented in the overload case as those queues will overflow.



(a) Maximum buffer usage



(b) Absolute difference of maximum buffer usage between the two scenarios

Fig. 4.6: Buffer usage

We conclude from Figure 4.6 that the buffer size will have to at least contain 32 packets – or 48 kB when using a maximum packet size of 1522 B – in order to prevent buffer overflow and packet loss.

One interesting point to notice is that the queue size is not directly linked to global bandwidth usage (see Table 4.3), especially for priority 7. This is also presented on Figure 4.6b, where we evaluated the absolute difference on the maximum buffer usage between the two scenarios. We notice a maximum change of only 3 packets.

4.5 Conclusion on hybrid scheduling

We studied in this chapter the cabin topology presented in Section 2.2.2. Our goal here was to investigate a solution which isolates real-time from best-effort traffic and evaluate the influence best-effort traffic can have on the performances of the other flows (Research Objective **O1.1**). We proposed in this chapter to use a hybrid scheduling architecture mixing strict priority (for the real-time traffic) with fair queuing (for the best-effort traffic).

In order to see the influence of this scheduler architecture on the flow performances, we devised two scenarios. The first one consisted of flows using their expected bandwidth. For the second scenario, we considered that wireless access points located in cabin could overflow the network and hence possibly influence the performances of real-time traffic.

The performance evaluation of the network was done using discrete-event simulation on three different variants of the cabin topology presented in Section 2.2.2. We evaluated the maximal end-to-end latency and jitter of the different flows of the network. The performance of the non best-effort flows is affected by the increase of the load in the second scenario, but only by a difference of ±50 %. We also noticed that the position on the line has an influence on the performance difference, especially for jitter. This study also enabled us to learn some insights on latencies experienced by real-time traffic on this daisy chain topology using the store-and-forward principle which will be useful for a comparison in Chapter 5.

Key insights and contributions

Evaluation of hybrid priority/fair queuing We have presented in this chapter a performance evaluation of the cabin network with the hybrid scheduling architecture presented in Figure 4.1, which mixes priority and fair queuing.

Influence of overload on the performance of real-time traffic We evaluated the influence of best-effort traffic by comparing a normal and overload use-case. We noticed that the chosen hybrid scheduler does indeed isolate the high priority traffic from large performance deviations (Research Objective **O1.1**).

5. REDUCING REAL-TIME LATENCIES WITH TIME-AWARE SCHEDULING

Note This chapter is based on our previous publication [125], published in *Proceedings of the 6th International Workshop Nets4Cars/Nets4Trains/Nets4Aircraft*, 2014. Section 5.5, the evaluation of the simulation results using Equation (4.3) and the associated applications (*i.e.* Figures 5.3b, 5.4b and 5.5b) in Section 5.6 are an extension of the original publication.

5.1 Introduction

We evaluated and discussed in Chapters 3 to 4 about scheduling architectures working on prominent principles, namely work-conserving schedulers and store-and-forward Ethernet switches, and saw the performances we can expect on the avionic networks presented in Section 2.2. We propose in this section to have a look at another solution to accomplish better performances for real-time traffic with low-latency requirements in case of the cabin network.

The first solution we investigate in this chapter is *cut-through switching*, a packet switching method where the transmission of an Ethernet frame is started before the complete frame has been received. The rationale behind this idea is that due to the daisy chain topology used for the cabin network (see Figure 2.3) and the communication pattern where the exchanges only occur with the server at the front of the line, devices placed at the opposite side of the server will experience a higher latency than devices connected directly one hop after the server. This was illustrated before in Figure 4.4a, where the latencies differences between the first and the last hop on the daisy chain was about one order of magnitude. This means that using cut-through switching, we have to account for the forwarding delay only once for the whole line, and not at each hop as with store-and-forward.

The second solution we consider here is a time-based access to the backbone, where schedulers at different points of the daisy chain are synchronized in such a way that high-priority low-latency packets do not experience queuing delay. The rationale behind this idea, is that queuing delay has a large influence on end-to-end latencies and avoiding it would lead to low latencies. To do so, the scheduler should function in a non-work-conserving manner, where it will prevent that a frame is sent before the real-time frame and result in queuing delay for the real-time packet arriving a short moment later. In parallel to the expected better performances for the scheduled traffic, we also have to take into account the mixed-criticality property introduced in Section 2.4, where we still need some level of Quality-of-Service for non-scheduled traffic.

Our goal in this chapter is to evaluate if those two technical solutions really bring better performances for real-time traffic and how it compares to a more traditional Ethernet network. As there is currently no standard way to provide the wished technical solutions, we must first solve the following challenges:

How can we schedule the access of the daisy chain in order to prevent overloading? This question will be answered in Section 5.3, where we propose to use a Time Division Multiple Access (TDMA) schedule with reserved time windows for each traffic type.

How can we synchronize the schedulers in order to prevent queuing delay? This question will be answered in Section 5.3, where we propose to use a master-slave protocol in order to avoid the challenge of clock-synchronization.

Which packet scheduler should be used to mix real-time and best-effort traffic? This question will be answered in Section 5.4, where we propose the Time-Aware Deficit Round Robin (TADRR) packet scheduling algorithm, a novel scheduler able to mix scheduled real-time traffic with fair-queuing, which is the main contribution of this chapter.

Structure of this chapter

We first look at the related work in Section 5.2. We then introduce in Sections 5.3 and 5.4 our general scheduling architecture and packet scheduling algorithm proposition. We then present in Section 5.6 the performance evaluation of our scheduling architecture, via discrete-event simulations, and we compare the performance gains (or loss) with a more traditional architecture. Finally, Section 5.7 summarizes and concludes this chapter.

5.2 Related work

We note that the idea of using a dedicated network-wide time window for specific real-time traffic is not new and has been proposed in various other networking technologies besides Ethernet. For instance, the ISO 17458 part 1 to 5 standards [7] – also referred as FlexRay – uses TDMA-based traffic separation similar to the one used here, where time-slots are reserved for static real-time traffic, while other time-slots are reserved for dynamic traffic, as illustrated previously in Figure 5.1.

In the case of Ethernet, this approach has also been proposed by Kopetz and Grünsteidl in [162] and later implemented in the proprietary solution known as Time-Triggered Ethernet (also called TTEthernet). The basic idea of the approach proposed by Kopetz and Grünsteidl is that packets are scheduled offline and the correct play of the schedule depends on a share clock information among the different participants of the network. This technique offers exact worst-case performances as the packets global schedule is computed before-hand, but comes at the cost of difficulties for computing this schedule and also the dependency to a precise clock synchronization protocol. This drawback has an impact on the scalability of such solution. For an example of the use of TTEthernet, Kamieth *et al.* proposed in [156] to use such network for an automotive use-case.

Regarding open solutions for Ethernet, the approach of allocating time windows for specific traffic was recently proposed in the IEEE Time-Sensitive Networking task group [11] with the IEEE 802.1Qbv draft standard [9] under the name *time-aware shaper*. This mechanism was studied recently by Alderisi *et al.* in [29] where using simulations they showed the benefits in term of performances of this time-aware shaper compared with traditional AVB (as presented in Chapter 3) and TTEthernet.

On the topic of finding a schedule for periodic tasks in distributed real-time systems, as it is the case in the use-case presented here, various solutions were proposed such as the work from Ramamritham in [204], or Hou and Shin in [141].

5.3 Real-time traffic and access to the daisy chain

One specific requirement for the network studied in this chapter is that it has to support a real-time service with ultra low-latencies. In this protocol, a packet is sent by the server every 31.25 μs, containing real-time data, and switches on the daisy chain answer to this packet in a Round Robin manner. The requirement on this packets exchange is that the two-way end-to-end delay must happen before the next periodic packet, *i.e.* be smaller than 31.25 μs. We already notice that using the same architecture as in Chapter 4 is not sufficient. The goal of this part is to ensure that this periodic real-time packet experiences the least possible delay in the network.

The solution adopted here is to use a Time-Division Multiple Access (TDMA) architecture, where each switch on the daisy chain is allowed to send packets on the backbone only during certain time-slots as presented in Figure 5.1. We use a simple master-slave protocol to orchestrate which node is allowed to send. In order to avoid the use of clock synchronization protocols to synchronize the time-slots – which might be prone to possible complex failure cases – we take advantage of the 31.25 μs periodic packet which serves as synchronization token.

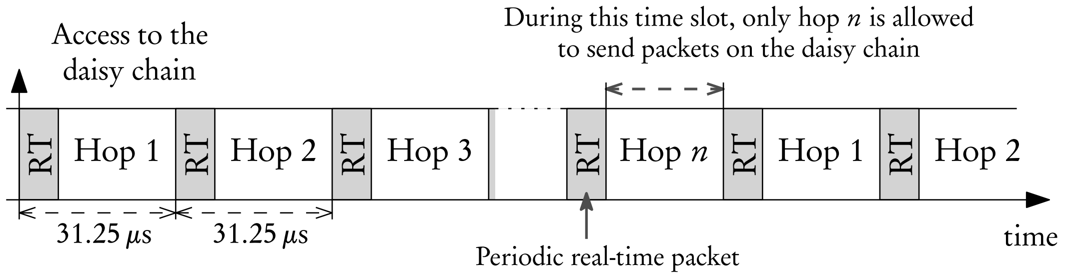

Access to the daisy chain
During this time slot, only hop n is allowed to send packets on the daisy chain
RT
Hop 1
RT
Hop 2
RT
Hop 3
RT
Hop n
RT
Hop 1
RT
Hop 2
31.25 μs
31.25 μs
Periodic real-time packet
time

Fig. 5.1: Access to the daisy chain following and TDMA and round-robin schedule

Following the TDMA schedule presented in Figure 5.1, the bandwidth allocated to each host will then be:

$$\frac{C_{backbone} - B_{real-time\ protocol}}{\text{number of hops in the line}} \tag{5.1}$$

and is able to forward packets on the backbone every

$$31.25\,\mu\text{s} \cdot \text{number of hops in the line} \tag{5.2}$$

Note that while this schedule is simplistic, as we basically have only one periodic flow, this idea can be extended to multiple flows using methods mentioned in Section 5.2.

5.4 Packet scheduling: Time-Aware Deficit Round Robin

We noted in the previous section that dedicated network-wide periodic time slots are used for real-time traffic. We discuss in this section about the use of the remaining time window between two real-time time slots to transmit so-called best-effort traffic. In the solutions cited in Section 5.2 (FlexRay, Time-Triggered Ethernet and IEEE 802.1Qbv), there is either no best-effort traffic (*i.e.* all packets must be scheduled), no differentiation for best-effort traffic, or a simple priority scheduling mechanism.

Our proposition – and main contribution of this chapter – is to use a (weighted) fair-bandwidth sharing aware for the best-effort time slots. We introduce for this purpose a new packet scheduling algorithm, called *Time-Aware Deficit Round-Robin* (TADRR). It is a variant of the well-known Deficit Round-Robin scheduler from Shreedhar and Varghese [222] presented earlier in Section 3.3.2. This algorithm mixes the two following functionalities:

- It is time-aware, meaning that the scheduler respects specific timing where it is allowed to forward packets or not. This function is used to ensure that the TDMA scheme presented in Section 5.3 is respected.
- It ensures a (weighted) fair distribution of the available bandwidth between the different queues or flows for the time between two real-time time windows.

An illustration of the output of the algorithm is presented in Figure 5.2.

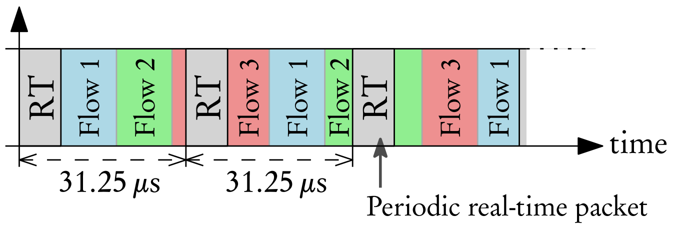


Fig. 5.2: Illustration of Time-Aware Deficit Round Robin

The approach we propose here with TADRR is similar to the hybrid-scheduling approach presented in Chapter 4, namely that we mix a packet scheduling algorithm giving the priority to real-time traffic, and the rest is scheduled with a fair-queuing algorithm. The difference here is on the scheduling of real-time traffic where we use some information on the pattern followed by the packets (*i.e.* its periodicity) in order to optimize their performances, as opposed to traditional strict-priority queuing used in Chapter 4 where this information is not used.

As for the original DRR algorithm, we decomposed TADRR into two modules: an *enqueuing* module presented in Algorithm 5.1 where packets are queues as they arrived to the server; and a *dequeuing* module presented in Algorithm 5.2 which is called each time the transmission of a frame is finished or when a packet arrives and the system is currently idle. We use the same notation as Shreedhar and Varghese in their original paper presenting Deficit Round Robin [222], summarized here in Table 5.1.

The dequeuing module presented in Algorithm 5.2 is a direct modification of the DRR one. We define the two following states: `WAIT_SLOT` where the scheduler has to wait for the

Notation	Signification
Q_i	Quantum allocated to $Queue_i$
DC_i	Bytes that $Queue_i$ did not use during the last round
p	Packet
$C_{backbone}$	Bandwidth of the backbone
IFG	Ethernet inter-frame gap
minEthernetPacketSize	Minimal Ethernet packet size (*i.e.* 64 B)
$Enqueue(i, p)$	Puts packet p into queue i
$Dequeue(i)$	Gets the head-of-line packet of queue i
ActiveList	List of active flows
FreeBuffer()	Frees a buffer from the flow with longest queue

Tab. 5.1: Notation used in Algorithms 5.1 and 5.2

Algorithm 5.1 Enqueuing module of Time-Aware Deficit Round Robin (from [222])

```
Enqueuing module: on arrival of packet $p$
1: $i \leftarrow ExtractFlow(p)$
2: if not $ExistsInActiveList(i)$ then
3:     $InsertActiveList(i)$
4:     $DC_i = 0$
5: end if
6: if no free buffers left then
7:     $FreeBuffer()$
8: end if
9: $Enqueue(i, p)$
```

trigger from the server to be allowed to send, and `ALLOWED_TRANSMIT` where the scheduler is allowed to forward packets. The main idea of the algorithm is that it will serve non-real-time queues following the standard DRR algorithm as long as the head-of-line packets can still be transmitted, *i.e.* their transmission time is smaller than the remaining time before the next RT slot.

We extended in Algorithm 5.2 the dequeuing module from DRR [222] by making the following additions:

Lines 1 to 7 The maximum allowed packet size is computed using the end of the time-slot (*endTimeslot*) and the current timestamp (t). Note that the Ethernet inter-frame gap (*IFG*) is accounted for.

Lines 14 and 19 to 23 We use the previously calculated maximum allowed packet size and check it against the head-of-line packet.

Note that compared to the original DRR algorithm from Shreedhar and Varghese [222], the complexity of the TADRR algorithm is also $O(1)$. But TADRR is *non work-conserving*, *i.e.* it may be idle although packets are ready to be served, as opposed to DRR.

Algorithm 5.2 Dequeuing module of Time-Aware Deficit Round Robin

```
Dequeuing module:
 1: if $state \neq$ ALLOWED_TRANSMIT then return
 2: end if
 3: $allowedPacketSize \leftarrow C_{backbone} \cdot (endTimeslot - t) - IFG$
 4: if $allowedPacketSize < minEthernetPacketSize$ then
 5:     $state \leftarrow$ WAIT_SLOT
 6:     return
 7: end if
 8: while True do
 9:     if $ActiveList$ is not empty then
10:         Remove head of $ActiveList$, say $Queue_i$
11:         $DC_i \leftarrow DC_i + Q_i$
12:         while ($DC_i > 0$) and ($Queue_i$ not empty) do
13:             $packetSize \leftarrow Size(Head(Queue_i))$
14:             if $packetSize > allowedPacketSize$ then break
15:             end if
16:             if $packetSize \leq DC_i$ then
17:                 $Send(Dequeue(Queue_i))$
18:                 $DC_i \leftarrow DC_i - packetSize$
19:                 $allowedPacketSize \leftarrow allowedPacketSize - packetSize - IFG$
20:                 if $allowedPacketSize < minEthernetPacketSize$ then
21:                     $state \leftarrow$ WAIT_SLOT
22:                     return
23:                 end if
24:             else break
25:             end if
26:         end while
27:         if $Empty(Queue_i)$ then
28:             $DC_i \leftarrow 0$
29:         else $InsertActiveList(i)$
30:         end if
31:     end if
32: end while
```

5.5 Cut-through switching and packet scheduling

Note This section is an extension of the original publication [125]. It clarifies some points regarding a possible implementation of the concepts presented in Sections 5.3 and 5.4 in a real network.

We noted in Section 5.1 that in order to reduce the frames' end-to-end latencies, we propose to use cut-through switching. With this frame forwarding method, a switch starts transferring a frame as soon a certain amount of bits of the frame has been received (if no other frame is currently been transmitted on this port). This is an improvement over the more traditional *store-and-forward* method where a switch starts sending a frame only if the complete frame has been received.

In case of standard Ethernet with cut-through switching, the first six bytes of a frame (corresponding to the destination MAC address) first have to be buffered in order to determine which output port to use, and only then the frame can be sent by the output port. In the concept described in Sections 5.3 and 5.4, the same principle is applied for the real-time frames.

For the non real-time frames scheduled here with TADRR as presented in Section 5.4, we note that the algorithm makes use of the frame size in order to schedule packets. To efficiently combine this scheduler with cut-through switching, we need to determine the size of the Ethernet frame before it has been completely received. Since Ethernet does not have a field describing the length of the frame, we have the three following cases:

1. Some encapsulated protocols have a fixed-size frame format, meaning that the frame size can be derived from the Ethertype field. This concept applies to ARP (Address Resolution Protocol) packets for example.

2. The frame size can be derived from the first bytes of the encapsulated packet. In case of IPv4 for example, the first 32 bits of the IPv4 header contain a length field. Other examples are IPv6 or the AVB transport protocol (*i.e.* IEEE 1722).

3. In case the frame size cannot be derived from any headers of the frame or its encapsulated protocols, the complete frame has to be buffered.

In the event that the majority of the packets fall in the third case, cut-through switching will not bring any benefit compared to store-and-forward unless some other parameters of the network configuration can be used or a length field is added via modification of the protocol or encapsulation. An example of this is the network-wide TDMA schedule presented in Section 5.3 and Figure 5.1. We have here a daisy-chain topology where time-slots are reserved for each hop. This means that hops which are not allowed to transmit packet during a specific time-slot will not act as multiplexers, and hence do not need to schedule packets but only need to forward them to the next hop on the daisy-chain. In this example, cut-through switching is still usable for those time-slots.

In the scenarios presented in Section 2.2, the majority of the traffic falls in the first two cases, meaning that we can effectively take advantage of cut-through switching. We note from this section that cut-through switching with packet scheduling is not necessarily straightforward to implement in a real network and might not be applicable to every use-case.

5.6 Evaluation in case of the cabin network

We make in this section a performance evaluation of the scheduling architecture presented in Sections 5.3 and 5.4 using the discrete-event simulator OMNeT++ [17] and its framework INET [14]. The goal is to compare the performances of this architecture with a more traditional one as presented in Chapter 4.

5.6.1 Presentation of the use-cases

We use the star/daisy chain architecture presented earlier in Section 2.2.2 for evaluating the scheduling architectures introduced earlier. We study here three topologies variations as described in Table 5.2, similar to the one used in Chapter 4. We define the *uplink* direction as the direction of the packets from a node connected on a line to the central server, and the *downlink* direction as the opposite direction. The utilization columns correspond to the portion of the backbone bandwidth that is used.

	Configuration		**Devices per domain**			**Utilization** (%)	
Topology	**Lines**	**Aggregators**	**ACD**	**AISD**	**PIESD**	**Downlink**	**Uplink**
A	3	21	18	2	2	15.4	17.4
B	3	21	20	2	2	15.6	13.9
C	4	28	7	1	29	43.7	10.5

Tab. 5.2: Parameters used for evaluated topologies

The results presented here compare two operation modes of the backbone aggregators (*i.e.* “Main Switch” in Figure 4.2):

The cut-through configuration corresponding to the description made in Sections 5.3 and 5.4, with the TDMA schedule and the TADRR packet scheduler;

The store-and-forward configuration where the FPGA acts as a traditional store-and-forward Ethernet switch, without any considerations for the real-time protocol.

5.6.2 Performance evaluation using discrete-event simulations

Note In order to give a more precise quantitative comparison between the two scenarios (cut-through *vs.* store-and-forward), a modified version of Equation (4.3) – which describes the relative difference between the performances – and its application in Figures 5.3b, 5.4b and 5.5b are an extension of the original publication [125].

We follow a Monte Carlo method of simulating multiple runs, each time with a different seed and different initialization vector. We use the following assumptions for the simulation:

- All links are set to 1 Gbit/s;
- Switches processing time is set to 100 ms;
- Queue sizes are set to 1000 packets;
- All the addressing schemes are static.

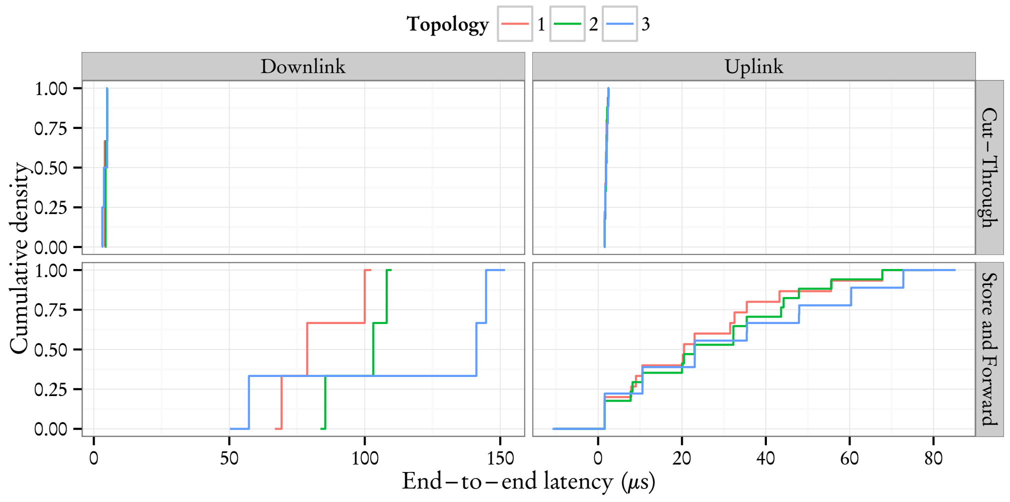
Flows are generated by sending UDP packets at a fixed bandwidth and using a uniform distribution for the packet size. The network supports both the real-time protocol, with its 31.25 μs timing, as well as additional non real-time applications.

End-to-end latency of the real-time protocol

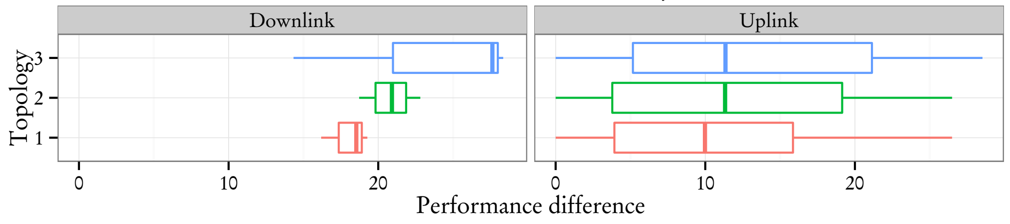
Figure 5.3a presents the maximum experienced end-to-end latency of the real-time 31.25 μs periodic protocol described in Section 5.3. By scheduling appropriately on the cut-through configuration the time when the Ethernet frames of the real-time protocol are sent, we are able to achieve end-to-end latencies below 5 μs for both directions. This is a promising result, as it enables us to have strict feedback loops and it satisfies the requirement set earlier where the two-way end-to-end latency must be below 31.25 μs.

In the store-and-forward configuration, the end-to-end latency is much worse for the real-time protocol and our requirement on the two-way end-to-end latency is not fulfilled. We see then a clear benefit of having dedicated time-slots, where the network is contention free for the real-time protocol.

In order to get a quantitative assessment regarding the performance difference of using the cut-through architecture described earlier, we use here the modified relative error introduced in Section 4.4 with Equation (4.3). The results are presented in Figure 5.3b. We see here that in average we get a performance improvement of almost an order of magnitude.



(a) Maximal end-to-end latency



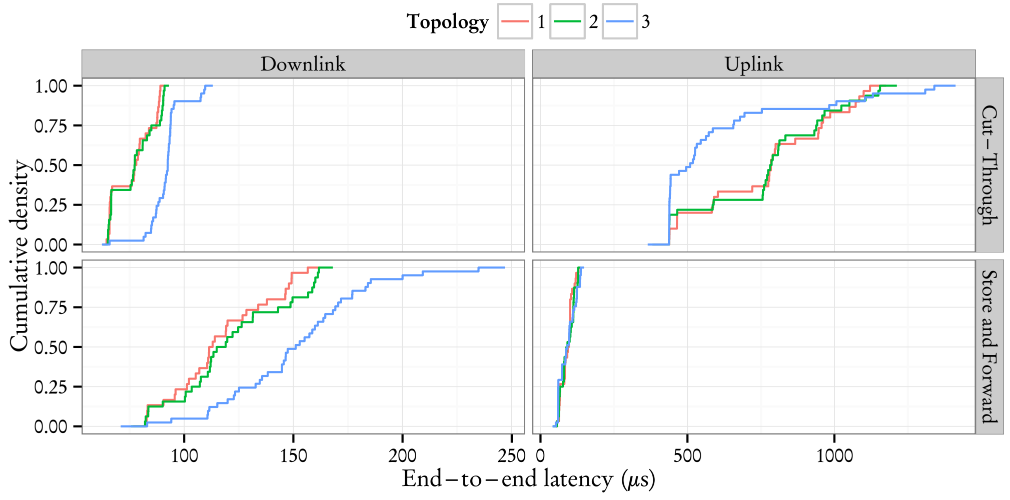
(b) Performance difference for all flows between cut-through and store-and-forward architectures

Fig. 5.3: End-to-end latency of the real-time protocol

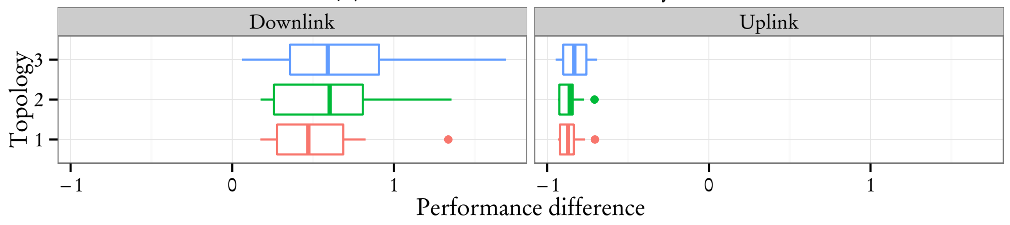
End-to-end latency of the non real-time applications

Figure 5.4a presents the maximum experienced end-to-end latency of the different devices in the topology. Regarding the downlink direction, we see a definitive benefit in the cut-through configuration. Regarding the uplink direction, we see that the round-robin schedule presented in Section 5.3 is sub-optimal compared to the store-and-forward performances, with almost an order of magnitude of difference. This can be explained by the short time window of 31.25 μs where a network node can only transfer a few Ethernet frames. It means that in some cases, packets have to wait multiple cycles of the master-slave protocol in the aggregator queue before being able to be transferred.

As for the real-time traffic, Figure 5.4b presents a quantitative assessment of the performance difference according to Equation (4.3)[1]. While the downlink direction benefits from a average performance improvement of about 50 %, the uplink direction suffers from a average performance decrease of around 80 %.



(a) Maximal end-to-end latency



(b) Performance difference for all flows between cut-through and store-and-forward architectures

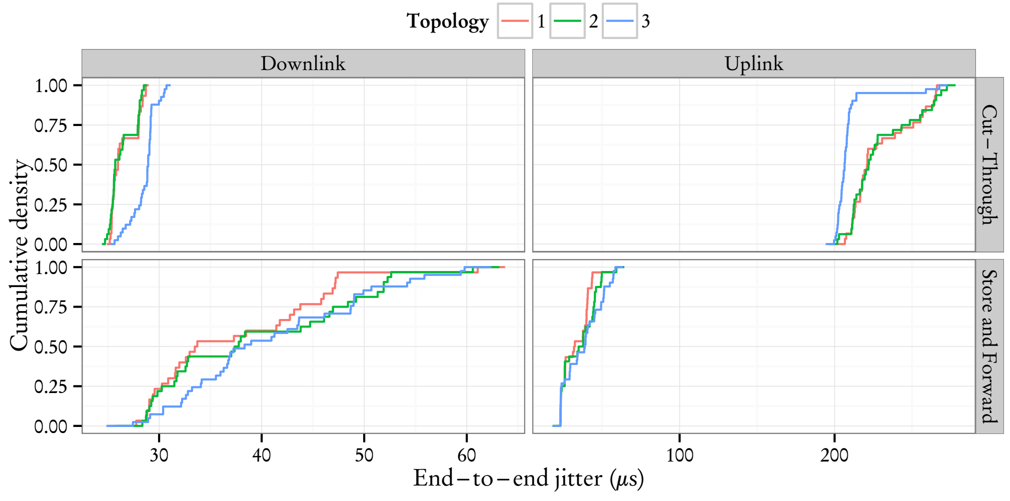
Fig. 5.4: End-to-end latency of the applications

[1] The box-and-whisker plot values used in Figures 5.4b and 5.5b correspond to the default ones used in GNU-R, *i.e.* $\pm 1.5 \cdot IQR$ for left and right whiskers, and 25 %, 50 % and 75 % quantiles for the left, middle and right hinges.

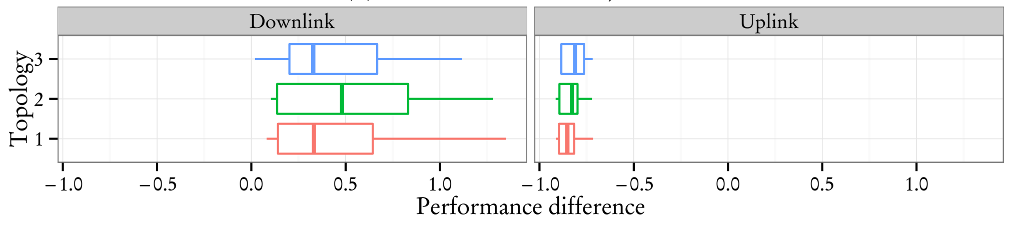
End-to-end jitter measures

As in Section 3.5.1, we used the *interarrival jitter* definition from RFC 3550 [220] for our end-to-end jitter measurement. It is defined as the measure of packet arrival time spacing at the receiver smoothed with an exponential filter with parameter $^1/_{16}$ as shown in Equation (3.7). Regarding the real-time protocol, we have a jitter of 0 because of the contention-free scheduling used here.

Figure 5.5 presents the maximum end-to-end jitter experienced by the different best-effort devices. As for the end-to-end latencies, Figure 5.5b presents a quantitative assessment of the performance difference according to Equation (4.3) (where the latency is substituted by the jitter). The results are similar to the ones of the end-to-end latencies, namely that the downlink direction benefits from a average performance improvement of about 50 % and the uplink direction suffers from a average performance decrease of around 80 %.



(a) Maximal end-to-end jitter



(b) Performance difference between cut-through and store-and-forward architectures

Fig. 5.5: End-to-end jitter of the applications

5.7 Conclusion on time-aware scheduling

In case the mechanisms and scheduling algorithm proposed and evaluated in Chapters 3 and 4 are not sufficient enough with regards to end-to-end delay, we proposed in this chapter a method to achieve low latencies. Via a performance evaluation of a novel packet scheduling structure, we presented in this chapter some key insights regarding the multi-domain Ethernet cabin network architecture presented in Section 2.2.2 and how performances of real-time traffic can be improved.

Due to the multi-hop daisy chain topology inherent to the physical constraints of the network, we have seen in Chapter 4 that a major contributor to end-to-end latency and jitter is the number of hops. Hence, we proposed in this chapter to investigate cut-through switching, a packet-forwarding method which enables us to achieve smaller latencies than with a more traditional store-and-forward mechanism. Due to this method of forwarding packets, special care has to be taken in order to fully exploit its advantages and avoid queuing delays for real-time traffic. A special packet scheduling structure was there for proposed based on three points: *(i)* a network-wide "macro-schedule" based on Time-Division Multiple Access (TDMA) where time-slots are allocated to each member of the network; *(ii)* a network-wide orchestrator using a master-slave protocol to distribute a common time and decide which node is allowed to transmit packets; *(iii)* a novel local "micro-schedule" able to make service differentiation using the information of the macro-schedule, called Time-Aware Deficit Round Robin (TADRR).

Via discrete-event simulations we showed that our scheduling architecture provides a major benefit on the performances of the real-time traffic by improving the end-to-end latency and jitter of downlink packets more than an order of magnitude in average compared to the standard store-and-forward configuration. This increase of performances could be beneficial for applications such as control-loops. Regarding the best-effort traffic, the uplink direction also benefits from a performance increase, although less substantial than the real-time traffic with an average increase of only 50 %.

But as there are two sides to every coin, the benefits for the real-time traffic and the uplink direction come at the cost of poorer performances for the best-effort traffic for the uplink packets with a performance decrease of 80 % in average compared to the traditional store-and-forward architecture. Regarding the TDMA schedule of the different transmissions of the network aggregators, another algorithm than the round-robin scheme used here could bring better performances. As some network aggregators produce more bandwidth than other ones, allocating time windows in accordance to this output bandwidth seems logical. One solution would be to use an offline schedule, which would require careful analytical evaluation of the traffic usage at the different points of the network. Another solution to investigate would be to look at dynamic time-slots allocation, where time-slots are allocated according to the current demand of each aggregator. Finally, an alternative solution to reduce queuing delay for real-time traffic is Ethernet frame preemption, currently under standardization in the IEEE 802.1Qbu standard [8].

Key insights and contributions

Cut-through switching We investigated in this chapter the benefits and drawbacks in term of performance of using cut-through switching. We have seen that if correctly used, gains for real-time traffic can be of an order of magnitude compared to store-and-forward, but it comes at the cost of poorer performances for other traffic.

Performance evaluation Via multiple performance evaluations we gained some insights regarding end-to-end latencies and jitter on the cabin network presented in Section 2.2.2 which can be viewed in the context of the time requirements presented in Table 2.1.

Novel scheduling algorithm We proposed a novel packet scheduling algorithm called Time-Aware Deficit Round Robin enabling us to mix time-triggered real-time traffic with best-effort traffic served using a fair queuing algorithm. We note that while we used it in a specific setting here, this algorithm can also be used in a more traditional network with store-and-forward.

Part III

FLOW-LEVEL NETWORK MODELING

6. INTRODUCTION TO ANALYTICAL PERFORMANCE EVALUATION OF TCP

6.1 Introduction

Chapters 3 to 5 were dedicated to the performance evaluation of the networks presented in Section 2.2 using discrete-event simulation. While this method offers the benefits of ease of use and more concrete results, this technique generally cannot be used to provide flow guarantees useful for certifying the good operation of an avionic network. Hence, we propose in this chapter and the following ones to investigate analytical framework for the performance evaluation of networks, with a focus on TCP traffic. This chapter also serves as an additional motivation for this thesis.

We reviewed in Section 2.3 the various formal methods which were used and proposed for giving guarantees in avionic networks. Those methods are currently mainly focusing on providing delay bounds to single Ethernet frames, which is vital for the reliable operation of applications with short and periodic messages such as sensor readings, actuator directions, closed-feedback loops, or streaming of multimedia flows. Looking at other applications such as file transfers or data loading, those per-packet guarantees are less relevant. One notable application, which will be used in this part and up to the end of this thesis, is bulk data transfer over an elastic transport protocol, where the relevant requirement is the time it takes to transfer data.

Another motivation for this work is that elastic flows accounts for a large portion of the traffic on Internet. Recent measurements in various network environments – such as work from Maier *et al.* on residential broadband traffic in [178], from Labovitz *et al.* on Internet inter-domain traffic in [164], or from Falaki *et al.* on smartphone traffic in [105] – show that TCP accounts for a range from 70 % up to 90 % of the usage. We note that TCP is not limited to bulk data transfers, as is currently also used for Internet video streaming and VoIP (ex: YouTube and Skype) or even real-time traffic such as proposed by Brosh *et al.* in [72].

To this purpose, we propose to investigate mathematical models focusing on elastic flows, with transport layers and above. We focus on one of the most notably used elastic transport protocol, which is the Transport Control Protocol or TCP. This protocol has the following attractive features:

– It provides an elastic service, meaning that it adapts its bandwidth to network conditions using a closed-loop control.

– It provides reliable communications, where applications of higher layers are guaranteed to receive ordered and error-checked data.

The approach which will be used until the end of this thesis is as follows. We first investigate in Chapter 7 the static behavior of TCP traffic in Ethernet networks by studying and modeling infinite flows. We will then extend this work in Chapter 8 by considering the dynamic behavior of TCP flows and look at flows with limited durations. As parameters of those flows can be arbitrarily chosen, we will propose in Chapter 9 a method to give realistic parameters. Finally, we will provide methods to give guarantees to those elastic flows with limited durations in Chapter 10.

Structure of this chapter

We first give in Section 6.2 a brief overview of TCP, how it operates, the various algorithms which have been proposed and which are been actually used in today's operating systems. In Section 6.3, we survey the state of art of analytical methods for studying the performances of TCP. Finally, we summarize and conclude this chapter in Section 6.4 by determining which mathematical framework is the most promising to use and extend for the challenges described in Chapter 1.

6.2 A brief overview of TCP and its different versions

As noted earlier, one important function of TCP is its *congestion control*, where the sending rate of a TCP flow varies according to network conditions. In order to provide reliable connections, TCP works on the principle of data acknowledgments, where the correct delivery of each TCP packet has to be acknowledged by the receiver. Instead of following an inefficient protocol where the sender has to wait on the correct acknowledgment of a packet before sending the next one, TCP uses the notion of *congestion window*. A window corresponds to a certain amount of packets sent at once by the sender. As soon as the sender receives delivery confirmation for at least one data packet, it transmits a new amount of packets. The size of a window, generally noted *cwnd*, and its evolution through time is the main task of TCP and what makes it elastic. A good congestion control algorithm generally has the three following goals: *(i)* eliminating the phenomenon of congestion collapse, where too much traffic might hurt the performance of all the flows, as promoted by Floyd and Fall in [114]; *(ii)* effectively using the available network resources; *(iii)* share the bandwidth between different flows fairly.

As the types of networks where TCP flows are used (in term of latencies, packet drops, buffer sizes, interfering traffic, etc.) as well as the definition of "effective use of the network resources" can vary widely, various congestion control mechanisms and algorithms have been proposed since the early days of TCP. Although those different variants of TCP are quite diverse, they generally follow the recommendations prescribed in RFC 5681 [31]. We refer to the work from Afanasyev *et al.* [26] for a recent survey on the research on congestion control algorithms, the book from Fall and Stevens [106] for a more in-depth explanation of the different TCP congestion control algorithms, and RFC 7414 [102] for a reference document on the standardization of TCP. Despite this large body of work, only a few algorithms have an actual implementation in an operating system or discrete event simulators as shown in Tables 6.1 and 6.2. Note that while unofficial patches are sometimes available for other congestion control algorithms, we only included in Tables 6.1 and 6.2 the ones which are currently available in the official source code of the according kernels and simulators. We included in Table 6.1

kernels used in every-day desktops and servers (mainly Windows NT, XNU, Linux or BSD based), as well as kernels used in embedded devices, as one target of our work is industrial networks (Linux or BSD based for not-resource restricted devices, VxWorks as an example of real-time operating system, and lwIP for a lightweight TCP/IP stack often used in embedded systems). We included in Table 6.2 popular open-source discrete event simulators often used in the literature.

Kernel	**TCP congestion-control algorithms**	**Source code**
Windows NT	Reno [149], Compound TCP* [231]	Not public
XNU[a] *(v. 2782.1.97)*	NewReno [135], CUBIC* [131], LEDBAT [221]	`bsd/netinet/tcp_*`
Linux *(v. 3.18)*	Reno [149], NewReno [135], BIC [245], CUBIC* [131], DCTCP [30], High Speed TCP [112], H-TCP [170], Hybla [75], Illinois [174], LP [163], Scalable TCP [158], Vegas [71], Veno [117], Westwood [183], YeAH [43]	`net/ipv4/tcp_*`
FreeBSD *(v. 10.1)*	NewReno* [135], CUBIC [131], Vegas [71], H-TCP [170], HD [74], CHD [133], CDG [134]	`sys/netinet/cc/*`
4.4BSD-Lite2[b]	Reno* [149]	`sys/netinet/tcp_*`
VxWorks	Reno* [149]	Based on 4.4BSD-Lite2[c]
lwIP[d] *(v. 1.4.1)*	Reno* [149]	`lwip/src/core/tcp*`

[a] XNU is the kernel used for Apple's operating systems: Darwin, Mac OS X and iOS.

[b] 4.4BSD-Lite2 is included here as its network stack served as a base of a lot of more modern operating systems.

[c] Although the source code for VxWorks is not public, the documentation states that the network stack is based on 4.4BSD-Lite2 (which uses TCP Reno). Also, there are no explicit parameters to specify which congestion-avoidance algorithm to use, which leads us to conclude that Reno is used as the default algorithm.

[d] While lwIP is not a kernel in itself, it is a lightweight open-source TCP/IP stack which is often used in embedded systems or some real-time operating systems (ex: FreeRTOS)

Tab. 6.1: TCP algorithms available in current version of kernels. Default congestion control algorithms are marked with "*".

As noted by Allman and Falk in [32], one important question to address when making research on TCP is to choose which version to use. Yang *et al.* recently reported in [246] that the current dominant TCP versions used in web-servers are, by order of popularity:

- BIC and CUBIC [245, 131] with 46.9 %;
- Compound TCP [231] with 13.4 % to 24.57 %;
- Reno [149] with 3.3 % to 14.5 %.

This is a direct echo of the current market share of web-servers which is dominated by Linux at the moment. From Table 6.1, we see that for embedded devices only Reno is (officially) available. Hence, for the rest of this work we will focus on the two following congestion control algorithms:

Simulator	TCP algorithms
ns-2 (v. 2.35)	Reno* [149], NewReno [135], Vegas [71], algorithms from Linux[a]
ns-3 (v. 3.21)	Tahoe, Reno* [149], NewReno [135], Westwood [183], algorithms from Network Simulation Cradle[b]
OMNeT++/INET *(v. 4.6 and 2.5.0)*	Tahoe, Reno* [149], NewReno [135], Vegas [71], Westwood [183], algorithms from Network Simulation Cradle and lwIP
GTNetS *(v. Oct-10-08)*	Tahoe*, Reno [149], NewReno [135]

[a] The source code of the congestion control algorithms from Linux version 2.6.22.6 have been imported in *ns-2* (see [240]). Via minor modifications, this process can be reapplied to recent versions of the Linux kernel.

[b] The source code of the congestion control algorithms from Linux version 2.6.10, 2.6.18, 2.6.26, lwIP, OpenBSD 3.5 and FreeBSD 5.3 have been imported in the Network Simulation Cradle (NSC) [150].

Tab. 6.2: TCP algorithms available in current version of open-source discrete event simulators. Default congestion control algorithms are marked with a "*".

Reno due to its ubiquitous availability in all kernels and discrete event simulators presented in Tables 6.1 and 6.2, the large body of research it has attracted, and the fact that when other protocol are made "TCP friendly" as defined by Floyd and Fall in [114], it refers generally to Reno, like for example in RFC 5348 [113];

CUBIC as it is the current default congestion control algorithm used in Linux since November 2006[1] and in XNU.

The mathematical models developed in Chapters 7 and 8 will mainly focus on Reno while giving some pointers and references on similar models for CUBIC.

6.2.1 Standard TCP behavior

Although there is no unanimous definition of "standard TCP behavior", we take RFC 5681 [31] as the baseline for describing the recommended behavior of TCP. This RFC is based on Reno. We describe here shortly how TCP congestion control works.

In order to be able to adapt its behavior to network conditions, which are not known when a TCP connection is initiated, a *slow start* phase first occurs in order to probe the state of the network. During the phase, the congestion window *cwnd* is incremented by one TCP segment after each correctly received acknowledgment. This phase ends when *cwnd* reaches a certain threshold value (generally referred as *ssthresh*) or when congestion is detected. Congestion is detected by the reception of a triple duplicate acknowledgment or via a timeout.

After the slow-start phase, the *congestion avoidance* phase starts and lasts until the end of the connection. During this phase *cwnd* is updated as follows after each correctly received acknowledgment [31]:

$$cwnd_{n+1} = cwnd_n + \frac{(\text{segment size})^2}{cwnd_n} \tag{6.1}$$

[1] Reno was the default TCP congestion control algorithm until Linux 2.6.8 (August 2004). With this version, BIC was set as the default. CUBIC was then set default starting with Linux version 2.6.19 (released November 29, 2006) with commit `597811ec167fa01c926a0957a91d9e39baa30e64`, until now (July 2015).

Note that in Equation (6.1), *cwnd* is expressed in Bytes and is usually a multiple of the segment size.

When congestion is detected during the congestion avoidance phase, *cwnd* is divided by 2, unless a retransmission timeout occurs, in which case the slow-start phase starts again. This general congestion avoidance algorithm is also called Additive Increase Multiplicative Decrease (AIMD).

6.3 State of the art on performance evaluation of TCP traffic

Note This section is partially based on our previous publications [120] published in *Proceedings of the 6th International Conference on Simulation Tools and Techniques*, 2013, [121] published in *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, 2013, and [123] published in *Proceedings of the 39th IEEE Conference on Local Computer Networks*, 2014.

Regarding performance evaluation of TCP traffic, we distinguish four approaches:

Mathematical frameworks as presented with a short survey later in this section;

Simulations with discrete event tools such as *ns-2* [15], *ns-3* [16] or OMNeT++ [17];

Emulations with tools such as mininet [165] or dummynet [207];

Measurements on real networks either on private ones, or public ones such as PlanetLab [90] or GENI (Global Environment for Network Innovations) [13].

While each approach has its benefits (such ease of use, applicability or precision) and drawbacks (such as reproducibility or imprecision), we will focus in this chapter and the subsequent ones on mathematical models. By having an accurate and faithful mathematical formulation of the performance of TCP flows, we can enable effective traffic engineering practice.

We give in this section an overview on the different models which have been proposed for the performance evaluation of TCP traffic.

6.3.1 A categorization of the mathematical models

Current works on performance evaluation of TCP flows generally try to put mathematic models on the proposed congestion algorithms. Those work can be divided in the following (non-exhaustive) orthogonal categorizations:

Packet-level or flow-level Model can either model the detailed dynamics of the window size of TCP flows or focus more on a higher layer and abstract the behavior of TCP. In case of QoS policies for elastic traffic, requirements are generally given at the flow-level (compared to packet-level for real-time traffic).

Dynamic or statistical behavior Models are either involved in the dynamic behavior of the throughput of TCP flows, meaning characterizing *cwnd* as a function of time, or its statistical behavior, meaning statistical indicators on the evolution of *cwnd* such the average or some percentile values. In case of QoS policies, we are generally more interested into giving guarantees on the statistical behavior of TCP flows (mean or tail performances).

Infinite or short flows Models can either focus on the behavior of infinite TCP flows, meaning flows where there is always data to transmit, short flows, meaning flows with a finite amount of data to transfer, or a combination of both. Regarding QoS policies, the three characterizations are interesting and actually depend on the requirements on the flows.

Single or multiple flows Models can either target the behavior of a single TCP flow or multiple ones. As networks generally share more than one flow in practice, models targeting multiple flows are more useful.

Realistic or ideal bandwidth sharing When modeling multiple interacting TCP flows, the characterization of the bandwidth sharing can either be based on a congestion algorithm which has been implemented or a purely mathematical model where there are no real implementations available. Regarding QoS policies, the first approach is the most suitable as it maps to reality. Nevertheless, the second approach has also some practical applications as the bandwidth sharing of multiple TCP flows can be changed toward a purely mathematical one using correct packet scheduling algorithms.

Uniform or mixed congestion control algorithms Alongside the previous point, models either assume that all nodes on a network share the same congestion control algorithms or not. As we are in an industrial setting where we have more control over the targeted network and its network stacks, we do not put a special emphasis on this point.

Realistic or ideal network elements Models can either account for the characteristics (such as influence on latency or packet drop via queuing mechanisms) of the elements which are traversed by the flows (network card, switches, routers, firewall, etc.) or treat them as ideal elements. As we are targeting small Ethernet networks with low latencies where the effect of queue size or packet processing time have an influence on the performance of flows, we are more interested into models trying to incorporate the effect of the traversed network elements.

Basic or advanced network elements Alongside the previous point, models can either characterize traversed network as basic, meaning a simple queue, or advanced, meaning scheduling algorithms. Regarding QoS policy, taking into account scheduling algorithms is more beneficial.

Single or multiple bottlenecks Models can either be restricted to topologies with single bottlenecks, for the purpose of simplification or due to limitations of the underlying model, or can be used on more general topologies with more than one bottleneck. Regarding practical use, we are generally more interested into models able to handle multiple bottleneck topologies.

Basic or advanced effects of TCP behavior Depending on the topology used and its parameters, the bandwidth sharing of TCP might be counterintuitive. Among those effects, there are for instance the effect of cross-traffic (discussed later in Section 7.4.2), or TCP Incast (see [88]). In order to be able to define an effective QoS policy, a good model should include counterintuitive effects which have the biggest impact on TCP behavior.

6.3.2 The different approaches

While the body of work on performance evaluation of TCP is quite large, we survey here the major models matching the different criteria noted earlier. This means that we do not discuss here about models for single TCP flows, nor models targeted at specific topologies outside of the scope of this thesis.

In order to efficiently compare the different models presented hereafter, Table 6.3 defines common mathematical notations. Some works discussed below make use of matrix and vector operations, hence we define $\vec{x}$ as being the vectorized version the underlying x variable, and x_i an element of this vector. Note that the presentations of different mathematical formulations have been simplified, as we intend to only give an overview of the approaches and not formal definitions and their associated proofs.

Variable	Signification	Variable	Signification
t	Time	k	Queue index
i	Flow index	q_k	Queue size
r_i	Flow bandwidth	$\mathcal{F}_k$	Set of flows traversing queue k
RTT_i	Flow round-trip time	R	Routing matrix
w_i	Flow weight	C_k	Link capacity
$\mathcal{S}_i$	Set of queues traversed by flow i	p_k	Link drop probability

Tab. 6.3: Common mathematical notations used in Section 6.3.2

Network utility maximization

This approach, called *network utility maximization* or NUM, is based on the idea that the interaction between different elastic flows is a distributed optimization problem defined as:

$$\begin{aligned} \underset{\vec{r}\geq 0}{\text{maximize}} \quad & \sum_{\forall i} U_i(r_i) \\ \text{subject to} \quad & R\vec{r} \leq \vec{C} \end{aligned} \tag{6.2}$$

with U_i the utility function of flow i.

This idea was initially presented by Kelly *et al.* in [157], in which the authors describe various ideal bandwidth sharing utility functions. A particular one is the so-called *proportionally fair* bandwidth allocation, where each flow receives a fair amount of bandwidth proportional to the flow weight. This is similar to the Weighted Fair Queuing scheduling presented in Section 3.3.2. The utility function of this particular bandwidth sharing is:

$$U_i(r_i) = w_i \log(r_i)$$

Various other utility function based on ideal bandwidth sharing functions were proposed by Massoulié and Roberts in [184].

This idea was then adapted by Low in [177] to describes the utility functions of two specific TCP variants in conjunction with queuing policies: *(i)* TCP Reno with a Random

Early Detection or RED (see [115]) queuing policy; *(ii)* TCP Vegas with drop-tail queuing policy. The utility functions are defined as:

$$U_i^{\text{Reno}}(r_i) = \frac{\sqrt{3/2}}{RTT_i} \arctan\left(\sqrt{2/3}\, r_i RTT_i\right) \tag{6.3}$$

$$U_i^{\text{Vegas}}(r_i) = \alpha_i RTT_i \log(r_i) \tag{6.4}$$

with α_i a parameter of Vegas.

While those initial works were restricted to infinite flows, Chang and Liu proposed in [84] to adapt this approach to short flows alternating between think and transfer times, but with the restriction of a single bottleneck topology. More recently the NUM approach was extended by Ge and Tan in [118] to include the cross-traffic effect on TCP. We also note that the idea of using network utility maximization has been extended to other layers of the network stack, sometimes also modeling cross-layer interactions, as presented for instance in the survey [89] from Chiang *et al.*.

While this is a promising approach and it has been used in a wide range of applications, we note the following drawbacks with regards to our domain of application: *(i)* flow RTTs in Equations (6.3) and (6.4) are considered to be constant and known beforehand; *(ii)* queue size and scheduling algorithms are generally not taken into account; *(iii)* the approach for short flows is limited to topologies with single bottlenecks.

Fluid models with stochastic differential equations

A approach for studying TCP flows is based on the idea that the behavior of congestion control can be described as a set of coupled differential equations:

$$RTT_i(t) = \sum_{k \in \mathcal{S}_i} \frac{q_k(t)}{C_k} \tag{6.5}$$

$$\frac{\mathrm{d}q_k(t)}{\mathrm{d}t} = \sum_{i \in \mathcal{F}_k} \frac{W_i(t)}{RTT_i(t)} - \mathbb{I}\{q_k(t)\}\, C_k \tag{6.6}$$

$$\mathrm{d}W_i(t) = \frac{\mathrm{d}t}{RTT_i(t)} - \frac{W_i(t)}{2}\mathrm{d}p_i(t) \tag{6.7}$$

with $W_i(t)$, the TCP window size of flow i. Equation (6.5) describes the RTT of flow i as being the sum of the queue size of the traversed servers. Equation (6.6) describes the size of queue k as a differential version of the Lindley equation. Finally, Equation (6.7) describes the evolution of the window size as increasing after each round trip, and divided by two after each drop (*i.e.* it follows an AIMD congestion control). This approach is called *fluid modeling* as there is no notion of packets in Equations (6.5) to (6.7).

This model was initially proposed by Misra *et al.* in [191] and [192], and it was shown to accurately model the dynamic behavior of TCP flows. Liu *et al.* then extended in [175] the work from Misra *et al.* in order to address deficiencies of the initial model and include other variants of TCP, namely SACK, Reno and New-Reno. The influence of cross-traffic was later addressed by Barbera *et al.* in [48].

Baccelli and Hong proposed a similar idea in [39] and [40], where the evolution of the TCP window size is a function of the previous window size and state of the network. More recently, the initial work from Baccelli and Hong on Reno was then extended to Cubic by Belhareth *et al.* in [55].

The main drawback of this approach is that it is not adapted to our goal of having statistical behaviors. Indeed, the numerical evaluation of the equations in the previously cited publications is generally comparable to network simulators, via the use of time-stepped simulations or Runge-Kutta methods for instance.

Queuing theory based models

A natural way to model the network is to use the well-studied queuing theory. Using this theory, one has to decide which level of granularity one wish to have – individual packets or complete flows – as well as which analytical fits the best the input data.

Invidual packets granularity Casetti and Meo proposed in [81] to look at the packet level and model the different queues in the network as $M/D/1/K$, which corresponds to drop-tail FIFO queues where TCP packets are assumed to arrive following a Poisson process, with a constant packet size. In order to model the evolution of the TCP window size, the authors proposed to use a Markov chain where each state of the chain correspond to the triplet s:

$$s = (cwnd, W_t, l)$$

with $cwnd$ the current window size, W_t the current value of the window threshold when the TCP connection switches from slow-start to congestion avoidance, and l the indication that a loss occurred by was not yet detected.

The transitions between the different states of the chain are dictated by which TCP version is used. The original work from Casetti and Meo only characterized the evolution for TCP Tahoe. Wierman *et al.* later proposed to characterize TCP Reno, Vegas and SACK, in conjunction with a $M/M/1/K$ queue in [243].

Flow-level granularity When looking at the flow level and abstracting the packet level behaviors, a widely used approach is to model the bandwidth sharing of TCP as a fair sharing queue, as presented in Section 3.3.2. In this model, each flow receives an equal (or weighted) share of the bandwidth. The slow-start phase of TCP is neglected, such that when flows leave or enter the network, the other flows adapt instantaneously their bandwidth to the new number of active flows. The mathematical basis for this approach was developed long before the existence of TCP, and is presented in great details by Cohen in [92] and Kleinrock in [161]. Using such models, one can infer the statistical behavior of flows, like for instance in an $M/G/1$ fair sharing queue with or without weights, also sometimes referred as $M/G/1$-PS or $M/G/1$-DPS for the weighted variant, the number of active flows is characterized by the following distribution:

$$\pi(n) = \rho^n(1-\rho)$$
$$\rho = \frac{\lambda\sigma}{C}$$

with $\pi(n)$ the probability that n flows are in progress, ρ the link load, λ the mean inter-arrival time and σ the mean file size.

Models using queuing theory with a flow level granularity have gained the most attraction, as advocated for instance by Ben Fredj *et al.* in [57] or Kherani and Kumar in [159].

The main advantage of the queuing theory based approaches is that it can provide scalable statistical behavior of flows via well-known mathematical methods with closed-form solutions. The main drawback is that the results might not map to real network behaviors due to the high level of abstraction needed in order to have tractable results.

Multi-layer fixed-point models

The final approach presented here makes use of models proposed in the late 1990's and early 2000's describing the packet-level behavior characterizing the throughput of a single TCP connection as a function of round-trip time RTT_i and loss probability p_i. The most notable ones are the so-called square-root formula from Mathis *et al.* [185], and the PFTK formula from Padhye *et al.* [195]:

$$\text{Square-root formula [185]:} \quad r_i(RTT_i, p_i) = \frac{K_0}{RTT_i\sqrt{p_i}} \tag{6.8}$$

$$\text{PFTK formula [195]:} \quad r_i(RTT_i, p_i) = \frac{K_1}{RTT_i K_2\sqrt{p_i} + K_3 \min\left(1, K_4\sqrt{p_i}\right) p(1 + K_5 p_i^2)} \tag{6.9}$$

with K_i various constants of the models.

While those early models only describe the behavior of a single TCP flow, a similar approach can be applied to queues in the network, by defining the packet loss probability and queue size as a function of the flows traversing the queues. For instance the drop probability can be defined as:

$$p_i(\mathcal{F}_k) = \frac{\left[\sum\limits_{i\in\mathcal{F}_k} r_i - C_k\right]^+}{\sum\limits_{i\in\mathcal{F}_k} r_i} \tag{6.10}$$

which satisfies that the sum of the bandwidths of the flows traversing a queue ($\sum_{i\in\mathcal{F}_k} r_i$) does not exceed the available capacity C_k, or in mathematical terms:

$$\sum_{i\in\mathcal{F}_k} r_i \leq C_k, \forall k.$$

The combination of those two characterizations, results in two equations (the flow and the queue functions) with two unknowns (RTT_i and p_i). The numerical evaluation of those equations is generally performed using fixed-point evaluations. Gibbens *et al.* proposed in [126] one of the early models on this approach using the square-root formula to study the bandwidth of multiple TCP flows on networks with multiple bottlenecks modeled as $M/M/1/K$ queues. Firoiu *et al.* in [110, 111] and Bu and Towsley in [73] later proposed a similar model based on the PFTK formula for arbitrary networks with TCP and non-TCP flows and RED queue management. Various mathematical formalisms and proofs for flow-level models were then proposed by Altman *et al.* in [33] by building on the results of Bu and Towsley [73].

The models previously cited were extended by Hassan *et al.* in [132] to include scheduling algorithms, namely priority queuing and weighted fair queuing.

The main advantages of this approach are that it uses accurate packet-level behavior of TCP bandwidth sharing and it is applicable to topologies with multiple bottlenecks with a variety of scheduling policies. The main drawbacks are that it can only handle infinite TCP flows and advanced effects on TCP behavior are not taken into account.

6.4 Conclusion on performance evaluation analytical frameworks

Table 6.4 summarizes the different approaches described in Section 6.3.2 according to the categorization presented in Section 6.3.1. From this categorization and our domain of application, we note that an ideal approach should fall in the following categories:

Statistical behavior of the flows for evaluating the behavior of short flows;

Multiple flows as we are interested in the interaction between multiple flows;

real bandwidth sharing in order to map our model to real networks, but we note that a model using only an ideal or artificial bandwidth sharing might also be of use as there are practical ways (*i.e.* packet scheduling) to enforce such artificial bandwidth sharing in real networks;

Mixed congestion control algorithms as default TCP versions used in practice may change;

Real and advanced network elements for being able to map to actual network elements (*i.e.* switches, queues and packet scheduling) used in practice;

Multiple bottlenecks topologies as the network we are studying have generally more than one bottleneck;

Address some counterintuitive effects of TCP behavior for being able to anticipate topologies impacting TCP performances.

From Table 6.4 we see that no approach fits all the points of the ideal model described earlier. Nevertheless, we note that a promising approach is the multi-layer fixed-point one as it fits eight out of ten of the criteria described earlier, the missing points being that it only deals with infinite flows and does not include advanced effects of TCP behavior. Hence, we propose to focus in the rest of this thesis on this approach and address those shortcomings.

Key insights and contributions

Choice of relevant TCP variant We gave a brief overview in this chapter of the different variants of TCP which have been proposed, and surveyed which ones are actually used in today's operating systems. This led us to focus on mainly on TCP Reno for the next chapters.

Overview of different mathematical frameworks We surveyed in this chapter the different analytical frameworks for the performance evaluation of UDP and TCP flows in Ethernet networks (Research Objective **O2.2**).

Cat. \ Approach	**Utility max.**	**Fluid models**	**Queuing theory**	**Fixed-point**
Level	Flow	Packet	Packet/Flow	Flow
Results	Statistical	Dynamic	Statistical	Statistical
Flow length	Inf. and short	Inf. and short	Short	Inf.
Number of flows	Multiple	Single/Multiple	Multiple	Multiple
Bandwidth sharing	Realistic/Ideal	Realistic	Ideal	Realistic
Congestion control	Mixed	Mixed	Uniform	Mixed
Network elements	Ideal	Realistic	Ideal	Realistic
Network elements	Basic/Advanced	Advanced	Advanced	Advanced
Bottlenecks	Single/Multiple	Single/Multiple	Single/Multiple	Multiple
Advanced effects	Cross-traffic	Cross-traffic	None	None

Tab. 6.4: Comparison of the different approaches for modeling networks with TCP flows. Lines where a slash is present (ex: "A/B") mean that some variants of the approach are restricted to either A or B.

Choice of which method will be used for the subsequent chapters Via the assessment made in Table 6.4 we concluded that the multi-layer fixed-point approach is the most promising one (Research Objective **O2.2**).

7. FLOW-LEVEL MODELING OF PERSISTENT TCP FLOWS

Note This chapter is based on our previous publications [121] published in *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, 2013 and [124] published in *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, 2014. Section 7.5.2 is an extension of the original publication. Section 7.8.1 has also been partially extended to better highlight the parameters which may impact the accuracy of the model.

7.1 Introduction

We reviewed in Section 6.3 the different mathematical models which have been proposed for the performance evaluation of networks with TCP flows. The approach we selected is the multi-layer fixed-point models – also called flow-level network modeling – briefly introduced in Section 6.3.2. The main idea of this approach is to reuse well-established models of TCP characterizing the bandwidth of a TCP flow as a function of the path drop probability and round-trip time with models of queues and links characterizing the evolution of drop probability and latency as a function of their traversed bandwidth.

We propose in this chapter to give a unified formalism of this mathematical framework and extend it to our use-case, namely Ethernet networks with low latencies. The contributions of this chapter are threefold. First we formalize and extend the initial framework to the characteristics of Ethernet networks, namely FIFO queues and packet scheduling algorithms traditionally found in Ethernet networks (such as the ones presented in Section 3.3). Our second contribution is to model the impact of cross-traffic on TCP flows which is often neglected in previous studies using a similar approach, as we show that it can lead to major errors on the evaluation of performances of TCP on specific topologies. This phenomenon of cross-traffic is well known in traffic engineering and was firstly attributed to ACK compression by Zhang *et al.* in [252] and more recently the principle of data pendulum by Heusse *et al.* in [137]. Finally, our third contribution is a tool implementing the various mathematical models presented in this chapter.

Our solution takes into account this phenomenon by including TCP acknowledgments into our flow-level network model. With our framework, we aim at evaluating Ethernet LANs where nodes communicate using TCP or UDP, and give the following results: average throughput, end-to-end delay and loss probability. We also extend our approach to network supporting the general packet scheduling algorithms presented in Section 3.3, namely Strict-

Priority Queuing (SPQ), packetized versions of Generalized Processor Sharing (GPS) such as Weighted Fair Queuing (WFQ), as well as hierarchical scheduling based on the former algorithms.

Structure of this chapter

In Section 7.2, we present similar research studies. Section 7.3 highlights the basic principles of our framework, while details about flow modeling are introduced in Section 7.4, and details about queues and schedulers are given in Section 7.5. We present in Section 7.6 our algorithm for finding a solution to the model. We introduce in Section 7.7 PETFEN, our tool implementing the results presented in Sections 7.3 to 7.6. With Section 7.8, we evaluate our framework across different topologies where we highlight the flaws of a model not taking into account cross-traffic. Finally, Section 7.9 summarizes and concludes our work, and gives an overview of future improvements for our framework and our tool.

7.2 Related work

An overview of the origins of the analytical model used here was presented previously in Section 6.3.2 paragraph *Multi-layer fixed-point models* (Page 82). The following works were cited: [185, 195, 126, 111, 73, 33, 132]. We refer to Chapter 6 for a broader survey on the related analytical frameworks for the performance evaluation of TCP flows in networks with multiple bottlenecks.

Regarding one of the problem addressed in this chapter, Velho and Legrand noted in [233, 234] that previous work on flow-level modeling did not include the effect of cross-traffic on TCP flows. They proposed a solution to overcome this problem by including TCP acknowledgments flows into a fixed point formulation using a RTT-aware max-min model. While the solution proposed in [234] seems appropriate for their studied use cases, it is not clear if the evaluation of the TCP model takes account of other behavior of TCP than RTT-unfairness, such as TCP timeouts. Indeed, the work that lead to the PFTK formula showed that TCP timeouts have a significant impact on TCP sending rate. We present in this chapter a solution to the cross-traffic problem based on the early work from Firoiu *et al.* in [111].

7.3 Mathematical framework for flow-level network modeling

7.3.1 Elements of the studied network

We define the following assumptions for the topologies studied in this chapter. We target the performance evaluation of Ethernet Local Area Networks (LANs) where entities communicate using standard Ethernet. Computers are interconnected through Ethernet switches and communicate with each other either by using protocols on top of TCP, or by using fixed rate flows (streaming) which is considered here to be UDP based. For the scope of this chapter, we consider that all communications are unicast and that the routing is static, meaning that we have a single path between a source and a destination.

The network is composed of Ethernet switches functioning on the principle of store-and-forward, meaning that switches need to first receive and store the complete frame before being able to forward it, as opposed to the principle of cut-through. Links between nodes of the network are assumed to be Ethernet cables, and can have different link speeds. A switch can have an internal processing delay for each frame. As we study Ethernet LANs with low latencies, meaning networks where queuing delay has a large influence on end-to-end delays, we do not neglect queuing delay in switches.

When discussing about packet size and flow throughput in the rest of the chapter, we consider them from the Ethernet point of view. In order to also take into account the preamble, start of frame delimiter and interframe gap of Ethernet, the packet size shall account for it.

7.3.2 Flow-level network model

Our flow-level network model consists of *servers*, which model the different queues of the network, as well as *flows*, which represent the communications between the nodes of the network.

We define a *server* as an entity receiving packets and forwarding them on a link. A server, noted here s_k with $k \in \mathbb{N}$, is defined by the following parameters:

- C_k is the maximum output bandwidth;
- D_k is an additional delay, which can be used to model propagation and processing delay;
- $\mathcal{F}_k = \{f_n\}_k$ is the set of flows going through this server;
- Q_k the buffer size of the server as the result of the function $H_k^Q(F)$ depending on a set of flows $\mathcal{F}$;
- p_k the drop probability of the server as the result of the function $H_k^p(\mathcal{F})$ depending on a set of flows $\mathcal{F}$.

Details about the functions H_k^Q and H_k^p depend on which model to use and will be described in Section 7.5.

We define a *flow* as a sequence of packets sent from a particular source to a particular unicast destination of a specific transport connection or media stream. A flow, noted here f_i with $i \in \mathbb{N}$, is defined by the following parameters:

- $\mathcal{S}_i = \{s_n\}_i$ is the path of servers traversed by the flow from source to destination;
- $\overline{\mathcal{S}_i}$ is the path which will be used for the reply packets of flow f_i if a protocol is specified by requests and replies;
- r_i is the bandwidth of a flow at its source as the result of the function $\rho_i(\mathcal{S})$ depending on the path of servers S

We also define the throughput of a flow as the rate of successful message delivered to the destination. Details about the function ρ_i depend on which model to use and will be described in Section 7.4.

Based on those parameters, we describe the behavior of a network using the axioms presented hereafter.

Axiom 7.1. *The end-to-end drop rate $e2e_p$ of the path of servers S is defined by:*

$$e2e_p(\mathcal{S}) = 1 - \prod_{k \in \mathcal{S}} (1 - p_k) \tag{7.1}$$

Axiom 7.2. *The aggregated ingress bandwidth of server s_k is defined by the sum of bandwidth of the set of flows $\mathcal{F}_k$ traversing the server:*

$$B_k^{inp} = \sum_{i \in \mathcal{F}_k} \left[r_i \cdot (1 - e2e_p(\mathcal{U}(\mathcal{S}_i, s_k))) \right] \tag{7.2}$$

where $\mathcal{U}(\mathcal{S}_i, s_k)$ corresponds to the set of servers the flow i traverses before reaching s_k.

We account in Equation (7.2) for the fact that part of the bandwidth of the traversing flows is already dropped on the different paths leading to the studied server (noted here $\mathcal{U}(\mathcal{S}_i, s_k)$)).

Axiom 7.3. *The egress bandwidth of server s_k is equal to:*

$$B_k^{out} = (1 - p_k) \cdot B_k^{inp} \tag{7.3}$$

and must satisfy the constraint:

$$B_k^{out} \leq C_k \tag{7.4}$$

Axiom 7.4. *The end-to-end delay $e2e_D$ of a frame of size M along the set of servers S is defined by:*

$$e2e_D(\mathcal{S}, M) = \sum_{k \in \mathcal{S}} ((M + Q_k) \cdot C_k + D_k) \tag{7.5}$$

We account in Equation (7.5) for the forwarding time of the frame ($M \cdot C_k$), the time needed to process the queue ($Q_k \cdot C_k$) as well as an additional delay D_k for modeling propagation and processing delay.

Axiom 7.5. *The round-trip time for a flow with a request of size M_{req} and a reply of size M_{rsp} is:*

$$RTT(\mathcal{S}, M_{req}, M_{rsp}) = e2e_D(\mathcal{S}, M_{req}) + e2e_D(\overline{\mathcal{S}}, M_{rsp}) \tag{7.6}$$

A summary of the different notations is presented in Figure 7.1.

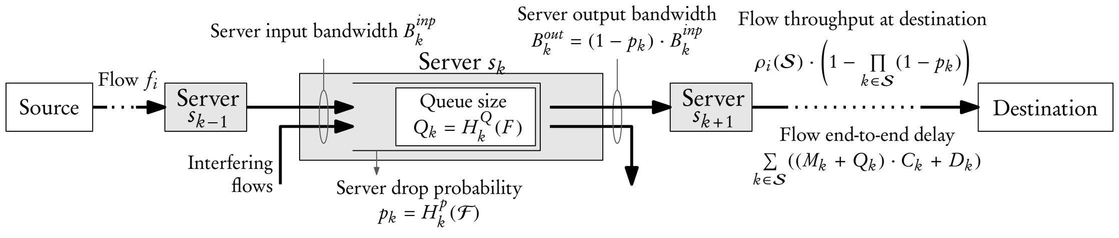

Server input bandwidth B_k^{inp}
Server output bandwidth
$B_k^{out} = (1 - p_k) \cdot B_k^{inp}$
Flow throughput at destination
$\rho_i(\mathcal{S}) \cdot \left(1 - \prod_{k \in \mathcal{S}} (1 - p_k)\right)$
Server s_k
Flow f_i
Source
Server s_{k-1}
Queue size
$Q_k = H_k^Q(F)$
Server s_{k+1}
Destination
Interfering flows
Server drop probability
$p_k = H_k^p(\mathcal{F})$
Flow end-to-end delay
$\sum_{k \in \mathcal{S}} ((M_k + Q_k) \cdot C_k + D_k)$

Fig. 7.1: Summary of the different notations used in this chapter

7.4 Flow models

The goal of the flow model is to define the function $\rho_i(S)$ representing the long-term average bandwidth of the flow as a function of the set of servers S traversed by the flow. We present in this section models of two types of flows: constant bitrate flows representing multimedia streaming based on UDP, and long-lived TCP flows.

Note that the formulas described here model large timescales (in the order of minutes to hours) where the effect of the burstiness of TCP can be neglected. Those effects are generally only visible for timescales in the order of magnitude of round-trip times. We refer to some of the approaches surveyed in Section 6.3.2 for models which are more focused on those short-term effects. Section 7.8.1 will highlight the sensitivity of the results with respect to which timescales are used.

7.4.1 Long-lived TCP flow model

The congestion control algorithm of TCP works in two different phases. The first phase, called *slow start* as in RFC 5681 [31], occurs at the beginning of the TCP connection and is used to estimate the link capacity. During this phase, only a small amount of data is transmitted. Once this phase is finished, a *congestion avoidance* phase takes place and transmits the rest of the data. For this study, we consider that TCP is used to transfer large data, meaning that we only account for the *congestion avoidance* phase, and we call this type of flows *long-lived TCP flows*.

Following our reasoning in Section 6.2, we focus here on TCP Reno [149]. Other congestion avoidance algorithms may be included following the methodology presented here. Table 7.1 presents other analytical packet-level models for other variants of TCP which may be used with our framework, although some models have limitations regarding applicability. We noted in Section 6.2 that we would also take into account CUBIC [131]. While some packet-level models have been proposed recently for CUBIC (see the work from Bao *et al.* [47], Poojary and Sharma [200] or Ledesma Goyzueta and Chen [169]), numerical evaluation of those models did not result in satisfactory results due to limitations of the models, namely the initial assumptions made by the authors for the packet loss are not realistic.

TCP variant	Packet-level analytical models
Tahoe [148]	Sikdar *et al.* [223]
Reno [149]	Mathis *et al.* [185], Padhye *et al.* [195], Sikdar *et al.* [223]
NewReno [135]	Parvez *et al.* [198]
Vegas [71]	Samios and Vernon [211]
CUBIC [131]	Bao *et al.* [47], Poojary and Sharma [200], Ledesma Goyzueta and Chen [169]

Tab. 7.1: Packet-level analytical models for the bandwidth of TCP in the literature

Axiom 7.6. *In case of a network without loss ($e2e_p(\mathcal{S}) = 0$), the long-term average bandwidth of a TCP flow is limited by:*

$$\rho_{TCP}(\mathcal{S}) = \frac{MSS \cdot W_{max}}{RTT(\mathcal{S}, MSS, M_{ACK})} \tag{7.7}$$

with MSS the maximum segment size, W_{max} the maximum windows size (in number of packets) and M_{ACK} size of a TCP ACK packet.

We note that we already use the size of an ACK packet for the RTT in Equation (7.7) in order to have a better accuracy of the model.

In case of packet loss, we use the bandwidth model developed in [195], also known as the PFTK formula which models the bandwidth of the TCP Reno protocol. We use here the approximated version of the PFTK formula:

$$f_{PFTK}(RTT,p) = \frac{MSS}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \tag{7.8}$$

with p the drop probability, T_0 the sender timeout delay, and b the number of packets that are acknowledged by a received ACK.

For a simpler model of TCP Reno, the so-called square-root formula (or SQRT formula) presented in [185] may also be used:

$$f_{SQRT}(RTT,p) = \frac{MSS \cdot K}{RTT\sqrt{p}} \tag{7.9}$$

with K a constant depending on the type of loss experienced by the TCP flow and the acknowledgment strategy (delayed vs. non-delayed) used at the receiver. A good value of K is 1.31 in the case where each packet is acknowledged (equivalent to $b = 1$ in the PFTK formula) and under random loss as presented by Mathis *et al.* in [185, Table 1].

Using those formulas, the following axiom can be defined:

Axiom 7.7. *The long-term average bandwidth of a TCP flow is defined as:*

$$\rho_{TCP}(\mathcal{S}) = \begin{cases} \frac{MSS \cdot W_{max}}{RTT(\mathcal{S}, MSS, M_{ACK})} & \text{if } e2e_p(\mathcal{S}) = 0 \\ \min\left[f_{PFTK}\left(RTT(\mathcal{S}, MSS, M_{ACK}), e2e_p(\mathcal{S})\right), \frac{MSS \cdot W_{max}}{RTT(\mathcal{S}, MSS, M_{ACK})}\right] & \text{otherwise} \end{cases} \tag{7.10}$$

with MSS the maximum segment size, M_{ACK} the size of a TCP ACK packet, $RTT(\ldots)$ defined in Axiom 7.5, and $e2e_p(\ldots)$ in Axiom 7.1.

We illustrate the bandwidth of TCP as a function of RTT and drop probability as presented in the Equations (7.7) and (7.8) in Figure 7.2.

7.4.2 Improved TCP model with ACKs

As illustrated later with the evaluation of topologies with cross-traffic in Section 7.8, the model presented before does not take into account the impact of cross-traffic on the bandwidth of a flow which can lead to significant errors. This counterintuitive behavior comes from the fact that we modeled the TCP data flow as unidirectional, where a real TCP flow has acknowledgments (ACK) which can be affected by cross-traffic.

In this improved model, we consider that a TCP connection is constituted of two flows: the TCP data flow and the TCP ACK flow. We consider that the raw bandwidth of the ACK

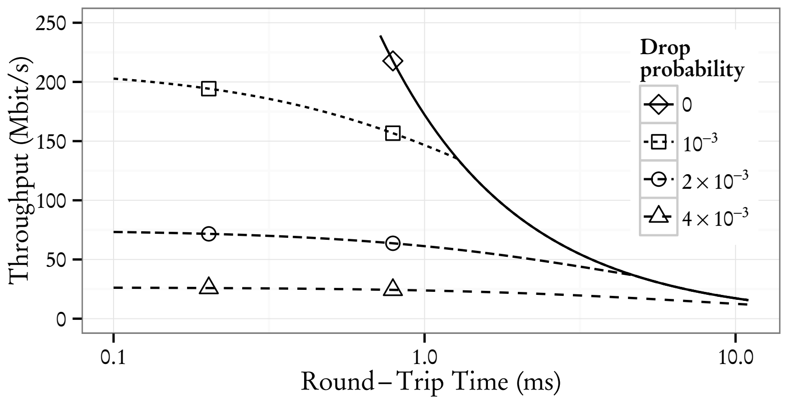


Fig. 7.2: Maximum bandwidth of TCP based on the approximate PFTK model, with $MSS = 1518\,\mathrm{B}$, $W_{max} = 14$, $T_0 = 1\,\mathrm{s}$, $b = 2$

flow corresponds to a certain fraction ϵ of the bandwidth of the Data flow: $\epsilon \cdot \rho_{data}$. We derive ϵ from the ratio of frames sizes between an ACK packet and a data packet, as well as b the number of packets that are acknowledged by a received ACK. For the numerical results presented later in Section 7.8 we choose:

$$\epsilon = \frac{M_{ACK}}{MSS \cdot b} = \frac{84\,\mathrm{B}}{1538\,\mathrm{B} \cdot b} \; (\approx 5 \times 10^{-2}, \text{ with } b = 1). \tag{7.11}$$

Axiom 7.8. *In order to account for cross-traffic, the bandwidth of the ACK flow ρ_{ACK} and the TCP Data flow ρ_{data} are constrained by the following set of equations:*

$$\rho_{data}\left(S_{data}\right) \le \rho_{TCP}\left(S_{data}\right) \tag{7.12}$$

$$\rho_{ACK}\left(S_{ACK}\right) \le \rho_{TCP}\left(S_{ACK}\right) \tag{7.13}$$

$$\rho_{ACK}\left(S_{ACK}\right) = \rho_{data}\left(S_{data}\right) \cdot \epsilon \tag{7.14}$$

with S_{data} the path of the data packets, S_{ACK} the path of the ACK packets, and $\rho_{TCP}(S)$ the application of Equation (7.10) *defined in Axiom 7.7.*

With this set of equations, we specify the dependencies between the bandwidth of the TCP data flow and the TCP ACK flow. With Equation (7.12) we constrain the bandwidth of the TCP data flow by the TCP bandwidth model on the path of the data packets $\rho_{TCP}(S_{data})$. Similarly, with Equation (7.13) we constrain the bandwidth of the TCP ACK flow by the TCP bandwidth model on the path of the ACK packets $\rho_{TCP}(S_{ACK})$. We take into account with this equation the effects of other flows on the path of the ACK packets (S_{ACK}) which corresponds to the cross-traffic, as well as asymmetric bandwidth. Finally, we establish the relation between the TCP data and TCP ACK bandwidth with Equation (7.14).

When the ACK flow is affected by cross-traffic and has a reduced bandwidth due to Equation (7.13), it has a direct impact on the bandwidth of the data flow using Equation (7.14).

7.4.3 Constant bitrate streaming flow model

Axiom 7.9. *For a flow f with constant bitrate (CBR) b without feedback or bandwidth adaptation, the bandwidth model can be expressed as:*

$$\rho_{CBR}(S) = b \tag{7.15}$$

This model is used for representing multimedia streaming flows based on UDP. As we model such flows with no feedback loop, the bandwidth of the flow is simply a constant value independent of the path.

7.5 Server model

Regarding our framework, a server corresponds to a queue in the network. Queues can be directly connected to an Ethernet physical interface or be regulated by a scheduler. Our model is able to support different types of scheduling algorithms. In this chapter, we describe the following elements constituting a server:

- Drop tail First-In-First-Out (FIFO) queue;
- Drop tail FIFO queue with traffic shaping;
- Random Early Detection (RED) queue;
- Strict Priority Queuing (SPQ) scheduling, as presented in Section 3.3.1;
- Approximations of Generalized Processor Sharing (GPS) scheduling and its packetized versions, such as the algorithms presented in Section 3.3.2;
- Hierarchical scheduler based of SPQ and GPS, as illustrated in Figure 7.3 and also evaluated earlier in Chapter 4.

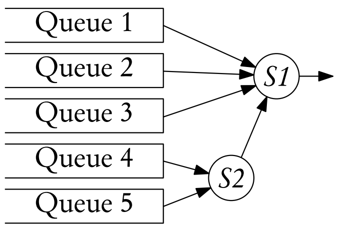


Fig. 7.3: Example of hierarchical scheduling with two schedulers *S1* and *S2*

Although we restrict this study to the aforementioned elements, other queue models might be used such as models based on queuing theory, like the $M/M/1/K$ queue used by Gibbens *et al.* in [126].

As defined earlier, a server is parameterized by C its maximum output bandwidth, D its additional delay, $F = \{f_n\}$ the set of traversing flows, Q its queue size specified by the function $H^Q(F)$, and p its drop probability specified by the function $H_k^p(F)$ depending on a set of flows F. The purpose of this section is to define the queue size function H^Q and drop probability function H^p of the queues. We consider that D the additional delay used for modeling propagation and processing delay is a constant value. More advanced models may define D as a function of the packet size or the usage of the server.

Schedulers regulate queues by allocating a specific bandwidth to the queues according to their own available bandwidth, noted here $C_{scheduler}$. To increase the accuracy of our model, the scheduler model should also adjust the additional delay due to the non-preemptive property of Ethernet, but we ignore it in the context of this chapter since we are interested at the flow layer.

7.5.1 Drop-tail First-In-First-Out queue

With a drop-tail FIFO queue, packets are served in their order of arrival. When the queue has no more space available for storing arriving packets, packets are simply dropped.

Previous research based the modeling of a queue on queuing theory, such as the work presented by Gibbens *et al.* in [126] or Ayesta *et al.* in [37] used a $M/M/1/K$ queue and the assumption that TCP packets arrive following a Poisson process. We propose to use here a simpler model which does not make any assumption on the input traffic.

The bandwidth available to the queue is noted C_Q.

Packet drop function $H^p(F)$

We consider here that the queue drop packets as soon as the incoming bandwidth is superior to the allowed output bandwidth.

Axiom 7.10. *The packet drop function of a drop-tail FIFO queue is expressed as followed:*

$$H^p(F) = \frac{\left[B^{inp} - C_Q\right]^+}{B^{inp}} \tag{7.16}$$

with $[x]^+ = x$ *if* $x \geq 0$, *and* 0 *otherwise.*

Equation (7.16) guarantees that $B^{out} \leq C_Q$ as defined in Axiom 7.3.

Queue size function $H^q(F)$

We model the queue size as follows:

Axiom 7.11. *The queue size function of a drop-tail FIFO queue is expressed as followed:*

$$H^q(F) = \begin{cases} M_Q & \textit{if } B^{inp} > C_Q \\ \max\{q|B^{out}(q) = \max(B^{out})\} & \textit{otherwise} \end{cases} \tag{7.17}$$

The queue is considered to be full (and equal to the maximum buffer size M_Q) when the incoming bandwidth is superior to the allowed output bandwidth. This is model by the first case of Equation (7.17).

When the queue is not full, the queue size will depend on how many packets may be transferred by the flows. As presented in Section 7.4.1 and Figure 7.2, TCP is able to fully utilize a link up to a certain limit of the round-trip time, or in other words by the buffer size of the different queues traversed by the flow. This means that the bandwidth of TCP, and hence B^{out}, is a function of the queue size. The queue size corresponds then to the maximum number of bits that has no impact on the bandwidth of the flows going through the queue. This is model by the second case of Equation (7.17).

7.5.2 Random Early Detection queue

Note This section is an extension of the original publication [121]. Due to its wide use in similar frameworks, we propose to include a model of the Random Early Detection queue for completeness purpose.

One queue type which is often used in the literature about flow-level modeling, such as for instance the work from Firoiu *et al.* in [111], is the Random Early Detection (RED) queue proposed by Floyd and Jacobson in [115]. The goal of this queue is to prevent tail dropping and TCP global synchronization by randomly dropping packets before the queue is completely full. This random packet dropping is defined with the following probability:

$$p = \begin{cases} 0 & \text{if average queue size} < min_{th} \\ max_p \cdot \frac{\text{average queue size} - min_{th}}{max_{th} - min_{th}} & \text{if } min_{th} \leq \text{average queue size} < max_{th} \\ 1 & \text{otherwise} \end{cases} \tag{7.18}$$

where min_{th} and max_{th} define the minimum and maximum threshold in term of queue size where the random packet dropping will occur, and max_p the maximum drop probability. Equation (7.18) is illustrated in Figure 7.4.

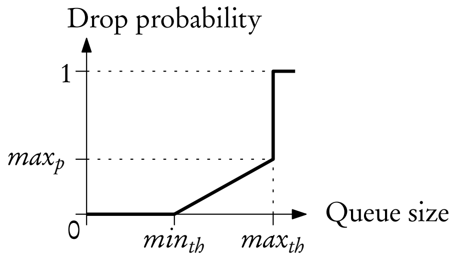


Fig. 7.4: Drop probability as a function of the queue size in RED queue

This queue is often used in flow-level modeling thanks to the following practical property: given that the flow bandwidth is a function of the drop probability of the path, the goal is to find p the value of the drop probability such that the flow bandwidth is not superior to the offered bandwidth, as defined earlier in Equation (7.4). Having found p, one can then infer the queue size by inverting Equation (7.18). While this is a clever application of the RED queue principle for finding the queue size, the drawback of this process is that it leads to difficulties when evaluating flows which are limited by the RTT (and hence queue size), as shown earlier in Equation (7.7). Indeed, when the queue size of the RED queue has to be lower than min_{th} for those particular RTT-limited flows, the process of inferring the queue size from the drop probability will not work.

Hence, we define here another formulation of a RED queue. For the queue size function $H^q(F)$, we reuse Axiom 7.11. The packet drop function $H^p(F)$ is then a reformulation of Equation (7.18) using the notation introduced earlier in Section 7.3.2.

Axiom 7.12. *The packet drop function of a RED queue is expressed as followed:*

$$H^p(F) = \begin{cases} 0 & \textit{if } H^q(F) < min_{th} \\ max_p \cdot \frac{H^q(F) - min_{th}}{max_{th} - min_{th}} & \textit{if } min_{th} \leq H^q(F) < max_{th} \\ \max\left(1, \frac{\left[B^{inp} - C_Q\right]^+}{B^{inp}}\right) & \textit{otherwise} \end{cases} \quad (7.19)$$

7.5.3 Queue with rate limiter or shaper

Some queues may use a shaping function, namely restrict the allocated bandwidth of queue although more bandwidth is available. An example of such function is the IEEE 802.1Qav studied earlier in Chapter 3.

In this case, the model we use is the same as the FIFO drop tail queue, but we modify the parameter C_Q of the server to allow a lower bandwidth than the available link bandwidth ($C_Q < C_{scheduler}$).

7.5.4 Strict Priority Queuing

Strict Priority Queuing (SPQ) is a scheduling algorithm where each queue is assigned a priority. The algorithm works as follows: all queues are polled in their priority order, until a non-empty queue is found and served. This process is restarted each time a packet needs to be dequeued.

This means that a queue served by SPQ can use the bandwidth that was unused by the queues of higher priorities.

Axiom 7.13. *When the SPQ scheduler has an available bandwidth of $C_{scheduler}$, each server s_k served by SPQ (queue or other scheduler), with k from 0 (highest priority) to N_q (lowest priority), has the following available bandwidth:*

$$C_k = \begin{cases} C_{scheduler} & \textit{if } k = 0 \\ \left[C_{scheduler} - \sum\limits_{i<k} B_i^{out}\right]^+ & \textit{otherwise} \end{cases} \quad (7.20)$$

with $[x]^+ = x$ if $x \geq 0$, and 0 otherwise.

We describe in Equation (7.20) that the queue with the highest priority ($k = 0$) has access to the all the available bandwidth ($C_{scheduler}$), while the rest of the queues have access to the bandwidth that is unused by the queues of higher priority.

7.5.5 Generalized Processor Sharing

With Generalized Processor Sharing (GPS), the scheduler distributes its available bandwidth $C_{scheduler}$ fairly among a set of queues $\boldsymbol{Q}$. Each queue q has a weight w_q and will receive the bandwidth:

$$C_q = \frac{w_q}{\sum\limits_{j \in \boldsymbol{Q}} w_j} \cdot C_{scheduler} \quad (7.21)$$

in case all queues are able to fully their allocated bandwidth ($B_q^{out} \geq C_q, \forall q \in Q$). When a queue uses less than its allocated bandwidth ($B_q^{out} < C_q$), the remaining bandwidth is redistributed to the other queues, according to their respective weights. This process of bandwidth sharing is described in Axiom 7.14.

This model corresponds to a simplification of Weighted Fair Queuing (WFQ) [197], Worst-Case Fair Weighted Fair Queuing (WF^{2}Q) [58], Deficit Round Robin (DRR) [222] or similar packet scheduling algorithm with proportional fairness with regards to the bandwidth. Those algorithms were presented earlier in Section 3.3.2.

Axiom 7.14. *In order to compute the allocated bandwidth of each queue, we use the following iterative process. We use the index n to mark the iteration step. We define $\mathcal{R}^n$ as the remaining unused bandwidth, C_q^n the bandwidth allocated to queue q, and Q^n as the set of queues using more than their currently allocated bandwidth C_q^n. We define the following initial values:*

$$C_q^{n=0} = 0, \forall q \in Q^{n=0} \tag{7.22}$$

$$Q^{n=0} = \left\{ q \middle| 0 \leq q \leq N_q \; and \; F_q \neq \varnothing \right\} \tag{7.23}$$

$$\mathcal{R}^{n=0} = C_{scheduler} \tag{7.24}$$

$Q^{n=0}$ corresponds to all queues with traversing flows as the initially allocated bandwidth is 0. We run the following iterative process until $Q^n = \varnothing$ or $\mathcal{R}^n = 0$:

$$C_q^{n+1} = \begin{cases} C_q^n + \frac{w_q}{\sum\limits_{i \in Q^n} w_i} \cdot \mathcal{R}^n & if \; q \in Q^n \\ B_q^{out} & otherwise \end{cases} \tag{7.25}$$

$$Q^{n+1} = \left\{ q : B_q^{out} \geq \frac{w_q}{\sum\limits_{i \in Q^n} w_i} \cdot \mathcal{R}^n \right\} \tag{7.26}$$

$$\mathcal{R}^{n+1} = \sum_{q \in Q^n} \left[\frac{w_q}{\sum\limits_{i \in Q^n} w_i} \cdot \mathcal{R}^n - B_q^{out} \right]^+ \tag{7.27}$$

We define in Axiom 7.14 an iterative process. In the first iteration of the process ($n = 1$), we allocate the total bandwidth of the scheduler $C_{scheduler}$ to the non-empty queues following their respective weights. At each step of the iteration, we then determine how much of the bandwidth is unused with $\mathcal{R}^n$. We allocate this bandwidth to the set of queues Q^n which are using more than their allocated bandwidth according to their respective weights. We iterate the process until all the bandwidth is used ($\mathcal{R}^n = 0$) or there are no more queues able to use the unused bandwidth ($Q^n = \varnothing$).

Note that the process described in Axiom 7.14 is similar to the so-called *water filling* algorithm, also sometimes referred as *progressive filling* [61]. Other alternatives used in the so-called max-min allocation as presented for instance in the work from Nace and Pióro in [194] may also be adapted.

7.5.6 Hierarchical scheduling

As noted earlier, our model for a scheduling algorithm redistributes its available bandwidth $C_{scheduler}$ to the queues according to the output bandwidth B^{out} of the queues. Hence, when using hierarchical scheduling as illustrated by Figure 7.3, a scheduler acts on the bandwidth of a sub-scheduler in the similar way it acts on a queue.

In the hierarchical scheduler presented in Figure 7.3, $S1$ will allocate some bandwidth to the scheduler $S2$ in the same way as it allocates it to Queue 1 to 3. Then scheduler $S2$ will redistribute this bandwidth to Queue 4 and Queue 5.

7.6 Solving the model

As presented in Figure 7.5 and based on the different models previously described, we have the following relation: flows react on network changes by adjusting their packet sending rate, while the network reacts on flows by queuing and dropping packets.

The performance evaluation of the system is equivalent to finding the values Q_k, p_k and r_i of the different servers and flows which lead to an equilibrium or fixed point of the system described by the different axioms previously enumerated.

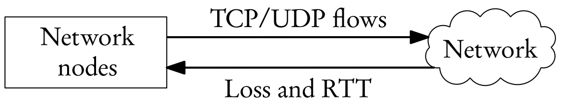


Fig. 7.5: Congestion control

Algorithm 7.1 describes the procedure to find the equilibrium of the system. We distinguish two parts in the algorithm. The first part (lines 1 to 5) initializes the variables Q_k, p_k and r_i to 0. The second part (lines 6 to 13) evaluates the functions until the fixed point is reached. While various methods can be used to check this condition, we compare here the absolute difference between current iteration and the last one, and check if it is below a given threshold. In mathematical term, this can be translated as:

$$\sum_i |r_i(\text{current iteration}) - r_i(\text{previous iteration})| \leq \epsilon$$

While a proof of existence of an equilibrium point was already given by Altman *et al.* in [33] for TCP flows, we define a safeguard function in order to avoid an infinite loop (line 12) in case an equilibrium cannot be reached, as we do not necessarily limit our framework to TCP flows using TCP Reno as in [33]. The simplest function to achieve this is to limit the number of iteration of the loop (line 6 to 13). An alternative way is to look at the evolution of Q_k, p_k and r_i, and determine if an equilibrium is reachable by looking at the convergence rate of the variables.

7.7 PETFEN: a tool for numerical evaluation of flow-level modeling

Note This section is based on our previous work [124] published in *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, 2014. Section 7.7.3 has been extended in order to give a more thorough overview of the capabilities of the tool.

Algorithm 7.1 Fixed-point evaluation algorithm for Axioms 7.1 to 7.5

```
Require: Set of servers $\mathcal{S}$, set of flows $\mathcal{F}$
 1: for all $k \in \{0, \ldots, |\mathcal{S}|\}$ do
 2:     $Q_k \leftarrow 0$
 3:     $p_k \leftarrow 0$
 4: for all $i \in \{0, \ldots, |\mathcal{F}|\}$ do
 5:     $r_i \leftarrow 0$

 6: while equilibrium not reached do
 7:     for all $k \in \{0, \ldots, |\mathcal{S}|\}$ do
 8:         $Q_k \leftarrow H_k^Q(\mathcal{F}_k)$
 9:         $p_k \leftarrow H_k^p(\mathcal{F}_k)$
10:     for all $i \in \{0, \ldots, |\mathcal{F}|\}$ do
11:         $r_i \leftarrow \rho_i(\mathcal{S}_i)$
12:     SAFEGUARD()                                  ▷ Function to avoid infinite loop
13: end while
```

While the literature on flow-level network models is extensive, as show in Section 7.2, there is still a lack of tools for numerical evaluations on user provided topologies and flows. We present in this section PETFEN – Performance Evaluation Tool for Flow-level network modeling of Ethernet Networks – a tool implementing the analytical results and algorithms presented in Sections 7.3 to 7.6.

We propose here a tool with the following goals:

1. Provide a convenient way to describe Ethernet topologies and flows which is human writable and readable, while enabling researchers to perform parameter studies in the most flexible way;

2. Evaluate those topologies with the flow-level network mathematical modeling framework described earlier in Sections 7.3 to 7.6;

3. Provide a convenient way to compare numerical evaluations of mathematical models with other tools, such as discrete event simulators or emulators.

PETFEN was programmed in Java, and uses a special Lisp-based domain specific language to describe and generate network topologies and flows procedurally.

7.7.1 Related tools

We present here the few tools using similar flow-level network models. To the best of our knowledge, there are no other available tool using similar mathematical models for modeling a network and its TCP flows.

OptorSim [56] is a tool proposed by Bell *et al.*, designed to study data replication on grid networks, where communications are modeled using a flawed fair bandwidth sharing. As noted by the authors themselves, and documented in the `BUGS` file of the OptorSim distribution, the implemented bandwidth sharing give too pessimistic results on networks with more than one bottleneck. The example given by the authors is that if there is a network with two bottlenecks with available bandwidth C_1 and C_2, with one flow over the two bottleneck and a second flow over only over the second one, the flows will receive respectively a bandwidth of $\min(C_1, C_2/2)$ and $C_2/2$, regardless if $C_2/2 > C_1$ or not.

SimGrid [80] is a more general tool proposed by Casanova *et al.* for the study of grid networks. Various mathematical models of flow-level networks can be used in SimGrid, the default one being a modified max-min bandwidth sharing fitted to grid networks as presented by Velho and Legrand in [233]. Note that this supports the effect of cross-traffic on TCP. This approach is analogous to the network utility maximization approach presented earlier in Section 6.3.2, which treats the bandwidth sharing as a distributed optimization problem. Results of SimGrid for the evaluation of TCP flows were shown to be accurate in [233]. Like PETFEN, it also includes modules for interacting with external tools (ns-3 and GTNetS).

fs [226] was proposed more recently by Sommers *et al.*, with the goal of generating representative flow export records of various applications with a focus on scalability. While using a similar approach than ours for the evaluation of the bandwidth of the TCP flows, *fs* requires the user to predefine the packet drop probability of flow, which in case of Ethernet networks is not straightforward.

While those tools propose some interesting features, we found that they do not meet all of our requirements for the study of industrial Ethernet networks, namely:

1. No tool supports packet-level scheduling.
2. Queue sizes are ignored, which results in imprecision when evaluating Ethernet topologies with low latencies as shown later in Section 7.8.6 where we compare the results of PETFEN with SimGrid.
3. Parameter studies are often not flexible enough and require computer generated configuration files to be really efficient.
4. Those tools can be viewed as simulators, meaning that the results they produce are based on traces of predefined or pseudo-random events, while we aim at having mean performances.

Table 7.2 summarizes the comparison between the various tools and our own tool PETFEN.

7.7.2 PETFEN general architecture

PETFEN is a Java based tool for the numerical evaluation of networks, using the mathematical tools and axioms presented in Sections 7.3 to 7.6. Its architecture can be divided into four parts, as presented in Figure 7.6. Classes for describing an Ethernet topology and its attributes form its base: computer, network card, queue and scheduler. A topology corresponds to a graph of those classes. Additionally, there are classes for TCP and UDP flows, which use this graph for the routing. A topology can be created either directly by using the available Java API, or via a domain specific language as detailed later in Section 7.7.3.

This graph of objects representing the various elements of the network is used by the flow-level network modeling toolbox, which is our implementation in pure Java of the results presented in Sections 7.3 to 7.6.

Tool	Mathematical model	Results	Parameter study
OptorSim [56]	Fair sharing with bug	Traces	No flexibility
SimGrid [80]	B:[184] (max-min), D:[233]	Traces	Some flexibility
fs [226]	B:[77, 185]	Traces	No flexibility
PETFEN	B:[77, 195], D:[121, 123]	Mean performances	High flexibility

Tool	Cross traffic	Scheduling	Interaction with other tools
OptorSim [56]	✗	✗	✗
SimGrid [80]	✓	✗	GTNetS [206], ns3 [16]
fs [226]	✗	✗	✗
PETFEN	✓	✓	OMNeT++ [17], ns2 [15], mininet [165]

Tab. 7.2: Comparison of existing tool using flow-level network modeling. Abbreviations used: *"B:"* = Base model, *"D:"* = Detailed model.

The graph is also used by various modules for interacting with external tools. Namely we developed modules for ns2 [15], OMNeT++/INET [17, 14], as well as experimental modules for mininet [165] and SimGrid [80]. The modules handle the following tasks:

- Export of the topology and configuration of the flows sources and destinations for the external tool;
- Execution of the external tool with the specified topology;
- Import of the results of the tool for easier comparison with the results of the mathematical toolbox.

This enables us to easily validate and compare the numerical results from the mathematical toolbox against other well established methods and tools.

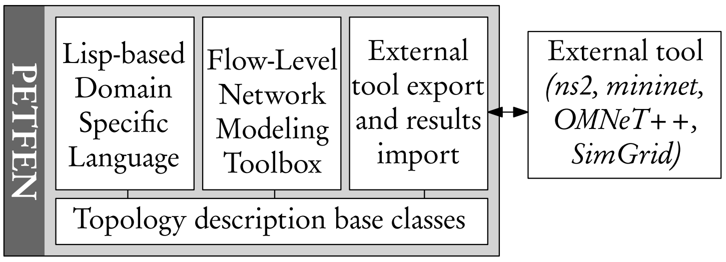

PETFEN
Lisp-based Domain Specific Language
Flow-Level Network Modeling Toolbox
External tool export and results import
Topology description base classes
External tool (ns2, mininet, OMNeT++, SimGrid)

Fig. 7.6: General architecture of PETFEN

7.7.3 PETFEN domain specific language and scripting

In order to describe the studied topologies, tool creators generally use one of the following solutions:

A Graphical User Interface (GUI) where users can easily input their topologies using predefined widgets;

A markup language where users need to describe topologies in a text file following a prescribed markup (ex: XML, JSON, CSV);

A dedicated Application Programming Interface (API) where users need to interface the tool in a prescribed programming language;

A Domain Specific Language (DSL) where users need to describe their topologies in a text file, but with additional functionalities compared to a simple markup language.

One important aspect of a tool is to enable so-called parameters studies or *what-if* studies, meaning that having a base network, researchers are generally interested into how it behaves when parameters change (ex: number of users using the network, or latency of a link). While the first two solutions can offer such capability, they generally are lacking in term of flexibility. This is why we focused on the two last solutions for PETFEN.

As PETFEN is written in Java, we can take advantage of the Java Virtual Machine (JVM) and its interoperability with a variety of languages for defining a DSL. We chose to use Clojure [139] for this task, a functional Lisp dialect running on the JVM. Being a Lisp dialect, it enables us to have:

- A compact programing and markup language which is easily understood and used, as shown in Listing 7.2;
- The ability to define and use functions and macros, giving us great flexibility when doing parameters studies, as shown in Listing 7.6.

Basic commands

PETFEN DSL is based on a few Clojure functions for describing a network topology and its flows:

Listing 7.2: Basic functions used in PETFEN DSL

```
Definition of a node in the network, representing a switch, router or computer.
Options such as internal latency or packet scheduling policy can be specified.
(node <name> <opt.>)

Definition of a link between two nodes.
Options such as link latency or packet drop probability can be specified.
(link <node 1> <node 2> <speed> <opt.>)

Definition of a TCP flow between a source and destination.
Options such as TCP version, message size can be specified.
(tcpflow <name> <source> <dest.> <opt.>)

Definition of a UDP flow between a source and destination.
(udpflow <name> <source> <dest.> <opt.>)

Definition of a queue.
(queue <opt.>)

Definition of a scheduler.
(scheduler <type> <queue 1> ... <opt.>)
```

Thanks to Clojure's named keywords, options can be easily specified (noted *opt.* in Listing 7.2). For example, for defining a drop-tail FIFO queue with a maximum number of 50 packets and a weight of 2, we have:

Listing 7.3: Example of use of Clojure named keyworks

```
(queue :qtype DROPTAIL :K 50 :W 2)
```

Similarly, the hierarchical scheduler presented in Figure 7.3 can be described as:

Listing 7.4: Example of hierarchical scheduler in PETFEN DSL

```
(schedulingpolicy HierarchicalScheduler
  (scheduler SPQ
    (queue :prio 1)
    (queue :prio 2)
    (queue :prio 3)
    (scheduler WFQ
      (queue :prio 4 :W 2)
      (queue :prio 5 :W 1))))
```

We describe below a small example of a dumbbell topology, as illustrated in Figure 7.7.

Listing 7.5: Dumbbell topology with 2 hosts as illustrated in Figure 7.7

```
(node Cli1) (node Cli2)
(node Srv1) (node Srv2)
(node SW1) (node SW2)

(link SW1 SW2 (Mbps 100))
(link Cli1 SW1 (Mbps 100))
(link Cli2 SW1 (Mbps 100))
(link Srv1 SW2 (Mbps 100))
(link Srv2 SW2 (Mbps 100))

(tcpflow Flow1 Cli1 Srv1)
(tcpflow Flow2 Cli2 Srv2)
```

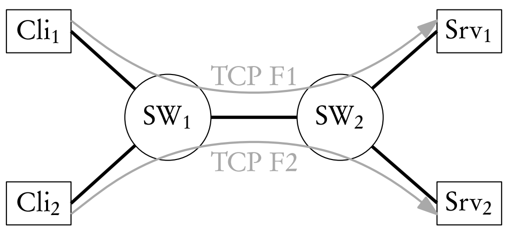


Fig. 7.7: Dumbbell topology without cross-traffic

Parameter studies and scripting

Thanks to the use of a full programing language for describing the topologies, parameter studies can be easily done with PETFEN. Topologies can be procedurally constructed using any standard Clojure function.

The example in Listing 7.6 describes the same dumbbell topology, but with a variable number of client/server pairs, as pictured in Figure 7.8. With lines 1 to 12, we first write a function which generates a topology according to N, the desired number of clients and servers. Lines 6 to 11 correspond to the loop generating the N clients, servers and flows. On line 12, we trigger the evaluation of the generated topology, either with mathematical models or via external tools. Then with lines 14 and 15, we evaluate this function against the desired values of N, here from 1 to 100.

Listing 7.6: Dumbbell topology with variable number of hosts as illustrated in Figure 7.8

```
 1: (defn generateTopology [N]
 2:   (newTopology (str "Topology N=" N))

 4:   (node SW1) (node SW2)
 5:   (link SW1 SW2 (Mbps 100))
 6:   (dotimes [i N]
 7:     (let [cli (node_ (str "Cli" i))
 8:           srv (node_ (str "Srv" i))]
 9:     (link cli SW1 (Mbps 100))
10:     (link srv SW2 (Mbps 100))
11:     (tcpflow_ (str "Flow" i) cli srv)))
12:   (performAnalysis))

14: (doseq [N (range 1 100)]
15:   (generateTopology N))
```

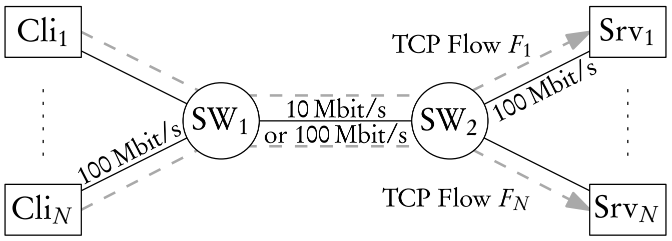


Fig. 7.8: Dumbbell topology

7.8 Numerical evaluation

We evaluate in this section different topologies with the framework presented in Sections 7.3 to 7.6 and compare its results with simulations. When not otherwise specified, we consider that links between nodes are full-duplex, using a 10 m Ethernet cable, with a propagation delay of 5×10^{-8} s, and a bandwidth of 100 Mbit/s. All elements of the network are considered to have no internal processing delay. Ethernet switches have an internal drop-tail queue with a default maximum number of 10 packets for each port. Computers are considered to have no queue and no scheduling element for the egress part of the Ethernet interface.

We use the results of the discrete event simulator OMNeT++ [17] and its framework INET [14] as a comparison for our model. The standard modules `StandardHost` and `EtherSwitch` from the INET framework were used for modeling computers and switches, and we configure the TCP stack to use the values noted earlier.

For the configuration of TCP, we use the default parameters used in OMNeT++, namely a maximum window size W of 14 packets, a maximum segment size of 1538 B (Ethernet frame size with preamble and interframe gap) and a timeout time T_0 of 1 s. b, the number of packets acknowledged by an ACK, is set to 1. As noted earlier, we use here TCP Reno.

To evaluate the difference between the flow-level network model and the results of the OMNeT++ simulation, we use the log-error of the throughput of flow f_i, as defined by Velho and Legrand in [233]:

$$LogErr(f_i) = \left|\log\left(r_i^{Model}\right) - \log\left(r_i^{Simulation}\right)\right| \tag{7.28}$$

7.8.1 Validation of the TCP model

In order to validate the behavior observed in Figure 7.2, we evaluate a simple topology where two PCs, *Cli* and *Srv*, are connected to the switch SW, as presented in Figure 7.9. We define the latency of packets going from SW to *Srv* as a parameter for this study.

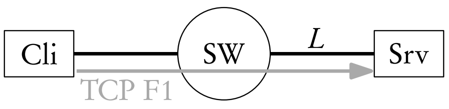


Fig. 7.9: First topology with a variable latency L between SW and *Srv*

The bandwidth of the TCP flow between *Cli* and *Srv* is presented in Figure 7.10. The log-error between the results of our model and the results of OMNeT++ suggests that the flow-model is indeed relevant regarding the influence of round-trip time.

We note on Figure 7.10 that the main cause for the drop in bandwidth for $L \geq 1.5$ ms is due to a limitation of the maximum window size. This behavior corresponds mainly to the case where the bandwidth is limited by Equation (7.7). Nevertheless, this evaluation validates our model of the size of a drop-tail queue defined in Equation (7.17).

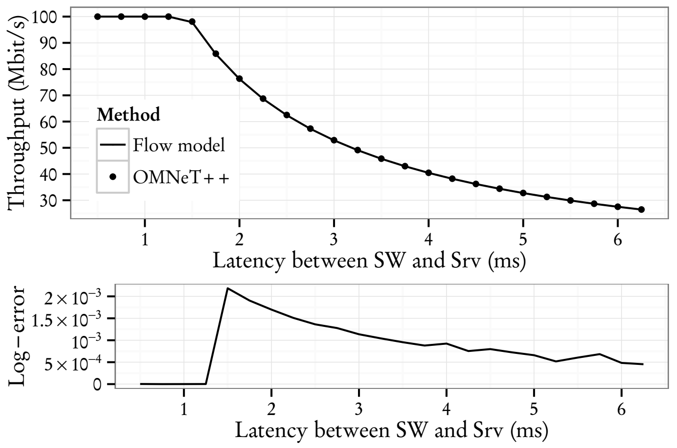


Fig. 7.10: Comparison between the flow model and the OMNeT++ results for the topology presented in Figure 7.9, with the bandwidth (above) and the log-error (below)

Note The remaining text of this section and the results presented in Figures 7.11 and 7.12 are extensions of the original publication [121]. Those additional results highlight the network conditions where the PFTK formula (hence also our model) looses accuracy as well as an evaluation of the sensitivity of the simulation's parameters.

We evaluated in Figure 7.10 the case of an Ethernet networks where packet loss mostly occurs due to queue overflow as the bit error rate of an Ethernet link is negligible (usually a value between 10^{-9} and 10^{-12} depending on the technology used). We evaluate a second case where we add artificial packet drops for packets going from *SW* to *Srv*. The packet drop follows a Bernoulli distribution, with packet error rates from 10^{-4} to 5×10^{-3}. We also evaluate here the influence of the maximum window size W (in packets), with $W = 14$ (default value of OMNeT++) and $W = 40$ (value near the maximal 65 535 B limit for TCP without the window scale option defined in RFC 1323 [147]). Simulations were performed for a duration of 300 s and repeated 5 times following a Monte-Carlo approach.

Results are presented in Figure 7.11. The model is accurate for small packet error rates (10^{-4}), while the gap with the simulation grows as the packet error rate increases. This is especially visible for the packet error rate of 5×10^{-3} where the error is larger than 100 %.

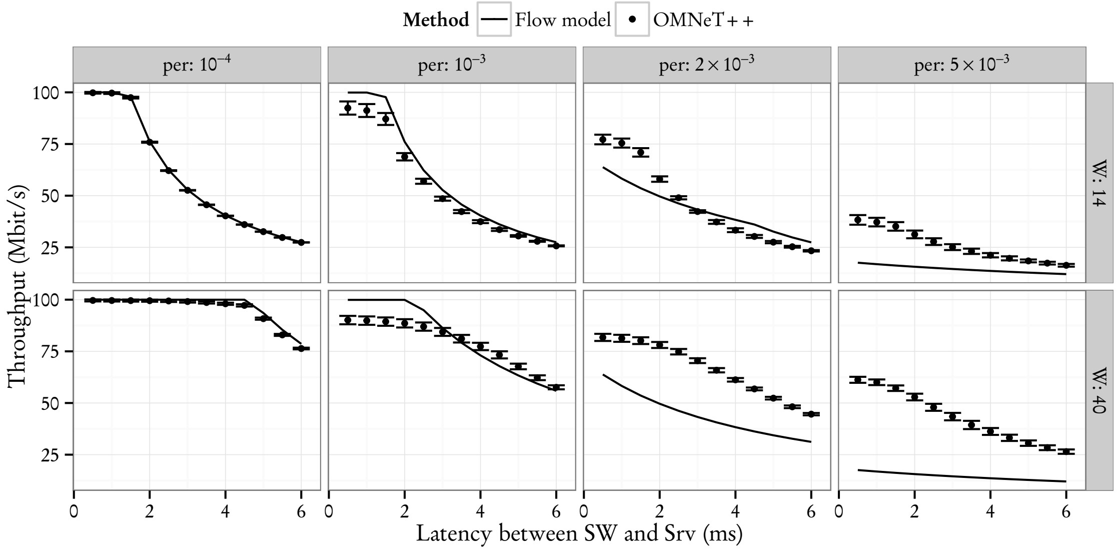
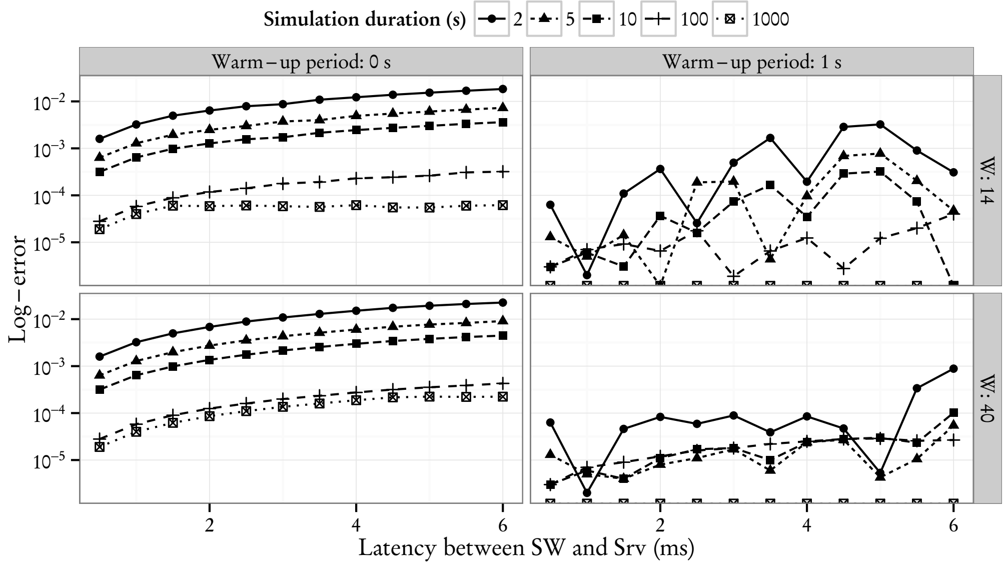


Fig. 7.11: Comparison between the flow model and the OMNeT++ results for the topology presented in Figure 7.9 with artificial packet drop

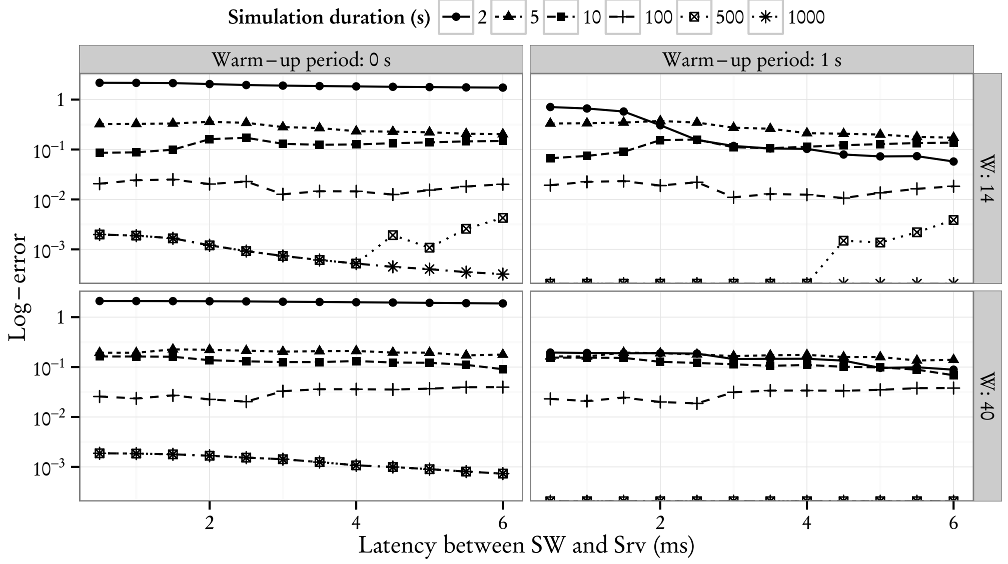
We noted in Section 7.4 that the bandwidth property in our flow-model represents long-term averages. In order to reflect this notion and produce accurate simulation results, one has to define two key parameters: the duration of a simulation and its warm-up period. The warm-up time corresponds to the initial time during which the activity of the simulation is not recorded and is generally used to not take into account an initial transient phase (the TCP slow-start phase in our case).

To assess the sensitivity of those parameters with the end result, we evaluated the topology presented in Figure 7.9 with different simulation durations and warm-up periods. The results are presented in Figure 7.12. The log-error corresponds to the comparison between the simulation and the “ground-thruth” simulation, which corresponds here to the simulation with the longest duration and warm-up times.

The benefit of using a warm-up period in order to remove the effects of the TCP slow-start phase is clear on Figure 7.12. We also note that by extending the simulation’s duration we generally reduce the log-error.



(a) Without artificial packet loss



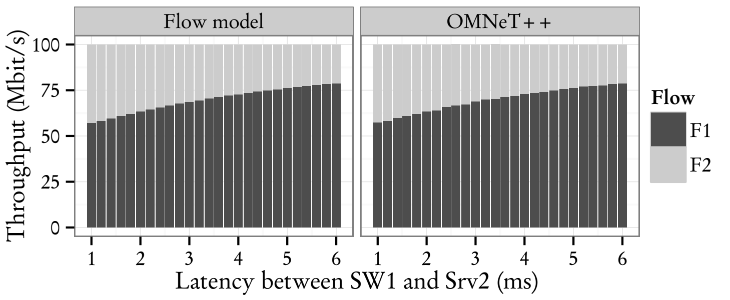
(b) With artificial packet error rate of 5×10^{-3}

Fig. 7.12: Comparison between different parameters of the simulation

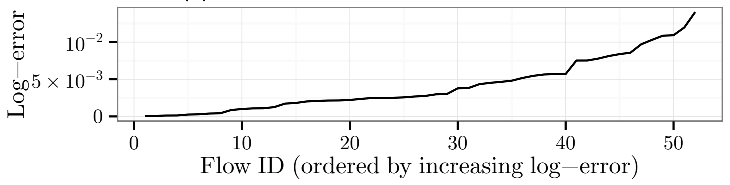
7.8.2 Dumbbell topology without cross-traffic

We study here the influence of asymmetrical latency on a dumbbell topology, as illustrated in Figure 7.7. All links have the same delay, except for packets going from *SW1* to *Srv2*, experiencing a delay between 1 ms to 6 ms. The maximum number of packets for the queues inside *SW1* and *SW2* is set to 30.

The individual bandwidth of each flow for this topology are presented in Figure 7.13. As expected, we do not see a fair sharing of the bandwidth between Flow *F1* and Flow *F2*, as it is known that TCP Reno favors flows with a lower round-trip delay time.



(a) Evaluation of the bandwidths



(b) Evaluation of the log-error

Fig. 7.13: Comparison between the flow model and the OMNeT++ results for the dumbbell topology (see Figure 7.7)

7.8.3 Dumbbell topology with cross-traffic

We study in this case the effect of cross-traffic on TCP flows. We use the same dumbbell topology as in Section 7.8.2, but we add TCP flow *F3* from node *Srv2* to node *Cli1*, as presented in Figure 7.14.

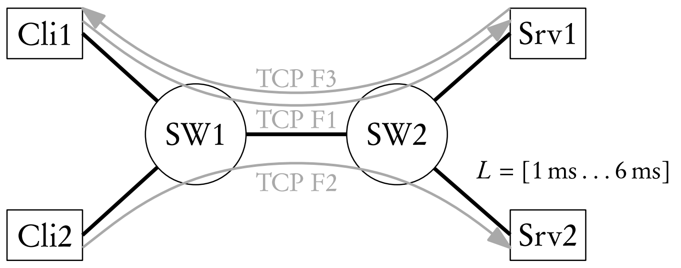


Fig. 7.14: Dumbbell topology with crosstraffic

We first present the results of this topology using the TCP model without the ACK flows in Figure 7.15. Results for flows *F1* and *F2* are comparable to the one presented in the previous topology. But for flow *F3*, we see that the results of the flow-level network model do not match the results from OMNeT++. Indeed, the effect of cross-traffic is visible here: *F3* is not able to fully use the bandwidth available between *Srv1* and *Cli1* although all links are full-duplex. The throughput is equal to only about half the available bandwidth, because the acknowledgments of *F3* are competing with the packets of *F1* and *F2*. As noted in Section 7.1, this phenomenon is well known in the literature, first explained by Zhang *et al.* as ACK compression in [252], and more recently by Heusse *et al.* as the principle of data pendulum in [137]. Techniques exist to overcome this problem such as in RFC 3449 [46], where a simple solution is to schedule the TCP ACK packets with a higher priority than the TCP data packets.

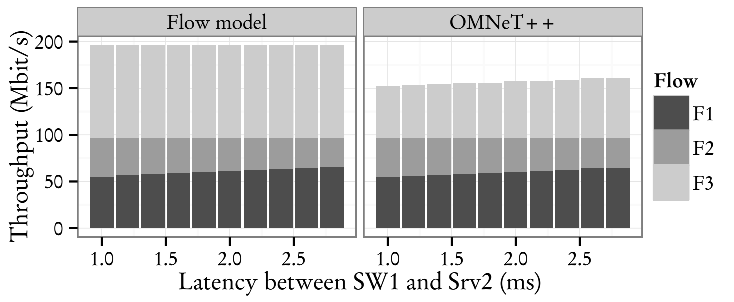


Fig. 7.15: Dumbbell topology with cross-traffic: throughput of the TCP flows without taking into account the TCP ACKs packets

As explained earlier, previous work on flow-level network model often neglect this problem by studying only topologies where there is no cross-traffic, and the models proposed will give a similar error as in Figure 7.15.

By using the improved TCP model presented in Section 7.4.2, we obtain the same behavior as in OMNeT++, as shown in Figure 7.16.

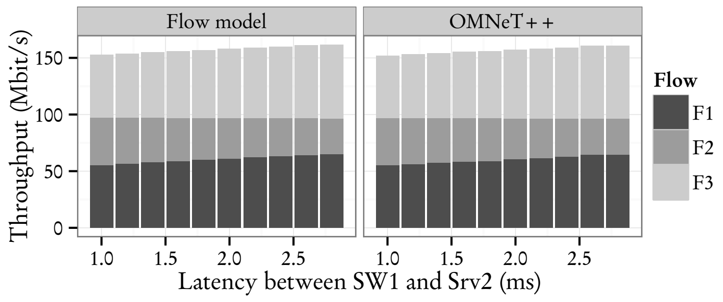
7.8.4 Topology with cross-traffic, WFQ scheduling and streaming traffic

We demonstrate here the ability of our framework to handle scheduling algorithms previously described as well as streaming traffic. We use the topology presented in Figure 7.17. The cross-traffic here is generated by flows *F3* and *F7*. The egress part of the switches uses Weighted Fair Queuing, with 3 priorities, from 0 to 2, with respective weights of 5, 1 and 2. We define a maximum queue size of 50 packets.

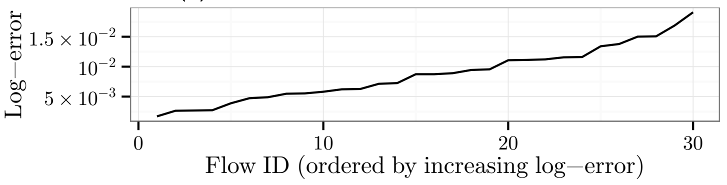
Results for this topology are presented in Figure 7.18, where we compare the results of the simple TCP model with the results of the improved TCP model. When the TCP ACKs are not taken into account large errors appear in the results. Our improved TCP model is indeed relevant compared to the OMNeT++ results.

7.8.5 Random tree topology

In order to evaluate our framework on other topologies, we used randomly generated trees following Algorithm 7.7. We used a tree topology as it guarantees a unique path between

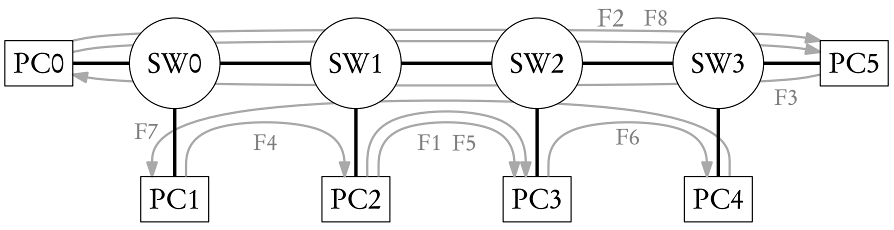


(a) Evaluation of the bandwidths



(b) Evaluation of the log-error

Fig. 7.16: Comparison between the flow model and the OMNeT++ results for the dumbbell topology with crosstraffic (see Figure 7.14) with the improved model taking into account TCP ACKs



Flow	Source	Dest	Flow type	Priority
F1	PC2	PC3	UDP rate 5 Mbit/s	1
F2	PC0	PC5	TCP	1
F3	PC5	PC0	TCP	1
F4	PC1	PC2	TCP	2
F5	PC2	PC3	TCP	2
F6	PC3	PC4	TCP	2
F7	PC4	PC1	TCP	0
F8	PC0	PC5	TCP	0

Fig. 7.17: Daisy chain topology with four switches and WFQ scheduling

two nodes of the topology, meaning that the path of flow is guaranteed to be the same in the model and in the simulation. The leaves of the tree correspond to computers, while the internal vertices correspond to switches. The algorithm generates only TCP flows, via the function TCP_FLOW(<SOURCE>, <DESTINATION>). Switches are considered to have infinite buffers.

We generated four random tree topologies using Algorithm 7.7 with parameters *maxDepth* = 4, *minLeaves* = 4, *maxLeaves* = 8, *minFlows* = 1, *maxFlows* = 1 and evaluated the log error

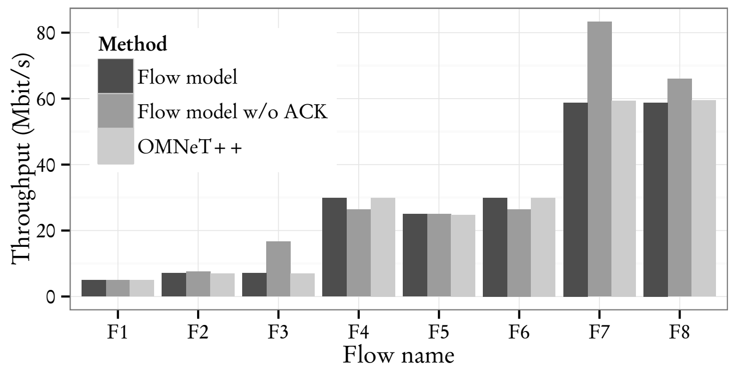


Fig. 7.18: Throughput of the flows for the daisy chain topology

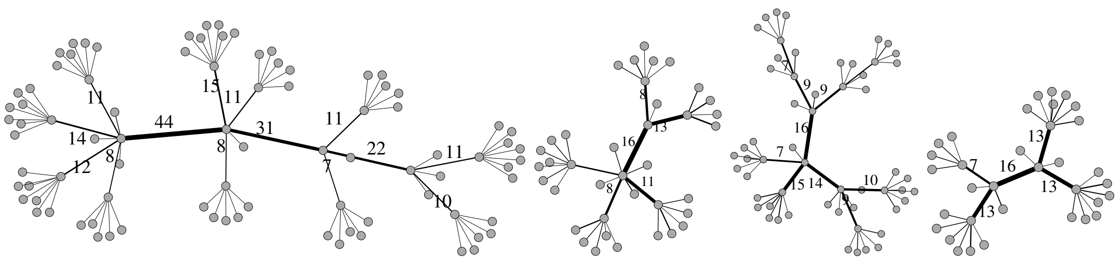
Algorithm 7.7 Random tree generation algorithm

```
Require: $maxDepth \geq 0$, $minLeaves \geq 0$, $maxLeaves \geq 0$, $minFlows \geq 0$, $maxFlows \geq 0$
 1: function GENERATETOPOLOGY
 2:     $root \leftarrow$ CREATENODE
 3:     $leaves \leftarrow$ GENERATELEAVES($root$, $maxDepth$)
 4:     for all $leaf$ in $leaves$ do
 5:         for RANDOMINT($minFlows$, $maxFlows$) do
 6:             TCP_FLOW($leaf$, RANDOM($leaves$))
 7:         end for
 8:     end for
 9: end function

10: function GENERATELEAVES($root$, $depth$)
11:     if $depth = 0$ then
12:         return $root$
13:     end if
14:     $leaves \leftarrow []$
15:     for RANDOMINT($minLeaves$, $maxLeaves$) do
16:         $node \leftarrow$ CREATENODE
17:         CREATELINK($node$, $root$, 100 Mbit/s)
18:         $d \leftarrow$ RANDOMINT($0$, $depth - 1$)
19:         $leaves \leftarrow$ [$leaves$, GENERATELEAVES($node$, $d$)]
20:     end for
21:     return $leaves$
22: end function
```

for the flow throughputs. The topologies correspond to the ones presented in Figure 7.19. For reproducibility purpose, the complete descriptions of those topologies can be found in Appendix B.

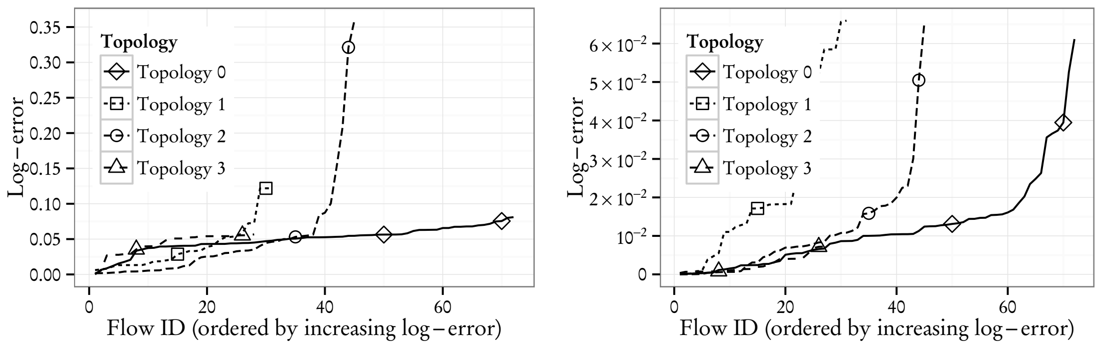
We first study the evaluation of the TCP model without acknowledgments as presented in Figure 7.20a. We see that the log-error reaches a maximum value of 0.37, which corresponds to a relative error of $\exp(0.37) - 1 = 44.8\,\%$. The four topologies are then evaluated with the improved TCP model including acknowledgments, and results are presented in Figure 7.20b.



(a) Topology 0 (b) Topology 1 (c) Topology 2 (d) Topology 3

Fig. 7.19: Example of four generated topologies with Algorithm 7.7 using the following parameters: *maxDepth* = 4, *minLeaves* = 4 and *maxLeaves* = 8. The width and the label of the edges represent the number of flows on the edge (for edges with more than 5 flows). The complete descriptions of those topologies is presented in Appendix B.

As expected, the accuracy of the model is improved, with a maximum log-error of 6.5×10^{-2} which corresponds to a relative error of $\exp(6.5 \times 10^{-2}) - 1 \approx 6.71\,\%$.



(a) Basic model without TCP ACKs (b) Extended model with TCP ACKs

Fig. 7.20: Log error of the flow throughput on four random tree topologies generated by Algorithm 7.7

7.8.6 Comparison with SimGrid

We saw in Section 7.7.1 that other related tools using similar mathematic models are available for evaluating the topologies presented earlier in this section. The goal in this subsection is to compare the numerical evaluations presented earlier, to the ones of other tool. We decided to focus here on SimGrid[1], as it is the only tool which also models the effect of cross-traffic on TCP, and hence the one which is the most likely to give the most accurate results out of the other tools.

Instead of the log-error used through Section 7.8, we use here the relative error, with

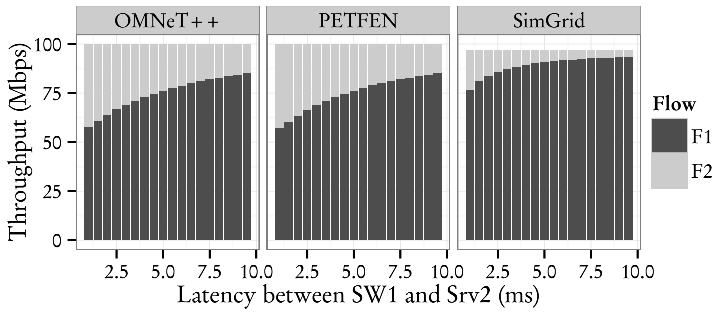
[1] This evaluation was performed with the C version of SimGrid 3.11.1 (2014-06-02 07:47)

OMNeT++ as reference, such that the error is defined as:

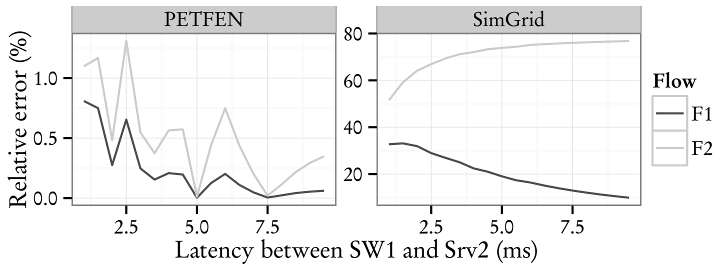
$$RelativeError^{tool}(f_i) = \left| \frac{r_i^{\text{OMNeT++}} - r_i^{tool}}{r_i^{\text{OMNeT++}}} \right| \tag{7.29}$$

with r_i^{tool} corresponds to the bandwidth of the measured flow using the tool *tool*.

We first look at the dumbbell topology presented in Figure 7.7, where the latency between SW$_2$ and Srv$_2$ varies between 1 ms and 10 ms. The results of the evaluation are presented in Figure 7.21. As expected, this topology is a good example to illustrate the unfairness of TCP regarding round-trip times. As queue sizes are not modeled in SimGrid, it gives larger relative error compared to the simulations, as queue sizes largely influence round-trip times in Ethernet. PETFEN, which models queue sizes, gives accurate results with a relative error between 0 % and 1.5 %.



(a) Evaluation of the bandwidths



(b) Evaluation of the log-error

Fig. 7.21: Bandwidth sharing between the two flows on the dumbbell topology presented in Figure 7.7, with relative error using OMNeT++ as a reference

We then proceed to evaluate the four randomly generated tree topologies described and illustrated earlier in Figure 7.19. The results are presented in Figure 7.22, where we look at the relative error between the analytical results and the OMNeT++ simulations. Those errors are comparable with the ones of the dumbbell topology. PETFEN gives results with a relative error below 6 % for all flows of the topologies. SimGrid gives error of up to 100 %, leading us to the conclusion that SimGrid is more adapted to topologies where queuing delays can be ignored.

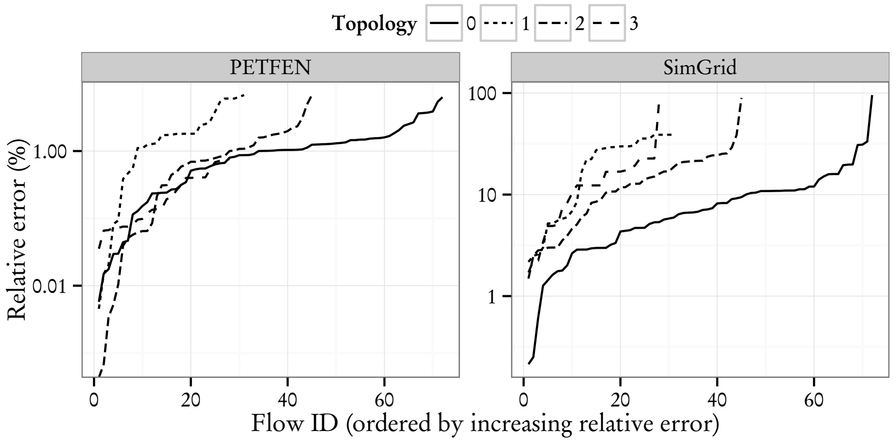


Fig. 7.22: Relative error between the tools and OMNeT++ on the tree topologies presented in Figure 7.19 and Appendix B

7.9 Conclusion on flow-level network modeling

We presented in this chapter a flow-level network framework for the performance evaluation of Ethernet topologies with long-lived TCP and UDP flows. We also introduced PETFEN – Performance Evaluation Tool for Flow-level network modeling of Ethernet Networks – a tool implementing the different algorithms and mathematical models introduced in this chapter.

This framework is based on two building blocks: servers for representing Ethernet interfaces and queues, and flows for communications between nodes of the topology. Our model for servers supports FIFO queues as well as more advanced queuing mechanisms, namely active queue management algorithms such as RED or packet scheduling algorithms such as priority based scheduling and fair-bandwidth sharing schedulers. The model presented in this chapter is for Ethernet flows supporting long-lived TCP connections as well as UDP multimedia streams.

The results of our framework were compared to the results of the discrete event simulator OMNeT++, as well as SimGrid, another tool using a flow-level based mathematical modeling. Different topologies where used in order to evaluate the accuracy of our model and our tool PETFEN. Our framework delivers results in accordance to the results of OMNeT++ simulations, even on large networks. Compared to previous work on the subject and SimGrid, we showed the importance of taking into account queue size, as well as modeling the TCP acknowledgments in topologies with cross-traffic.

As presented in Section 7.4.1, we limited this chapter to UDP and TCP flows to long-lived connections, which is not necessarily a realistic view of nowadays Internet traffic. Hence, with propose in the next chapters to look at the performance evaluation of short UDP and TCP flows.

Key insights and contributions

Formalization of the chosen analytical framework Following the choice made in Chapter 6, we formalized the flow-level network modeling framework (Research Objective **O2.2**).

Extension of the analytical framework In order to be able to study Ethernet networks with low latency, one contribution of this chapter is the extension of the analytical framework which takes into account the effects of cross-traffic on TCP and the modeling of FIFO queues (Research Objective **O2.2**).

Novel tool for using this framework We filled the current lack of tool using flow-level network modeling by proposing PETFEN, a Performance Evaluation Tool for Flow-level network modeling of Ethernet Networks.

Part IV

SHORT FLOWS: SESSIONS AND APPLICATION LAYER MODELING

8. STATISTICAL BANDWIDTH SHARING WITH NON-PERSISTENT TCP FLOWS

Note This chapter is an extended version of our previous publication [123] published in *Proceedings of the 39th IEEE Conference on Local Computer Networks*, 2014 as it was initially submitted for publication. Sections 8.5.3 and 8.5.4 are extension of the initial submission of [123]. Section 8.6.1 has also been partially extended.

8.1 Introduction

We have formalized and extended in Chapter 7 a framework for the performance evaluation of TCP flows sharing multiple bottlenecks on Ethernet networks, but with the limitation that flows are infinite. The purpose and main contribution of this chapter is to extend this work for the evaluation of non-persistent TCP flows. The main advantage of basing our work on flow-level network modeling is that we are able to take advantage of various mechanisms which were previously modeled, namely packet scheduling algorithms, fixed-rate flows, as well as effects on TCP acknowledgments. This is a definitive benefit compared to other methods for evaluating short-lived TCP flows which are often restricted to either simplistic networks elements or idealized bandwidth sharing models.

When studying non-permanent TCP flows, we first have to define what is meant by *non-permanent*. One first characterization is to view a non-permanent flow as an infinite sequence of active and idle phases, which we will also refer as ON/OFF. The second characterization is to define how those phases are related, namely if we consider an *open-loop* or *closed-loop* model. In the open-loop model, the generation of a new active phase is independent of the state of the previous active phase. On the other hand, in the closed-loop model, the generation of a new active phase is dependent on the completion of the previous active one. It means that the generation of a new active phase in the closed-loop model is directly correlated to the level of congestion and speed of the network as opposed to the open-loop model. This choice and influence of an open-loop model versus closed-loop one has already been addressed by various researchers, as shown by the works from Dhamdhere and Dovrolis in [101], Schroeder *et al.* in [219], or Prasad and Dovrolis in [201, 202] for instance. Note that the same distinction between the two behaviors is also made in queuing theory, where networks of queues are either characterized as open or closed (or sometimes a combination of both). We refer to the work from Baskett *et al.* in [50] for more details on this topic.

We argue that for modeling the behavior of users, the closed-loop model is the closest to reality. This claim is supported for instance by Prasad and Dovrolis in [201], where the authors analyzed various publicly available traces and reported that more than 60 % to 80 % of the traffic on well-known ports (mostly HTTP) could be attributed to a closed-loop model.

Hence, we define short-lived or ON/OFF TCP flows as flows alternating between OFF periods of random duration, and ON periods with transfers of data of random size. This simplified model fits HTTP usage, where users receive files and web pages during the ON state, while they read and consume the downloaded data during the OFF state.

Our aim in this chapter is to predict the throughput of individual flows, average durations for transferring a file, average numbers of concurrent users on the studied links, as well as other parameters such as average round-trip times, drop probabilities and queue sizes. We do not make any assumption on flow synchronization, meaning that when a flow is one state, the other flows may switch from active to idle or vice versa. Compared to traditional discrete-event simulations following a Monte-Carlo method, we account for all possible interactions between flows in our proposed framework.

Our model is based on the application and extension of the Engset formula (see [104]) to our use-case, similarly to the work presented by Heyman *et al.* in [138]. In [138], the studied topology is limited to the single bottleneck case with an idealized TCP behavior. Our contribution is the extension of traditional flow-level network modeling as presented in Chapter 7, initially limited to infinite flows, to the study of ON/OFF TCP flows. We aim at modeling the following features: key properties of TCP like slow-start and congestion-avoidance phases, distributions of file sizes and think time, network parameters such as drop probability and round-trip time, as well as applicability to multiple bottleneck topologies.

Structure of this chapter

We first look at related work in Section 8.2. In Section 8.3, we present the case where all ON/OFF TCP flows share the same properties: drop probability, RTT, file size and idle time distributions, and where an analogous model than the Engset formula can be directly applied. With Section 8.4, we generalize the previous case to ON/OFF TCP flows with heterogeneous properties. This model is then extended in Section 8.5 to include infinite TCP flows as well as effects of the TCP slow-start algorithm. We evaluate our framework across different topologies in Section 8.6 and compare our results with simulations done with *ns-2*. Finally, Section 8.7 summarizes and concludes this chapter.

8.2 Related work

In the line of packet-level models for a single infinite TCP flow like the PFTK model proposed by Padhye *et al.* in [195] and presented previously in Chapter 7, various researchers proposed to define the duration of a TCP transfer as a function of the path drop probability, round-trip time and size of the data to transfer. A seminal work on this topic is the CSA model proposed by Cardwell *et al.* in [77], where the authors make use of the PFTK model and extend it to include the slow-start phase of TCP. Similar approaches have subsequently been proposed by Mellia and Zhang in [188] or Sikdar *et al.* in [223] for instance. As for the PFTK model, this model is restricted to the study of a single flow and not the interaction between multiple flows.

Regarding our general approach of using flow-level models as introduced in Chapter 7, a large part of the related work is dedicated to the study of infinite TCP flows, with some extensions to include short-lived flows. Firoiu *et al.* briefly proposed in [111] a model using the CSA model, but with the limitation to flows without any idle period. It is also not clear from their formulation of the flow duration how the bandwidth in the congestion-avoidance phase is computed. Casetti and Meo used a similar ON/OFF model as the one used here in [81], but with a formulation based on a Markov chain for modeling the behavior of TCP. More recently, Baccelli and McDonald proposed in [41] a closed form formula for the distribution of the throughput obtained by an ON/OFF source, but with a limitation to the single bottleneck case where all flows share the same RTT and drop probability. Various works often propose to analyze short-lived TCP flows using a model based on queuing theory, but those models are either limited to the single bottleneck case or they do not account for the slow-start phase of TCP flows. Examples of such models are works from Bu and Towsley in [73] or Lassila *et al.* in [166]. As presented later in Section 8.6.4 for the numerical evaluation of our model, we will see that models taking into account the slow-start phase are generally more accurate for short TCP flows (message sizes below 1 Mbit in our evaluation for example).

We noted previously in Section 6.3 that other approaches for the evaluation of short TCP flows have been proposed. A notable one is the fluid model where the evolution of the TCP window is characterized as a function of time or the value of the window at the previous time increment. For instance, Ajmone Marsan *et al.* extended in [28] the partial differential equations method initially proposed by Misra *et al.* in [192], to ON/OFF TCP flows. As noted in Section 6.3, while such approach is interesting for observing the dynamics of multiple TCP flows in some specific use cases, we are generally more interested in the statistical performances of the flows. Regarding the network utility maximization approach which describes the interactions between flows as an optimization problem, Chang and Liu extended it in [84] to ON/OFF TCP flows, but it is limited to the single bottleneck case.

We also note that all the approaches noted here generally only consider unidirectional communications.

8.3 ON/OFF TCP flows with homogeneous properties

This section presents the first building steps of our model and also introduces part of the notation used in the following sections. Note that we use here part of the notation and results already presented by Heyman *et al.* in [138]. We assume in this first case that we have N ON/OFF TCP flows sharing the same network path, meaning that the flows experience the same drop probability and RTT. This case is illustrated in Figure 8.1. The bandwidth capacity of the path is noted c.

We define an ON/OFF flow as a flow alternating between an idle and an active state. The size of the data transferred during an active or ON state follows the distribution function H with mean $1/\mu < \infty$. This means that the duration of an active phase will depend on the activity of the other flows and may get lengthened due to congestion. There is no synchronization between the flows, which mean that when a flow is in one state, the other flows may change their state. After completion of the file transfer, an idle or OFF phase with a duration following a distribution function G with mean $1/\lambda < \infty$ will take place. All flows share the same distributions. The corresponding random variable of H and G are uncorrelated.

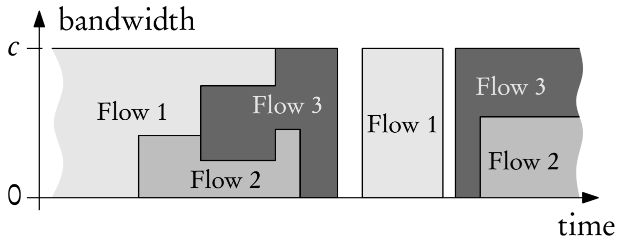


Fig. 8.1: Interaction between multiple ON/OFF TCP flows on a single bottleneck etwork

According to our view of an ON/OFF flow, we study here a closed network with finite population, as illustrated in Figure 8.2. It means that the bottleneck is shared by a maximum number of N concurrent flows in the worst-case.

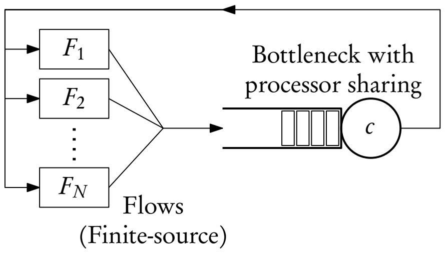


Fig. 8.2: Finite population model of N TCP flows sharing a single bottleneck

We make the following assumptions for this first case: *(i)* a single TCP flow is able to fully utilize the link; *(ii)* there are no packet loss on the link; *(iii)* queuing delays are neglected; *(iv)* the bandwidth sharing of TCP is perfectly fair, such that if n TCP flows are active at the same time, each flow will have a throughput of c/n. The next section is dedicated to the case where the bandwidth sharing is represented by a more realistic model of TCP.

We assume here that when a flow switches from one state to the other, all active flows will immediately adjust their bandwidth to the new number of active flows. This means that we do not include yet the TCP slow-start algorithm if a flow switches from OFF to ON. This case will be discussed in Section 8.5.2.

Let ρ_{ON} be the throughput of a flow in its ON phase. ρ_{ON} is equal to $c/\Psi(t)$ with $\Psi(t)$ the number of active flows at time t. It also depends on $1/\mu$, $1/\lambda$ and N. The exact formulation of it average value $\mathbb{E}[\rho_{\text{ON}}]$ will be derived in Equation (8.4). As a flow needs to transfer a message of mean size $1/\mu$, the average duration of an active phase $\mathbb{E}[T_{\text{ON}}]$ is then:

$$\mathbb{E}[T_{\text{ON}}] = \frac{1/\mu}{\mathbb{E}[\rho_{\text{ON}}]} \tag{8.1}$$

For simplification purpose we do not account in Equation (8.1), nor in Sections 8.4 and 8.5, for protocol overhead. The correct adjustments were performed for the numerical evaluations in Section 8.6.

Let $\Pr(St = \text{ON})$ be the long-run probability that a flow is in its ON state. $\Pr(St = \text{ON})$

can be written as:

$$\Pr(St = \text{ON}) = \lim_{t\to\infty} \frac{\text{total active time by } t}{t}$$

$$\Pr(St = \text{ON}) = \frac{\mathbb{E}[T_{\text{ON}}]}{\mathbb{E}[T_{\text{ON}}] + 1/\lambda} \tag{8.2}$$

Note that the probability $\Pr(St = \text{ON})$ is influenced by the interaction with the other flows, namely $\Pr(St = \text{ON})$ can also be written as:

$$\Pr(St = \text{ON}) = \sum_{j=1}^{N} \Pr(\text{studied flow active}|n(\mathcal{A}) = j) \Pr(n(\mathcal{A}) = j)$$

with $n(\mathcal{A})$ the cardinality of the set of active flows $\mathcal{A}$. As presented in [138], $\Pr(St = \text{ON})$ depends on the distributions G and H only through their means. This property is called insensitivity.

Using the definition of $\Pr(St = \text{ON})$, we derive the probability that j flows are active among the set of studied N flows:

$$\Pr(n(\mathcal{A}) = j) = \binom{N}{j} \Pr(St = \text{ON})^j (1 - \Pr(St = \text{ON}))^{N-j} \tag{8.3}$$

which is analogous to the model of [138] and the Engset formula [104].

Using the law of total probability and our fair bandwidth sharing model, we derive ρ_{ON}, the average throughput of a TCP flow in its active state, as:

$$\mathbb{E}[\rho_{\text{ON}}] = \sum_{j=1}^{N} \left(\frac{c}{j+1} \cdot \frac{\Pr(n(\mathcal{A}) = j)}{\Pr(St = \text{ON})} \right) \tag{8.4}$$

While $1/\mu$, $1/\lambda$ and N are known, $\Pr(St = \text{ON})$ and $\mathbb{E}[\rho_{\text{ON}}]$ are defined by an open-form expression. One way to give a numerical solution is to use a fixed point evaluation of Equations (8.1) to (8.4). We present an algorithm for solving such system of coupled equations in Section 8.4 with Algorithm 8.1, as the network studied here is a special case of the one in Section 8.4.

We see that as N, the number of concurrent flows, grows, $\mathbb{E}[T_{\text{ON}}]$, and hence $\Pr(St = \text{ON})$, increase also. When N tends to ∞, we have $\mathbb{E}[T_{\text{ON}}] \to \infty$ and $\Pr(St = \text{ON}) \to 1$.

We derived in this section the mean bandwidth of each ON/OFF TCP flow in its ON state as a function of the mean size of the data to transfer $1/\mu$ and the number of flows N sharing the same properties.

8.4 ON/OFF TCP flows with heterogeneous properties

We derived in the previous section a first basic model of the interaction between multiple ON/OFF TCP flows where all flows shared the same bottleneck and the same traffic properties (mean transferred file size and mean idle duration). Due to its relative simplicity and scalability, this model may be used as a first estimator for network engineering but is not sufficient for more complex networks.

We generalize in this section the previous model by evaluating flows which do not necessarily share the same characteristics or bottlenecks. Because of those heterogeneous properties, we cannot reuse the simplistic bandwidth sharing used earlier.

We define the set of heterogeneous TCP flows as $\mathcal{F} = \{F_1, \dots, F_N\}$, where each flow has its own path in the topology. A flow F_i can have multiple states, corresponding to the sample space Ω_{F_i}. We denote by $St(F_i)$ the state of flow F_i, and $\Pr(St(F_i) = \omega)$ the probability that the flow F_i is in state ω. As in Section 8.3, there is no synchronization between the flows, which mean that during the time a flow is in one state, the other flows may switch from one state to the other.

As in the previous section, we model an ON/OFF TCP flow F_i with two states so that $\Omega_{F_i} = \{\omega_{F_i}^{\text{ON}}, \omega_{F_i}^{\text{OFF}}\}$. We define $\mathcal{S}$ as the sample space or set of all possible combinations of states of the different studied flows. We define C as an event of the sample space $\mathcal{S}$, or in other words, a possible combination of the different flow states. Each combination C has a probability noted $\Pr(C)$. Using those properties, we characterize the studied probability space as:

$$\mathcal{S} = \prod_{F_i \in \mathcal{F}} \Omega_{F_i} \tag{8.5}$$

$$C = \{(F_1, \omega_{F_1}), \dots, (F_N, \omega_{F_N})\} \tag{8.6}$$

$$\Pr(C) = \prod_{(F_i, \omega_k) \in C} \Pr(St(F_i) = \omega_k) \tag{8.7}$$

The goal of this section is to derive the mean throughput of each flow according to its state and the mean duration of the state.

Step A We first compute in Section 8.4.1 the steady-state throughput of each flow $F \in \mathcal{F}$ in every possible combination of state C, using standard flow-level modeling. We denote this bandwidth $\rho(F|C)$.

Step B Using the results of Section 8.3, we then compute in Section 8.4.2 the probability of occurrence of each combination C, and derive the mean bandwidth of each flow according to its state, denoted $\mathbb{E}[\rho(F)|St(F) = \omega]$.

Step C Finally, using the law of total probability with the two previous results, we derive in Section 8.4.3 various performance measures of the network, such as mean bandwidths or average number of concurrent flows.

A summary of the method is illustrated in Figure 8.3.

8.4.1 Step A: Evaluation of the flows for each possible combination C

In this first step, we use flow-level modeling to determine the throughput of each flow as if flows were infinite, for every possible combination of flow states. We make here use of the results presented in Chapter 7. For the purpose of this section, we briefly recall the most important terms and notations already presented in Chapter 7.

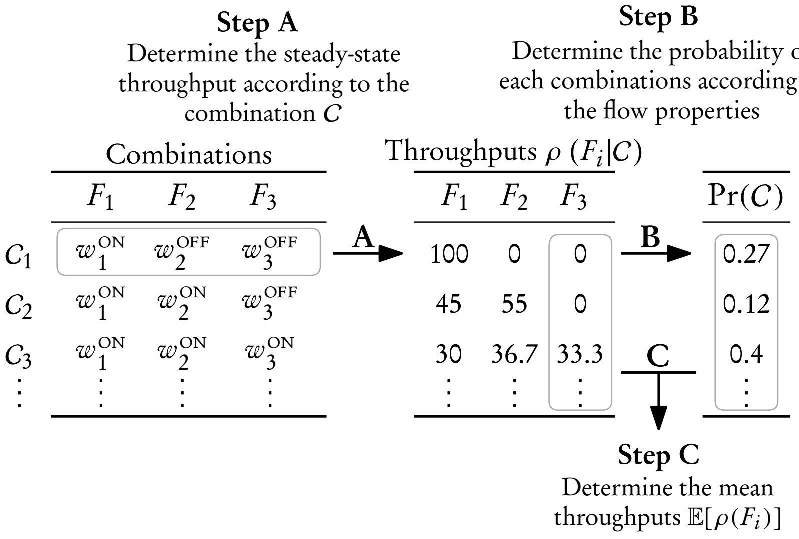

Step A
Determine the steady-state throughput according to the combination C
Step B
Determine the probability of each combinations according to the flow properties
Combinations
F_1 F_2 F_3
C_1 w_1^{ON} w_2^{OFF} w_3^{OFF}
C_2 w_1^{ON} w_2^{ON} w_3^{OFF}
C_3 w_1^{ON} w_2^{ON} w_3^{ON}
A
Throughputs $\rho\ (F_i|C)$
F_1 F_2 F_3
100 0 0
45 55 0
30 36.7 33.3
B
C
Pr(C)
0.27
0.12
0.4
Step C
Determine the mean throughputs $\mathbb{E}[\rho(F_i)]$

Fig. 8.3: Summary of the method presented in Section 8.4

Our flow-level network model consists of *servers*, which represent the different queues and links of the network, as well as *flows*, which represent the communications between the nodes of the network. As presented in Chapter 7, the bandwidth of a flow F is a function ρ_F of the set of traversed servers $\mathcal{S}$. Compared to the initial model, a bandwidth model is also dependent on the state of the flow, noted here ω.

According to the model of Section 8.3, a flow is either in the ON state or the OFF state. For the OFF state, when the flow is inactive, we have $\rho_F(\mathcal{S}) = 0$ regardless of the state of the network. For the ON state, when a flow transfers data, we use the PFTK formula derived ([195]) as presented in Axiom 7.7. Note that The PFTK formula describes only the *congestion-avoidance* phase of an active TCP connection. We will derive later in Section 8.5.2 a more advanced model where we also take into account the *slow-start* phase of a TCP flow.

For each combination of flow states in $\mathcal{S}$, we use the flow-level network model defined in Section 7.3 to derive the steady-state bandwidth of each flow F as well as other performance measures such as queue size and drop probabilities. We denote by $\rho(F|C)$ the throughput of flow according to the combination C.

8.4.2 Step B: Evaluation of the probabilities

Now that the steady-state throughput of the flows is known for each possible combination of flow states, we derive the time spent in each state using the concepts introduced in Section 8.3. The process described here is similar to the one expressed by Equations (8.1) to (8.4). The size of the data transferred during an ON state has distribution function H_F with mean $1/\mu_F < \infty$. The duration of an idle state has distribution function G_F with mean $1/\lambda_F < \infty$. Let $\mathbb{E}[T(\omega)]$ be the mean duration of state ω.

The long-run probability that flow F_i is in state ω is noted $\Pr(St(F_i) = \omega)$ and is specified by:

$$\Pr\left(St(F_i) = \omega\right) = \lim_{t\to\infty} \frac{\text{total time in state } \omega \text{ by } t}{t} \tag{8.8}$$

$$= \frac{\mathbb{E}[T(\omega)]}{\sum_{\omega_j \in \Omega_i} \mathbb{E}\left[T(\omega_j)\right]} \tag{8.9}$$

Using the law of total probability, we derive the mean throughput of a flow according to its state:

$$\mathbb{E}[\rho(F)|St(F) = \omega] = \sum_{\{\forall C \in \mathcal{S} | St(F)=\omega\}} \frac{\Pr(C) \cdot \rho(F|C)}{\Pr(St(F) = \omega)} \tag{8.10}$$

where $\Pr(C)$, the probability of combination C, has already been defined in Equation (8.7). According to our two-states model, the mean duration of each state is then:

$$\mathbb{E}\left[T(\omega_F^{\text{OFF}})\right] = 1/\lambda_F \tag{8.11}$$

$$\mathbb{E}\left[T(\omega_F^{\text{ON}})\right] = \frac{1/\mu_F}{\mathbb{E}\left[\rho(F)\middle|St(F) = \omega_F^{\text{ON}}\right]} \tag{8.12}$$

$\mathbb{E}\left[T(\omega_F^{\text{ON}})\right]$ corresponds to mean duration needed to transfer a file.

As in Section 8.3, Equations (8.9) to (8.12) and Equation (8.7) are coupled and we use a fixed-point evaluation to find the equilibrium of the system. Algorithm 8.1 describes a possible method to find a numerical solution. We distinguish two parts in the algorithm. The first part (lines 1 to 3) sets the probabilities $\Pr(St(F) = \omega)$ to an initial numerical value. The second part (lines 4 to 14) computes numerical values to the different properties given in Equations (8.9) to (8.12) and Equation (8.7). Note that we also define a safeguard function in order to avoid an infinite loop (line 13) in case an equilibrium cannot be reached. The simplest algorithm for this is to limit the number of iterations of the loop defined by lines 4 to 14. Note that Algorithm 8.1 is independent to the number of states of a flow.

8.4.3 Step C: Results of the topology

We derived in Sections 8.4.1 and 8.4.2 the throughput of each flow according to the combination of flow states C, as well as the probability of having each combination $\Pr(C)$. Using the law of total probability, we obtain the mean bandwidth $\mathbb{E}[\rho(F)]$ of each flow as:

$$\mathbb{E}[\rho(F)] = \sum_{\omega \in \Omega_F} \left\{\mathbb{E}[\rho(F)|St(F) = \omega] \cdot \Pr(St(F) = \omega)\right\} \tag{8.13}$$

Similarly, we derive the probability of having n active flows and the mean number of active flows $\mathbb{E}[\mathcal{A}]$:

$$\Pr(n \text{ flows active}) = \sum_{\forall C \in \mathcal{S}} \mathbb{I}\{\mathcal{A}(C) = n\} \cdot \Pr(C) \tag{8.14}$$

$$\mathbb{E}[\mathcal{A}] = \sum_{n=0}^{N} n \cdot \Pr(n \text{ flows active}) \tag{8.15}$$

Algorithm 8.1 Fixed-point evaluation algorithm for Equation (8.7) and (8.9) to (8.12)

```
Require: Bandwidth of all flows in $\mathcal{F}$ and for all possible combinations, meaning
    $\rho(F|C), \forall F \in \mathcal{F}, \forall C$
1: for all $F$ in $\mathcal{F}$, $\omega$ in $\Omega_F$ do
2:     $\Pr(St(F) = \omega) = 1/|\Omega_F|$
3: end for

4: while equilibrium not reached do
5:     for all combination $C$ do
6:         Compute $\Pr(C)$ according to Equation (8.7)
7:     end for
8:     for all $F$ in $\mathcal{F}$, $\omega$ in $\Omega_F$ do
9:         Compute $\mathbb{E}[\rho(F)|St(F) = \omega]$ according to Equation (8.10)
10:        Compute $\mathbb{E}\left[T(\omega_F^{\mathrm{ON}})\right]$ according to Equation (8.12)
11:        Compute $\Pr(St(F) = \omega)$ according to Equation (8.9)
12:    end for
13:    SAFEGUARD()                                ▷ Function to avoid infinite loop
14: end while
```

Again using the law of total probability, other mean performance measures can be computed, such as mean round-trip times or drop probabilities:

$$\mathbb{E}[RTT_F] = \sum_{\forall C \in \mathcal{S}} RTT_F(C) \cdot \Pr(C) \tag{8.16}$$

$$\mathbb{E}[p_F] = \sum_{\forall C \in \mathcal{S}} p_F(C) \cdot \Pr(C) \tag{8.17}$$

Note also that distributions of performance measures can be derived using the probability of each combination, such as for instance here the cumulative distribution function of the round-trip time of flow F:

$$\Pr(RTT_F \leq \tau) = \sum_{\forall C \in \mathcal{S}} \mathbb{I}\{RTT_F(C) \leq \tau\} \cdot \Pr(C) \tag{8.18}$$

As described in this section, part of the framework described here uses the traditional flow-level network models for evaluating the steady-state bandwidth of the studied flows. This means that we can take advantage of all the effort that have been made on flow-level network models, namely: adaptability to topologies with multiple bottlenecks, support of scheduling algorithm as well as modeling of various effects of TCP such as cross-traffic.

8.5 Extensions of the model

8.5.1 Mixture of ON/OFF and infinite TCP flows

While the development made earlier treated only flows with two states (idle and active), the formalism introduced in Section 8.4 can be easily adapted to a mixture of ON/OFF and infinite TCP flows. The motivation for this is the evaluation of the interaction between long-lived TCP connections, often referred as *elephants*, and short-lived TCP connections, often referred as *mice*. In the case of an infinite TCP flow F, we only have one state, so that:

$$\Omega_F = \{\omega_F^{\mathrm{ON}}\} \tag{8.19}$$

$$\mathbb{E}\left[T(\omega_F^{\mathrm{ON}})\right] = \infty \tag{8.20}$$

$$\Pr\left(St(F) = \omega_F^{\mathrm{ON}}\right) = 1 \tag{8.21}$$

8.5.2 Inclusion of TCP slow-start

We noted in Section 8.3 that we did not take into account the initial phase of a TCP flow, also known as *slow-start*. A model including only the *congestion-avoidance* phase of TCP might be valid for large transfers where the time spent in slow-start is negligible, but it does not hold for small transfers, where most of the active state is spent in slow-start.

To overcome this issue, we extend our two-states flow model to three states: idle ω^{OFF}, active in slow-start $\omega^{\mathrm{ON_{SS}}}$ and active in congestion-avoidance $\omega^{\mathrm{ON_{CA}}}$, so that the state sample space of a flow is now: $\Omega = \{\omega^{\mathrm{OFF}}, \omega^{\mathrm{ON_{SS}}}, \omega^{\mathrm{ON_{CA}}}\}$. This three-states flow is illustrated in Figure 8.4.

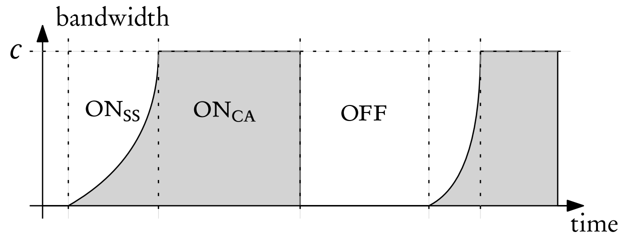


Fig. 8.4: ON/OFF TCP flow model with three states: ON in slow-start ($\mathrm{ON_{SS}}$), ON in congestion avoidance ($\mathrm{ON_{CA}}$) and OFF

In order to describe the behavior of a TCP flow in slow-start, we use the results from Cardwell *et al.* [77], often referred as the CSA model. With this model, we evaluate how much time is spent in slow-start, as well as the size of the data transferred during the slow-start state. The duration of the congestion-avoidance state will then be reduced according to the data already transferred during slow-start.

The mean time spent in the slow-start state $\omega^{\mathrm{ON_{SS}}}$ is the sum of three durations:

- The mean time for three-way TCP handshake ($\mathbb{E}[L_h]$ in [77, Equation (4)]);
- The mean time exponential growth phase ($\mathbb{E}\left[T_{exp}\right]$);
- The mean time needed to recover from the first packet loss which ends the slow-start ($\mathbb{E}[T_{loss}]$ in [77, Equation (20)]).

We then derive the complete duration of the $\omega_F^{\mathrm{ON_{SS}}}$ state and it associated throughput:

$$\mathbb{E}\left[T(\omega_F^{\mathrm{ON_{SS}}})\right] = \mathbb{E}[L_h] + \mathbb{E}\left[T_{exp}\right] + \mathbb{E}[T_{loss}] \tag{8.22}$$

$$\mathbb{E}\left[\rho(F)\middle|St(F) = \omega_F^{\mathrm{ON_{SS}}}\right] = \frac{\mathbb{E}[d_{ss}]}{\mathbb{E}\left[T(\omega_F^{\mathrm{ON_{SS}}})\right]} \tag{8.23}$$

with $\mathbb{E}\left[T_{exp}\right]$, the time spent in the exponential phase:

$$\mathbb{E}\left[T_{exp}\right] = \begin{cases} RTT \log_\gamma\left(\frac{\mathbb{E}[d_{ss}](\gamma-1)}{w_1} + 1\right) & \text{if } W_{ss} \leq W_{max} \\ RTT\left(\log_\gamma\left(\frac{W_{max}}{w_1}\right) + 1\right) & \text{otherwise} \end{cases} \tag{8.24}$$

with $\gamma = 1 + 1/b$. Note that $\mathbb{E}\left[T_{exp}\right]$ is different from $\mathbb{E}[T_{ss}]$ in [77, Equation (15)]. In [77] it is assumed that when the TCP window reaches its maximum window size W_{max}, the TCP window will remain constant and all the remaining data will be transfered. We differ here by saying that we switch to the congestion-avoidance state to account for the interaction with other flows.

The $\omega_F^{\mathrm{ON_{CA}}}$ state has to account for the data that was already transferred during the $\omega^{\mathrm{ON_{SS}}}$ state, so that if there is still data to be transfered (*i.e.* $\mathbb{E}[d_{ss}] < {}^{1}/_{\mu}$, with $\mathbb{E}[d_{ss}]$ from [77, Equation (5)]):

$$\mathbb{E}\left[T(\omega_F^{\mathrm{ON_{CA}}})\right] = \frac{1/\mu_F - \mathbb{E}[d_{ss}]}{\mathbb{E}\left[\rho(F)\middle|St(F) = \omega_F^{\mathrm{ON_{CA}}}\right]} + RTT_F \tag{8.25}$$

Otherwise $\mathbb{E}\left[T(\omega_F^{\mathrm{ON_{CA}}})\right] = 0$. We add one round-trip time to $\mathbb{E}[T(\omega^{\mathrm{ON_{CA}}})]$ to account for latency to send the first data segment in the congestion-avoidance phase, and latency for the last acknowledgment.

Now that the three-states flow model is defined, we use the same method as in Section 8.4 to solve numerically our system.

8.5.3 Bidirectional ON/OFF flow

Note This section is an extension of the original publication [123]. Because the model of unidirectional communications used in Sections 8.3 and 8.4 is essentially artificial, we extend the original publication to include a more realistic model where messages are exchanges bidirectionally. We note that this model also partially introduces the work presented in Chapter 9.

One shortcoming of the ON/OFF model presented earlier is that communications are unidirectional, which does not match what happens in common network protocols. A conventional idiom found in network protocols is the request-reply pattern, where a client initiates a TCP connection with a server, sends a request to the server, and the server replies according to the request. An example of such request-reply pattern is HTTP, where the most common case is that a client (a web-browser) sends a HTTP `GET` requesting a file (HTML file, image, ...) and the server sends the requested file. A similar idea was proposed for instance by Weigle *et al.* with the *tmix* tool [241] to simulate realistic network traffic in *ns-2*.

Hence, we propose here a simple bidirectional ON/OFF model as presented in Figure 8.5. For simplicity purpose, we reuse here the basic model presented in Section 8.3. A client first sends a request message with mean size $1/\mu^{req}$. Secondly, when the request message has been fully received by the server, a reply message with mean size $1/\mu^{rsp}$ is sent. Finally, when the reply has been fully received by the client, it enters an idle state with mean time T_{OFF}.

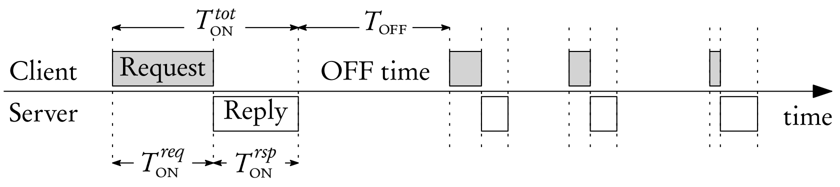


Fig. 8.5: Bidirectional ON/OFF TCP flow model

We define T_{ON}^{req}, respectively T_{ON}^{rsp}, as the average duration of the active phase for the request message, respectively the reply message. In order to model this bidirectional flow according to the model presented in Section 8.3, we split this flow in two unidirectional ON/OFF sub-flows. The expression of the mean duration of the active period of the ON/OFF client sub-flow (T_{ON}^{req}) is similar to Equation (8.1), The idle period of the sub-flow (T_{OFF}^{req}) corresponds to the sum of the global idle period (T_{OFF}) and the duration of the active period of the server sub-flow (T_{ON}^{rsp}). A similar relation is established for the server for the server. In mathematical terms, this can be written as:

$$\text{client sub-flow:} \begin{cases} \mathbb{E}\left[T_{\text{ON}}^{req}\right] &= \frac{1/\mu^{req}}{\mathbb{E}\left[\rho_{\text{ON}}^{req}\right]} \\ \mathbb{E}\left[T_{\text{OFF}}^{req}\right] &= T_{\text{OFF}} + \mathbb{E}\left[T_{\text{ON}}^{rsp}\right] \end{cases} \tag{8.26}$$

$$\text{server sub-flow:} \begin{cases} \mathbb{E}\left[T_{\text{ON}}^{rsp}\right] &= \frac{1/\mu^{rsp}}{\mathbb{E}\left[\rho_{\text{ON}}^{rsp}\right]} \\ \mathbb{E}\left[T_{\text{OFF}}^{rsp}\right] &= T_{\text{OFF}} + \mathbb{E}\left[T_{\text{ON}}^{req}\right] \end{cases} \tag{8.27}$$

Equations (8.26) and (8.27) are coupled, and as done earlier for a similar set of equations, we use a fixed-point evaluation for the numerical evaluation of this system of equation, as highlighted in Algorithm 8.2. We note that the approach presented here can be easily adapted and applied to the more advanced model presented in Section 8.4.

Algorithm 8.2 Fixed-point evaluation algorithm for Equations (8.26) and (8.27)

```
1: $\mathbb{E}\left[T_{\text{ON}}^{req}\right] \leftarrow 0$
2: $\mathbb{E}\left[T_{\text{ON}}^{rsp}\right] \leftarrow 0$

3: while equilibrium not reached do
4:     Compute $\mathbb{E}\left[T_{\text{ON}}^{req}\right]$ according to Equation (8.26)
5:     Compute $\mathbb{E}\left[T_{\text{ON}}^{req}\right]$ of the client sub-flow according to Equations (8.1) to (8.4)
6:     Compute $\mathbb{E}\left[T_{\text{OFF}}^{rsp}\right]$ according to Equation (8.27)
7:     Compute $\mathbb{E}\left[T_{\text{ON}}^{rsp}\right]$ of the server sub-flow according to Equations (8.1) to (8.4)
8: end while
```

8.5.4 Computational cost reduction

Note This section is an extension of the original publication [123], as the question of computational cost of the method described in Section 8.4 was not originally discussed.

We note that while the models described in Section 8.4 and Sections 8.5.1 to 8.5.3 provide accurate results compared to simulation as highlighted later in Section 8.6, one drawback of the model is its high computational cost due to the state-space explosion.

One solution to overcome it is to use the notion of *grouping*. In this case, flows with the same characteristics are grouped into classes. The classes are then studied instead of the individual flows. This process is akin to the first model of ON/OFF flows described in Section 8.3.

Similarly, one can use the fact that on larger topologies, flows do not necessarily share the same bottlenecks and hence computing their interaction is not necessary. This idea was for instance proposed by Scharbarg and Fraboul for the efficient study of AFDX network with deterministic network calculus in [214]. They showed that on an industrial AFDX configuration, eliminating flows which do not interact together eliminates an average of 60 % of the flows.

Regarding the extended model presented in Section 8.5.2 including TCP slow-start, adding a state in the model means that it also increases the number of possible combinations of states where if N is the number of studied flows, we go from 2^N to 3^N combinations to evaluate. While this results in a higher computation cost, we show later in Section 8.6.4 that it provides more accurate results for small transfers than the two-states model from Section 8.4. To overcome this drawback, one solution is to use the three-states model for small transfers, and use the two-state model for larger transfers where the influence of the slow-start phase is minimal compared to the steady-state phase. One could use a simple threshold on the transfer size for instance to distinguish between small or large transfer.

8.6 Numerical evaluation

We evaluate in this section the accuracy of our model by comparing analytical results of our model with results of discrete-event simulations made with *ns-2* [15]. We limit here the studied cases to topologies where counterintuitive effects of TCP such as cross-traffic are not present as the flow-level model used in Section 8.4.1 does not account for such effect. We note that those more advanced topologies can still be investigated using the results developed in Chapter 7.

Across all the simulations described here, we use the `Agent/TCP/Reno` agent of *ns-2* for the sender part with a maximum window size W_{max} of 20 packets (default value used in *ns-2*), and an initial window size w_1 of 1. The timeout delay T_0 is set to 1 s. The receiver uses `Agent/TCPSink`. Routers and switches are configured with a drop-tail policy. When not specified otherwise, we focus here on the evaluation of Ethernet topologies, with a maximum packet size of 1538 B, which includes the Ethernet preamble and inter-frame gap, and a TCP message size of 1460 B.

For the evaluation of the various topologies presented here, we use our previous work on flow-level model, which has been presented in Chapter 7, where we already showed its accuracy for the evaluation of the steady-state bandwidth of infinite TCP flows (see the numerical evaluation in Section 7.8).

8.6.1 Insensitivity to the distributions

In order to show the insensitivity of the model to the distributions G and H, we simulated several distributions for the file size and the idle time. We use the same topology as Heyman *et al.* in [138], pictured in Figure 8.6. Several TCP sources are connected on the same router using a 128 kbit/s link with a 100 μs delay in each way. The destination of those sources is connected on the router using a 1.5 Mbit/s link with a 150 ms delay in each way. The mean file size transferred by the sources is set to 200 kB, and the mean idle to 5 s. We use the two following distributions for the file size and the idle time: exponential and Pareto with shape parameter of 2.5. We simulate three cases where the number of sources N is set to 5, 10 or 15, which makes it a total of 12 scenarios.

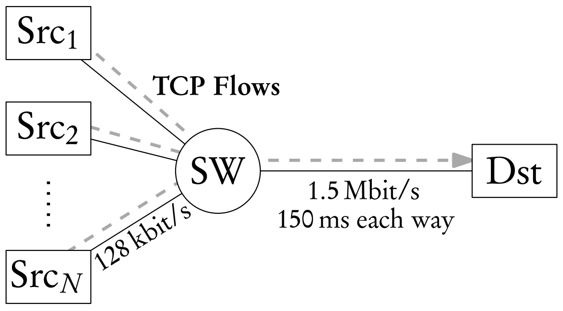


Fig. 8.6: Topology used for the evaluation in Section 8.6.1, similar to the one used [138]. N ON/OFF TCP sources send 200 kB messages to a single destination.

We evaluate here the density of the number of concurrent flows in the topology, which is presented in Figure 8.7. The distribution type has indeed a low influence on the number of concurrent flows as we see little to no differences between the different runs of the *ns-2* simulations. We see that the model accurately describes the steady-state performance of the system with regards to the number of concurrent flows regardless of the distributions.

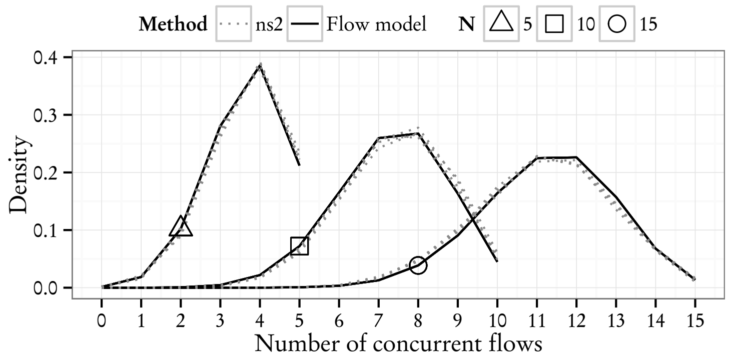


Fig. 8.7: Number of concurrent flows for the topology presented on Figure 8.6, with 5, 10 or 15 TCP sources, and with different distributions for the file size and idle time

Note The remaining text of this section and the results presented in Figure 8.8 are extensions of the original publication [123]. Those additional results highlight the sensitivity of the simulation's parameters.

We noted in Section 7.8.1 that the duration and warm-up period of the simulations have an impact on the accuracy of the result. To assess the sensitivity of those parameters with the end result, we evaluated the topology presented in Figure 8.6 with different simulation durations and warm-up periods. As in Section 7.8.1, we define a reference simulation corresponding to the simulation with the longest warm-up period and duration.

In order to compare different simulation parameters, we compare the distribution of number of active flows using using a two-sample Kolmogorov-Smirnov test. Namely, by denoting $F_{\mathcal{A}}$ as the empirical cumulative distribution (ECDF) of the number of active flows, the Kolmogorov-Smirnov statistic is defined as:

$$\sup_n \left| F_{\mathcal{A}}^{ref}(n) - F_{\mathcal{A}}(n) \right| \tag{8.28}$$

with $F_{\mathcal{A}}^{ref}$ the ECDF of the reference simulation.

The results of the comparison are presented in Figure 8.8. We note that the remarks made in Section 7.8.1 also apply here. Extending the simulation duration reduces the error of the distribution of number of active flows compared to the reference simulation.

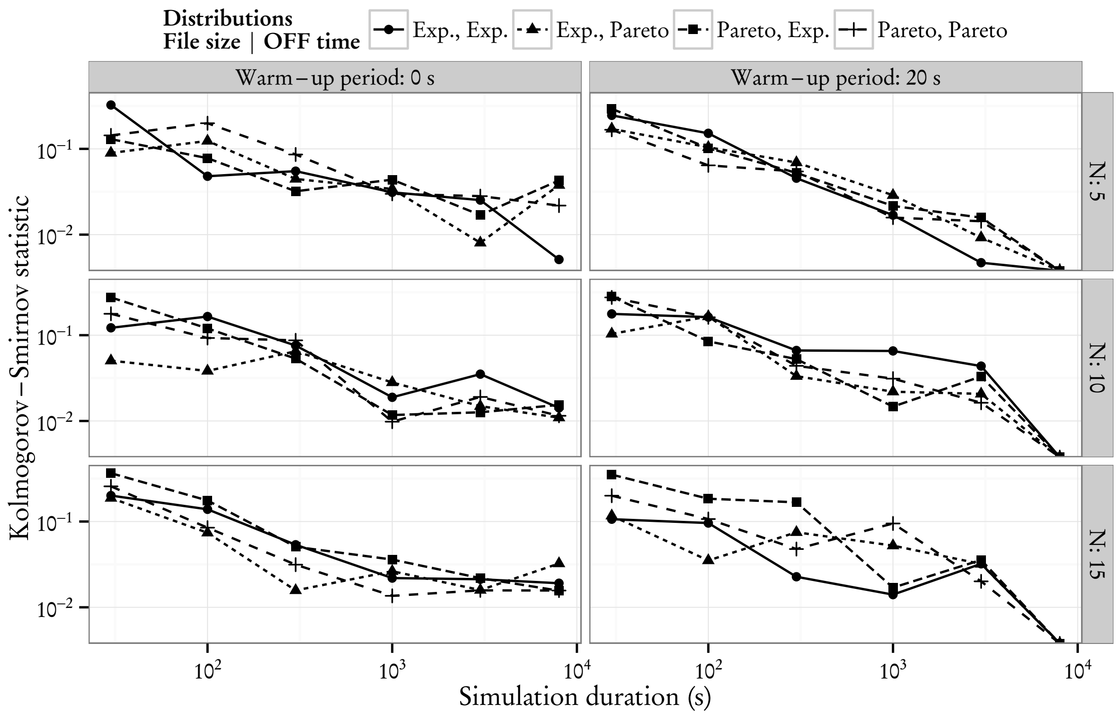


Fig. 8.8: Sensitivity of the results with regards to the parameters of the simulation

8.6.2 Ethernet dumbbell topology with one class

We evaluate here the influence of the number of concurrent flows on the mean-bandwidth of the flows. We use the topology presented in Figure 8.9, with the link between SW_1 and SW_2 set to 10 Mbit/s. Links are considered to have a null error rate and a latency corresponding to the propagation time (5×10^{-8} s for 10 m cables). The idle duration follows an exponential distribution of mean 5 s, and the message size to transfer follows an exponential distribution of mean 10 MB.

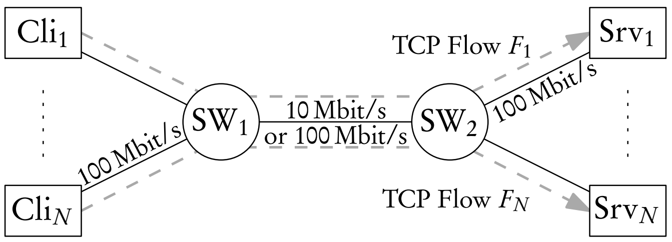


Fig. 8.9: Dumbbell topology with N TCP sources and N TCP destinations. Properties of the links (latency and drop probability) vary through the different use cases studied here.

We present in Figure 8.10 the mean bandwidth of individual flows. The number of clients in the topology varies between 1 and 15. We see that the model accurately predicts the results of the simulations.

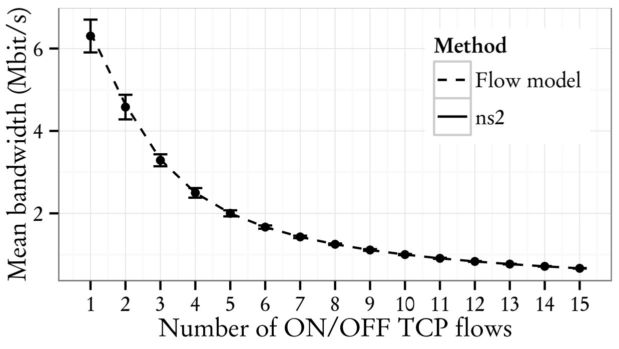


Fig. 8.10: Mean bandwidth of individual flows according to the number of simultaneous flows in the dumbbell topology presented in Figure 8.9. Error bars for the simulation results correspond to a 95 % confidence interval.

8.6.3 Ethernet dumbbell topology with two classes

To illustrate our framework with flows of heterogeneous properties, we use the topology presented in Figure 8.9 with two types of clients: n flows of class 1 with mean file size of 10 MB and mean idle time of 10 s, and one flow of class 2 with a mean file size of 30 MB and a mean idle time of 1 s. Both classes follow a Pareto distribution of shape 2.5 for the file size, and an exponential distribution for the idle time.

Results regarding the mean bandwidth of each flow are presented on Figure 8.11. The results of the model are in accordance with the results of the simulation. Because of its shorter idle time, the flow of class 2 uses more resources than the flows of class 1.

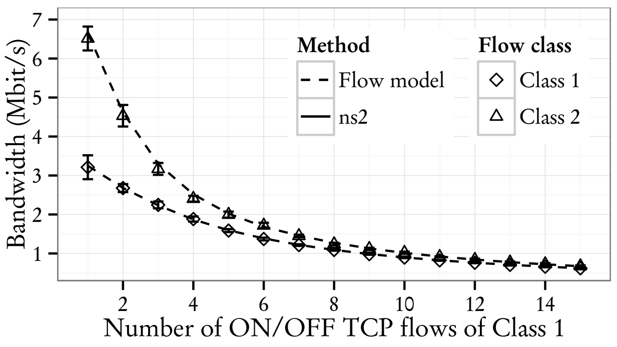


Fig. 8.11: Mean bandwidth of individual flows according to the number of simultaneous flows in the dumbbell topology presented in Figure 8.9. Error bars for the simulation results correspond to a 95 % confidence interval.

8.6.4 Slow-start and three-states flow model evaluation

We study here the impact of the slow-start algorithm on the accuracy of our model. We first assess the accuracy of the slow-start model which was developed in Section 8.5.2. We use here the dumbbell topology presented in Figure 8.9, with one TCP source and one TCP destination. The latency between SW_1 and SW_2 is set to 150 ms (each way), with a drop probability of 5×10^{-3} following a Bernoulli model (each way). Each test file size is simulated 20 times.

We first measure the average time needed to transfer a file between 1 kbit and 1.6 Mbit when only one TCP flow is in the topology ($N = 1$). The results are presented on Figure 8.12, where we compare the measures made on *ns-2* with the results of our analytical model. The gray ribbon on the figure corresponds to a 95 % confidence interval of the simulation results. As expected, the model including the slow-start part of the TCP algorithm produces better results than the model without. We see the influence of the slow-start part especially for file sizes up to 400 kbit.

Based on those results, we evaluate the same topology, with $N = 10$ clients. The file size follows a Pareto distribution with mean size 340 kbit and shape 2.5, which means that the TCP connection will spend the majority of its time in the slow-start phase. The duration of an idle period follows an exponential distribution with mean 3 s, increasing the probability of having more than one active flow at a single time. We present in Figure 8.13 the number of concurrent flows in the topology. As expected, our model including the slow-start part of the TCP transfer produces more accurate results compared to the *ns-2* simulations than the model which only includes the congestion-avoidance phase.

Using the three-states flow model results in a higher number of combinations to evaluate. On way to avoid this drawback is to use the three-states flow only for small transfers, and keep the two-states model for larger transfers as mentioned in Section 8.5.4. In order to give a quantitative meaning to this idea, we evaluate the same topology across a range of different message sizes. Results are presented in Figure 8.14. We observe for message sizes below 1 Mbit a similar behavior as the one observed in Figure 8.14. After 4 Mbit, the difference between the two methods becomes less noticeable.

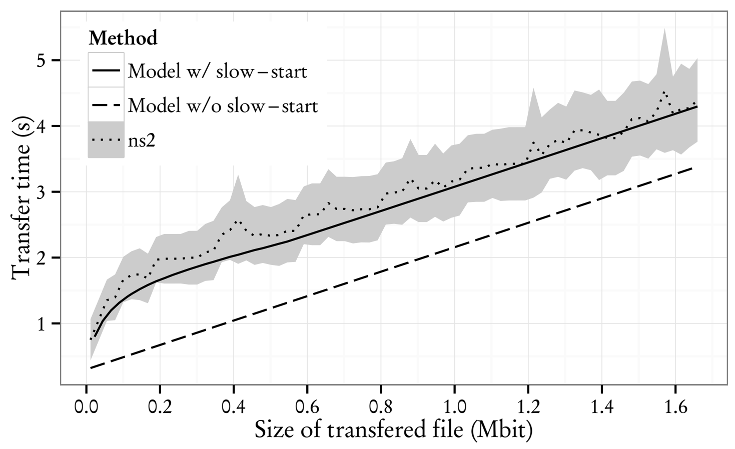


Fig. 8.12: Comparison between model and simulation regarding the time needed to transfer a file on the dumbbell topology, with an RTT of 300 ms and a drop probability of 5×10^{-3} in both directions. The gray ribbon corresponds to a 95 % confidence interval for the *ns-2* simulations.

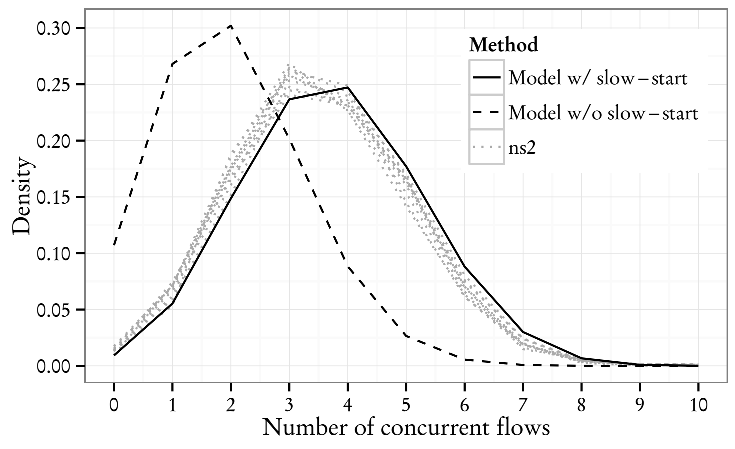


Fig. 8.13: Number of concurrent flows on the dumbbell topology with $N = 10$ ON/OFF TCP flows, a file size following a Pareto distribution of mean 340 kbit and shape 2.5, and an idle duration following an exponential distribution of mean 3 s.

8.7 Conclusion on stochastic flow-level network modeling

We proposed in Chapter 7 an analytical framework for the evaluation of infinite UDP and TCP flows, where it was shown to provide accurate results. The main contribution of this chapter is to leverage those results and propose a stochastic extension of this analytical in order study short flows. We proposed to use here a simple unidirectional ON/OFF model for characterizing the stochastic behavior of short flows. During an active phase a certain amount of data following a general distribution is sent between a sender and a receiver over the network. After this phase, an idle phase of duration following a general distribution takes place.

We first focused on the study of a single bottleneck topology shared by multiple TCP

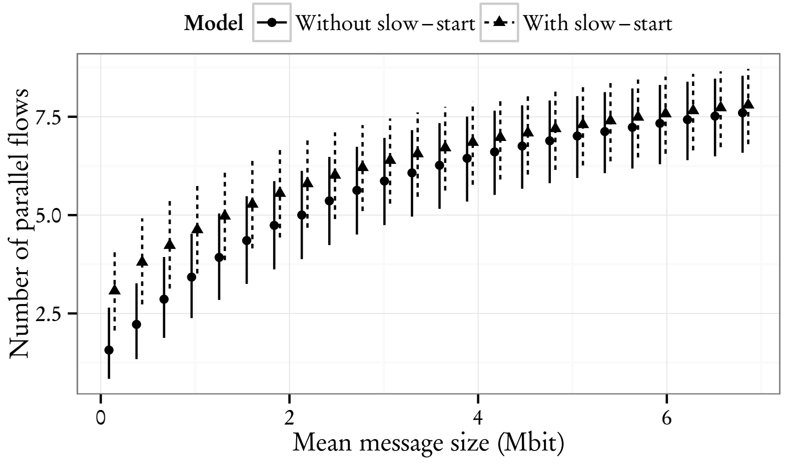


Fig. 8.14: Number of concurrent flows on the dumbbell topology with $N = 10$ ON/OFF TCP flows, a file size following a Pareto distribution with shape 2.5, and an idle duration following an exponential distribution of mean 3 s. The vertical lines correspond to the 25–75 percentile of the density of number of parallel flow, and the dots correspond to the mean value.

flows having all the same properties. This resulted in the characterization of the mean transfer duration as a function of the mean data size to transfer and the mean idle time. A simple and efficient algorithm with a computational complexity independent of the number of studied flows was proposed for this case. This result was then extended to the performance evaluation of multiple bottlenecks topologies with multiple ON/OFF TCP flows having heterogeneous properties.

Further extensions of this model were then proposed. As the two previous models ignored the slow-start phase of TCP, we first extended our model in order to take it into account and increase the accuracy of the model for flows where slow-start has a large impact on the performances. Secondly, we proposed to extend our model to the study of bidirectional flows in order to model the request-reply principle often used in network protocols. Finally, we proposed methods to reduce the computational cost of using those more advanced models. We showed via a numerical evaluation the accuracy of our model and where the more advanced models may be used in order to increase accuracy.

We note that this stochastic extension of flow-level network modeling provides some advantages which are not necessarily available in the other analytical frameworks presented earlier in Section 6.3, namely:

- The inclusion of slow-start and bidirectional flows;
- The ability to study topologies with multiple bottlenecks;
- The use of realistic models for the bandwidth of TCP and its eventual unexpected behavior with cross-traffic modeled in Chapter 7;
- The possibility to use this framework on Ethernet network topologies with various packet scheduling algorithms.

Most of those advantages come from leveraging the results previous presented in Chapter 7 (and its eventual future extensions). Regarding drawbacks, the computational cost of this approach is one, although various methods were proposed to overcome this issue. Another drawback is that the method developed in this chapter is limited to mean performances of stochastic flows. We will address this point in Chapter 10, where we extend partly this method in order to get bounds on the tail performances.

Key insights and contributions

Stochastic extension of flow-level modeling We proposed in this chapter methods to leverage the results of flow-level network modeling in order to study short UDP and TCP flows.

Extension to short TCP flow with slow-start In order to increase the accuracy of our model for short TCP flows where slow-start plays an important role, an extension of our framework was proposed.

Extension to bidirectional TCP flows and first step toward application level We also proposed an extension of our stochastic ON/OFF model to study network protocols based on the bidirectional request-reply paradigm.

9. ADVANCED MODELING: APPLICATION LAYER AND USER BEHAVIOR

Note This chapter is based on our previous publication [120], published in *Proceedings of the 6th International Conference on Simulation Tools and Techniques*, 2013. Compared to the original publication, Section 9.2 as been extended to include more related work, and the *packet* layer has been renamed *message* layer to disambiguate its meaning.

9.1 Introduction

We developed in Chapters 7 and 8 various frameworks to have mathematical models of the behavior of TCP and UDP flows. While Chapter 7 focuses on infinite flows and Chapter 8 on short flows, those models are inherently simplifications of the behavior of real protocols and concentrate on the transport layer of OSI model. Similarly, the various simulations performed in Part II used simplified traffic generators for simulating network flows. We propose in this chapter to characterize the behavior at the application layer and at the user level in order to have a more realistic view of network traffic.

The goals and contributions of this chapter are to first develop a general mathematical model able to represent realistic traffic flows and secondly provide a method to effectively yield values to the different parameters of the model. We aim at using this work with the mathematical frameworks presented in Chapters 7 and 8, as well as in simulations as highlighted in Part II. The model should be able to adapt to a wide range of protocols, and characterize accurately traffic patterns seen in a real network at different ranges of time scales, in order to reproduce both similar average bandwidths and burstiness. We base this model on the following hierarchy: user, session, connection, and packet level.

As a general challenge when transposing mathematical models to real use-cases is to choose which concrete numerical values to use, we propose to base the values of the parameters of our model on the analysis of real traffic capture. For this purpose, we introduce here a suite of two tools called *RENETO*, *RE*alistic *NE*twork *T*raffic for OMNeT++. A first tool – called *RENETO Trace Analyzer* – automates the analysis of traffic captures, in order to record the statistical distribution of various parameters of the model. In order to use our model in simulations, we introduce a second tool – called *RENETO Traffic Generator* – which reproduces those different layers in OMNeT++/INET, and generates flows which follow the statistical distributions recorded from the original captures by the first tool. The interaction between this suite of tools is presented in Figure 9.1.

While similar work often characterize and generate the different parameters of their under-

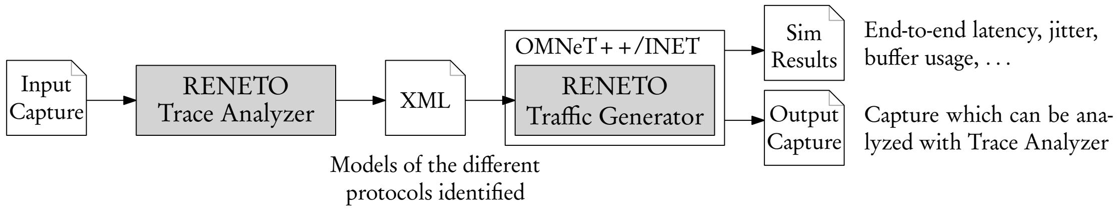

Input Capture
RENETO Trace Analyzer
XML
Models of the different protocols identified
OMNeT++/INET
RENETO Traffic Generator
Sim Results
End-to-end latency, jitter, buffer usage, ...
Output Capture
Capture which can be analyzed with Trace Analyzer

Fig. 9.1: RENETO toolchain, from network capture to simulation

lying model as statistically independent variables, one contribution of this work is to propose a model which adds correlation between some parameters of the model. Such technique is useful to capture patterns which can be seen in different protocols with bursty behavior. Our model was designed to work with any protocol on top of UDP or TCP and may be extended to other protocols in the future. By using a general model, our goal is to map realistic traffic behavior of UDP and TCP based protocols without having the need to know or implement the underlying protocol mechanisms.

The first aim of our model and tool is to mathematically characterize and simulate realistic traffic loads to serve as input for studying and dimensioning Ethernet networks. Although it was not designed for applications such as anomaly or protocol detection and categorization, our underlying mathematical model may also be used for a wider range of higher level studies.

Structure of this chapter

In Section 9.2, we present similar research studies. Section 9.3 details the model, its parameters, and how to extract values from a network capture. With Section 9.4, we present the design of the traffic generator developed for OMNeT++. In Section 9.5, we evaluate the different parts of the toolchain, and compare measures on original captures with measures on synthetic traffic. Finally, Section 9.6 summarizes and concludes this chapter.

9.2 Related work

Various works have been proposed in the literature regarding realistic traffic modeling, either using some mathematical formalism or via simulation, as it is a key tool for effective network engineering.

Regarding works on mathematical models, queuing theory with a focus at the packet level is generally a popular approach to the problem of characterizing network traffic. One of the simplest models for this purpose is the compound Poisson process, where the inter-arrival time between two packets follows an exponential distribution and the packet size follows a general distribution. While the mathematical properties of this approach are attractive, it has been shown – for instance by Leland *et al.* in [171], or by Paxson and Floyd in [199] – that this approach does not really match the properties of Ethernet traffic. An alternative to the compound Poisson process is to use more advanced arrival processes, such as Markov Arrival Processes (MAP), as proposed for instance by Andersen and Nielsen in [35]. Such approach has been extended recently by Casale *et al.* in [79] for instance. As it was shown in [35, 79], results using such methods are better suited at presenting the packet level behavior of

Ethernet traffic. But we note that the works previously cited are focused on packets behavior, which does not necessarily apply to elastic protocols where packet level behavior of flows varies according to network conditions as well as which variant of the elastic protocol is used. Namely, we may assume that the behavior of a user is uncorrelated to which TCP variant is used, although the behavior of two different TCP variant might seem different from a packet layer perspective.

Regarding discrete event simulation, generating realistic is generally challenging due to the need of a good characterization of the traffic. The problem of characterizing, modeling and generating network load in simulations is a core challenge when doing performance evaluations in simulators.

One method often used in simulators is to re-implement the complete protocol stack of the studied protocols, which leads to high accuracy of the model, but also a high cost of implementation and low flexibility. Work has already been performed on this problem on different simulators, but with focus on a certain type of traffic. For HTTP traffic, Barford and Crovella proposed SURGE in [49] for generating web traffic on standard Linux computer. Likewise, Jónsson proposed HttpTools in [155] for generating web traffic in OMNeT++. Bohge and Renwanz proposed in [63] a module for OMNeT++ to generate synthetic multimedia flows for VoIP. Similar tools are available for *ns-2*, such as PackMime [76] from Cao *et al.* for HTTP traffic, or ns2voip [42] from Bacioccola *et al.* for VoIP.

While such tools are useful for evaluating specific use cases, research was also done on more generic tools, working with wider range of applications. For instance tools such as Swing [235] from Vishwanath and Vahdat or Tmix [136] from Hernandez-Campos *et al.* are able to analyze network capture with TCP-based protocols and generate traffic accordingly. Using an analogous model than the one developed here, Harpoon was proposed by Sommers and Barford in [225]. Compared to our work, Harpoon does not take into account user behavior, is mostly focused on TCP-based protocols, and considers flows to be unidirectional.

Finally, the idea of correlating parameters of the model was already proposed by Rolland *et al.* for the extended version of LiTGen in [209], which focused on TCP traffic. The result of their study was that correlating the packet size and the inter-arrival time of packets was beneficial to synthesize realistic traffic. We use this concept for our own work and extend it to UDP traffic.

9.3 Traffic model

We present in this section the mathematical model which was developed for RENETO, as well as the tool *RENETO Trace Analyzer*, which extract such model based on a network capture.

9.3.1 General description

A structural model was developed for RENETO taking into account the different layers of the protocol stack. The structure presented here is similar to the ones presented by Ricciato *et al.* in [205], Rolland *et al.* in [209], or Vishwanath and Vahdat in [235], where individual packets are grouped into sessions and flows.

Our model is based on the analysis of the layer 4 payload of the packets. This means that for TCP based protocols, we will focus on logical messages, and not the underlying mechanisms of TCP flow control algorithms. The task of modeling TCP is delegated either to the underlying mathematical model characterizing TCP behavior, or to the TCP models provided by the discrete event simulator used.

The four following layers are used in RENETO:

Message layer We distinguish here two cases, depending on the layer 4 protocol of the studied application.

In case of UDP traffic, we use an approach similar to a renewal-reward process. We characterize the layer 4 payload size distribution of each frame in each way for a bidirectional communication, with the variable *reqSize* for requests, and *respSize* for responses. We characterize the distribution of inter-arrival time between packets with the variables I_{req} for requests, and I_{resp} for responses in case the connection uses more than one request/response exchange.

In case of TCP based flows, the notion of individual packet size (or Ethernet frame size) is not present in our model, as it highly depends on the version of TCP used, as well as the conditions of the network on which the trace was made. Regarding packet size, we do not look at individual packets like for UDP, but we group TCP segments together to form a layer 7 message or *object*. The length of this layer 7 messages will define the *reqSize* and *respSize* variables. We also characterize the think time after reception of a message, with the variable I_{req} for the client side, and I_{resp} for the server side. We note that this approach for TCP based flows is similar to the bidirectional ON/OFF traffic model proposed in Section 8.5.3. For a graphical illustration of this model, we refer to Figure 8.5.

Flow layer Because our approach is focused on TCP and UDP traffic, we define a flow as a sequence of packets or messages having the following unique parameters for a certain time: source and destination IP address, layer 4 protocol (TCP or UDP), source and destination layer 4 ports. By permuting source and destination, we capture both directions of a flow (when applicable). This definition fits both cases of streaming, where a server sends packets without expecting any reply or acknowledgments, and requests-responses between a client and server.

We characterize the distribution of number of request-responses pairs occurring in a flow (for bi-directional flows), with the variable N_{pairs}. We record also in this layer, the transport layer (TCP or UDP) which was used.

Session layer We define a session as a group of flows occurring around the same active period initiated by the same source. Between those sessions are inactive periods. This is similar to the notion of RRE (Request/Response Exchanges) proposed by Vishwanath and Vahdat in for Swing [235] and also described as an important part of the model from Ricciato *et al.* [205]. We characterize the distribution of the number of flows per session with the variable N_{flow}, as well as the distribution of time between the start of two flows with the variable I_{flow}.

User layer Finally, we define a user as the agent starting the different sessions. Similar to the session layer, we characterize the distribution of the number of sessions initiated by a user with the variable $N_{session}$ and the distribution of time between the start of two sessions with the variable $I_{session}$.

We identify single users by their IP addresses. While this definition can be accepted for private networks, it does not fit completely with the Internet of today, where multiple users can be behind a single public IP address. When making a network capture and analyzing it, such point has to be kept in mind.

A summary of the different variables used in the model and their description is given in Table 9.1. The parameters I_{req} and *reqSize* are also recorded using a bivariate distribution, as well as the parameters I_{resp} and *respSize*. We did not use the correlation of other parameters at this step of our work.

Layer	Variable	Signification
UDP Message	*reqSize*	Packet size of requests
	respSize	Packet size of responses
	I_{req}	Inter-arrival time of requests
	I_{resp}	Inter-arrival time of responses
TCP Message	*reqSize*	Message size of requests
	respSize	Message size of responses
	I_{req}	Think time before request
	I_{resp}	Think time before response
Flow	N_{pair}	Number of request-responses pairs
Session	N_{flow}	Number of flows
	I_{flow}	Time between start of flows
User	$N_{session}$	Number of sessions
	$I_{session}$	Time between start of sessions

Tab. 9.1: Summary of the RENETO model notation

In order to have a quick overview of how the structural model works, we take here the example of a user browsing a website (using HTTP), as illustrated in Figure 9.2. Events in the user layer of our model will typically correspond to user actions such as clicks on links. We record it in the user layer: number of events (such as a click) and time between events (such as time to read a page). When the browser loads web pages, different requests are made around the same time to first fetch the HTML page, and then other resources present on the page, such as images or scripts. The session layer records those requests: number and time between requests. As each request is an individual flow, we record the number of exchanges, which in case of HTTP is 1 or more, as well as the time between those exchanges (processing time of the response). Finally, we look at the message size (as HTTP generally works on top of TCP): less than 10 kbit for an HTTP GET request, and around 500 kbit in case the server answers with a small image (which would fit into multiple TCP segments).

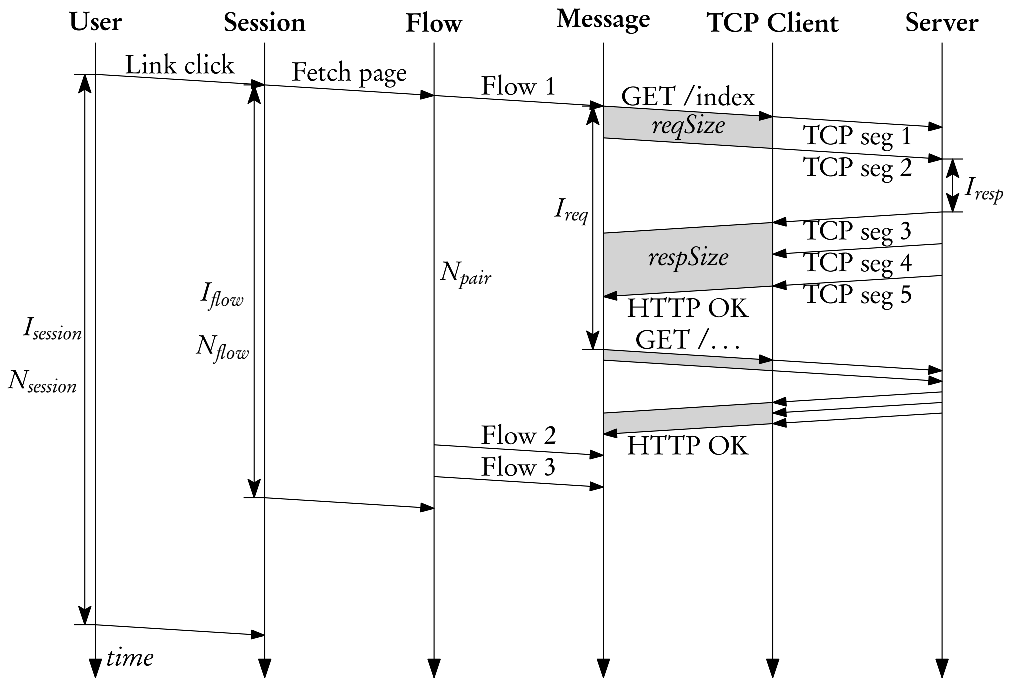


Fig. 9.2: Illustration of the RENETO model with HTTP

9.3.2 Trace analysis and parameters

In order to produce realistic traffic, we use publicly available captures or network captures made on testbeds. Our approach focuses on the analysis of full packet captures for a given link, typically using the PCAP file format, giving us the ability to investigate the application layer. This provides us with more information on a flow than other capture methods such as NetFlow.

The first step for treating the capture is to assign each packet to a traffic flow. For UDP and TCP traffic, a first base is to use IP addresses, port numbers of the transport layer, as well as the timestamp of the packets. In case of TCP, the use of the TCP flags and sequence number is also taken into account, and the end of a flow can be clearly defined. This step is delegated to the library `libflowmanager` from WAND Network Research Group [238], which also handles reordering of out-of-order TCP packets. One difficulty of bi-directional flows characterization is to determine the server and the client side in case the capture does not contain all the packets of the flows. This was solved by some heuristic function based on TCP flags and use of ephemeral port numbers which works in most cases.

The second step is to assign each flow to a certain application class. This means determining which application layer protocol is used for each flow. Because automatic classification and deep packet inspection is not the focus of this work, we based our tool on the existing packet inspection library `libprotoident` from WAND Network Research Group [239]. This library bases its approach on the analysis of the first four bytes of the packet payload (application layer) observed in each direction, as well as the packet size. The library is advertised to support more than 200 application protocols based on TCP or UDP. In case a protocol is not identified, we use the destination port number and the transport layer name to classify applications as a fallback scheme.

Once flows are classified, the last step is to build and populate for each application the RENETO structural model previously presented. For the first version of our tool, we decided to use the Empirical Cumulative Distribution Function (noted hereafter Empirical CDF or ECDF) for representing each parameter previously characterized. The empirical cumulative distribution function $F_X^n(t)$ of the $(x_1, \ldots, x_n)$ variates of X is defined as:

$$F_X^n(t) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{x_i \le t\} \tag{9.1}$$

with $\mathbb{I}\{x\}$ the indicator function, which returns 1 if x is true and 0 otherwise.

When the analysis is finished, each parameter of the model is then saved in an XML file. We approximate the inverse of the ECDF by evaluating it at equi-distributed points according to Algorithm 9.1. We define *expValues* as an array storing the experimental values, and N_S the number of intervals we wish to have. N_S will determine the precision of the generated number compared to the original distribution. The default value of N_S in the tool is 100. We study in Section 9.5 the impact of N_S on the accuracy of the generated numbers.

We choose to record the inverse of the ECDF because of the method we used for the random-number generation described in Section 9.4.1, which involves the inverse transform sampling method.

Algorithm 9.1 Inverse empirical cumulative distribution function

```
1: function buildAndSampleIECDF(expValues, $N_S$)
   ▷ Applies only if $N_S \le$ length(expValues)
2:     iecdf ← newArray()
3:     sort(expValues)
4:     n ← length(expValues)
5:     iecdf.insert(expValues[0])
6:     count ← 1
7:     $i \leftarrow 1$
8:     while $i \le n$ do
9:         $a \leftarrow \lfloor (i \cdot N_S)/n \rfloor$
10:        if $a = count$ then
11:            iecdf.insert(expValues[$i$])
12:            count ← count + 1
13:        end if
14:        $i \leftarrow i + 1$
15:    end while
16:    iecdf.insert(expValues[$n - 1$])
17:    return iecdf
18: end function
```

We described before that our model includes two correlated variables (packet size and inter-arrival time), which means that we need a bivariate CDF. We reduce this bivariate problem to two univariate problems by using the conditional distribution method. This is summarized in the following equation, where X_1 and X_2 are the two variables to record:

$$\begin{aligned} F^n_{X_1,X_2}(t) &= F^n_{X_1}(t) \cdot F^n_{X_2|X_1}(t) \\ &= F^n_{X_2}(t) \cdot F^n_{X_1|X_2}(t) \end{aligned} \tag{9.2}$$

By using such decomposition, we are able to record the first parameter (X_1) using the function described in Algorithm 9.1, and then we can record the second parameter using multiple ECDF indexed by the values of X_1. We also perform the same decomposition by using X_2 as the first parameter.

While this method for storing and generating correlated random number proved to be working well in our tests with limited values of X_1, it is a point of improvement for future versions of RENETO.

At this point of our research we consider the parameters of our model to be stationary. In other words, this means the method detailed here is not well suited for live capture where the parameters may vary between the different times of the day.

Using the algorithms described previously, our tool produces a inter-exchange file as presented in Appendix C.

9.4 Traffic generation in discrete event simulations

We describe in this section the modules which were developed for generating traffic in OMNeT++ as well as some underlying points for generating traffic.

9.4.1 Pseudo-random number generation

In Monte Carlo simulations, random number generation is a core principle. Our goal here is to generate pseudo-random numbers following the same statistical distribution than the recorded parameters by RENETO Trace Analyzer. Some key requirements for this pseudo-random number generator are: produce values which pass tests for randomness, be based on a seed in order to be able to reproduce the experiments, and have a low complexity as it will be used often in the context of simulation.

As stated before, each parameter is recorded using the inverse of the empirical cumulative distribution (noted hereafter IECDF). One advantage of representing the distributions as cumulative distribution function is the possibility to use the inverse transform sampling method for generating random numbers following the same initial distribution as shown by Devroye in [100, Section 2.2]. Figure 9.3 shows the basic principle of the inverse transform sampling method: we generate a random number u from a standard uniform distribution in the interval $[0, 1]$, and we find the value x_{gen} such that $ecdf(x_{gen}) = u$. The inverse transform sampling method transforms a uniform random number generator into a non-uniform random number generator.

The major drawback of this method for generating non-uniform random variates is that we need to know the inverse of the ECDF. In our case, we sampled the inverse of the ECDF at several equidistant points as explained in Algorithm 9.1. We then use one of the algorithms

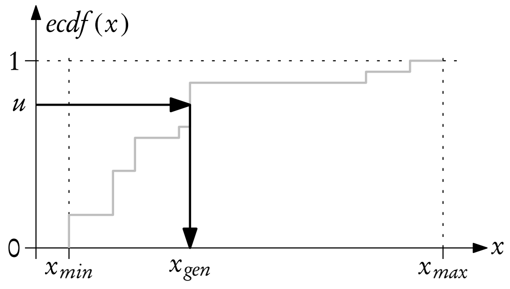


Fig. 9.3: Inverse transform sampling method applied to a discrete distribution

described in Algorithms 9.2 and 9.3 to generate pseudo-random numbers. In effect, the algorithm only accesses the IECDF table and returns the retrieved value, with piecewise constant interpolation by default (Algorithm 9.2), or linear interpolation if needed (Algorithm 9.3), making it an efficient algorithm.

Algorithm 9.2 Inverse transform sampling algorithm with piecewise constant interpolation

```
function getVariate(iecdf)
    $u \leftarrow$ randomUniformNumber(0, 1)
    $i \leftarrow \lfloor u \cdot N_S \rfloor$
    return $iecdf[i]$
end function
```

Algorithm 9.3 Inverse transform sampling algorithm with a linear interpolation

```
function getVariate(iecdf)
    $u \leftarrow$ randomUniformNumber(0, 1)
    $i \leftarrow \lfloor u \cdot N_S \rfloor$
    if $i = N_S$ then
        return $iecdf[i]$
    end if
    return $iecdf[i] + (iecdf[i+1] - iecdf[i]) \cdot u \cdot (N_S - i)$
end function
```

We will detail in Section 9.5.2 if other interpolation methods of the inverse of the ECDF would be more beneficial to our generator than the method we used here.

We used the pseudo-random generator provided by OMNeT++ as our source for uniformly distributed random numbers, which by default is the commonly used MT19937 uniform pseudo-random number generator by Matsumoto and Nishimura [186]. This pseudo-random number generator has a period of $2^{19\,937} - 1 \approx 4 \times 10^{6001}$ and is as fast or faster than the `rand` function from ANSI C, which makes it suitable for Monte Carlo simulations. We used the default seeds provided by OMNeT++ for this generator.

9.4.2 Packet generation process

In order to generate packets, we follow the same process defined for the trace analysis in Section 9.3.2, but in the reverse order. More precisely for the client side, we start by simulating the user layer, which instantiates sessions, and the session layer then instantiates the flows. This process is presented in Algorithm 9.4. The function getVariate(model.X) corresponds here to the generation of a sample of the parameter X of the model, using the method described in Section 9.4.1.

Algorithm 9.4 Simulation of user and session layers

```
    User layer
 1: function simulateUser($model$)
 2:     $numSession \leftarrow$ getVariate($model.N_{session}$)
 3:     $t \leftarrow$ getCurrentSimulationTime()
 4:     while $numSession > 0$ do
 5:         At time $t$ call simulateSession($model$)
 6:         $t \leftarrow t +$ getVariate($model.I_{session}$)
 7:         $numSession \leftarrow numSession - 1$
 8:     end while
 9: end function

    Session layer
10: function simulateSession($model$)
11:     numFlow $\leftarrow$ getVariate($model.N_{flow}$)
12:     $t \leftarrow$ getCurrentSimulationTime()
13:     while numFlow $> 0$ do
14:         At time $t$ call simulateFlow($model$)
15:         $t \leftarrow t +$ getVariate($model.I_{flow}$)
16:         $numFlow \leftarrow numFlow - 1$
17:     end while
18: end function
```

The last step is to generate the packets. This task is handled by the flow and packet layers, which send the packets to the UDP layer, respectively TCP layer, as presented in Algorithm 9.5, respectively Algorithm 9.6.

We presented in Algorithm 9.5, respectively Algorithm 9.6, the version of our algorithm with the correlation between packet size and inter-arrival time as shown on line 7, respectively line 9.

In our model definition, the server side corresponds to the entity replying to requests from clients. In other words, the server waits for request messages and responds with reply messages. This means that only the packet and flow layers are simulated on the server side, with algorithms similar to the ones presented for the client side.

Algorithm 9.5 Simulation of flow and packet layers for UDP

```
1: function SIMULATEFLOW($model$)
2:     $numPairs \leftarrow$ GETVARIATE($model.N_{pairs}$)
3:     $t \leftarrow$ GETCURRENTSIMULATIONTIME()
4:     while $numPairs > 0$ do
5:         $packetSize \leftarrow$ GETVARIATE($model.reqSize$)
6:         At time $t$ emit packet with size $packetSize$
7:         $t \leftarrow t +$ GETVARIATE($model.I_{req}|packetSize$)
8:         $numPairs \leftarrow numPairs - 1$
9:     end while
10: end function
```

Algorithm 9.6 Simulation of flow and packet layers for TCP

```
1: function SIMULATEFLOW($model$)
2:     $numPairs \leftarrow$ GETVARIATE($model.N_{pairs}$)
3:     while $numPairs > 0$ do
4:         $packetSize \leftarrow$ GETVARIATE($model.reqSize$)
5:         Emit packet with size $packetSize$
6:         Wait for reply from the server
7:         $numPairs \leftarrow numPairs - 1$
8:         if $numPairs > 0$ then
9:             Wait for GETVARIATE($model.I_{req}|packetSize$)
10:        end if
11:    end while
12: end function
```

9.4.3 Application to OMNeT++ modules

The discrete-event simulator used in this chapter is OMNeT++ [17], and its framework INET [14] which contains models for several protocols commonly found in a network. For our target application we focus on the following models provided by INET: Ethernet, IP, ARP, UDP and TCP.

One client module and one server module is available for TCP, respectively UDP. They implement the `ITCPApp` module interface, respectively `IUDPApp` module interface, defined in INET, making them compatible with the TCP, respectively UDP, stack of INET.

An example of such model is presented on Figure 9.4, where we used the `StandardHost` model and attached TCP and UDP applications.

9.5 Numerical evaluation

We evaluated our tool on two common public traces for reproducibility and comparison purposes. We also used one of our own traces to demonstrate the capabilities of our tool with UDP traffic, which correspond to an industrial use case.

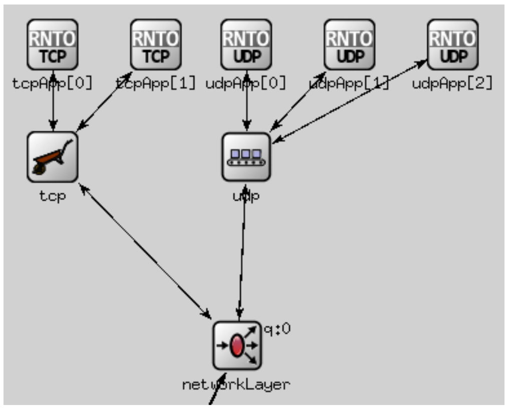


Fig. 9.4: Screenshot of the OMNeT++ `StandardHost` model with 2 TCP and 3 UDP RENETO applications

We used the LBNL-FTP-PKT trace [196] as a first benchmark for our tool. This trace is composed of more than 3 million anonymized packets of FTP control request and responses (no data transfer). Complete packets are available such that the packet inspection and protocol classification library can be used.

The second trace which was used is the 3-day ACM SIGCOMM'01 conference capture [45] composed of more than 16 million packets. In this capture, packets were truncated to record only up to the transport layer and exclude all application layers for anonymization purpose. For this case, packet classification is based on port numbers.

Finally, the last trace used for this evaluation is a trace with SNMP (Simple Network Management Protocol) traffic made on our own testbed. It is composed of 1426 packets.

9.5.1 RENETO Trace Analyzer

Packet classification

As we know exactly what is in the LBNL-FTP-PKT trace (FTP control packets), we are able to evaluate if the protocol identification library chosen for this tool is efficient and correct. Table 9.2 gives the result of the tool for the LBNL-FTP-PKT trace. As we can see less than 1 % of the packets were misclassified. Some packets with a server port of 21 could not be identified and others were misclassified as SMTP (Simple Mail Transfer Protocol), because FTP and SMTP share some commands and reply codes. A misclassified flow can be linked to a flow which has been cut due to the start or the end of the capture in the middle of the flow.

Protocol	Number of packets		Expected
FTP Control	3 231 941	(99.0 %)	100 %
TCP Port 21	27 462	(0.84 %)	0 %
SMTP	4549	(0.14 %)	0 %

Tab. 9.2: Packet classification result of LBNL-FTP-PKT trace

For the second trace, as packets were truncated to remove all layers above layer 4, no protocol identification could be used. Instead, we used the server port number as a simple heuristic for packet classification. 2227 different server ports were reported by our tool.

For our last trace, SNMP was correctly identified for all the packets by the protocol identification library.

Execution time

The CPU execution time was measured on a computer equipped with an Intel Core2Duo E8400 clocked at 3 GHz. Results are summarized in Table 9.3. We conclude that the execution time is almost linear with the number of packets, with around 10 μs spent per packet.

Trace	Number of packets	Execution time
SNMP	1426	0.05 s
LBNL-FTP-PKT [196]	3 264 050	34.5 s
SIGCOMM'01 [45]	16 329 537	195.3 s

Tab. 9.3: CPU execution time of RENETO Trace Analyzer

9.5.2 Pseudo random number generation

As the pseudo-random number generator is a core element for the generation of network traffic, we evaluate it here and also look at the number of intervals N_S needed to achieve a good precision.

Continuous distributions

We use the u-error as recommended by Hörmann and Leydold in [142] for evaluating our pseudo random number generator. It is defined as:

$$\epsilon_u(u) = |u - F(G^{-1}(u))| \tag{9.3}$$

for $u \in [0, 1]$, with F the cumulative distribution function, and G^{-1} the approximation of the inverse cumulative distribution function. We can see that if $G^{-1} = F^{-1}$, we will get an error $\epsilon_u = 0, \forall u \in [0, 1]$.

We used the math toolkit of the `boost` C++ library[1] for performing our tests, which provides a direct access to the CDF and the inverse CDF of a wide range of well-known distributions[2]. We first evaluated the precision of the CDF and ICDF functions of the `boost` library by using the u-error previously described in Equation (9.3), with $G^{-1} = F^{-1}$. Results showed an u-error below 10^{-15} for the tested distributions, meaning a good precision for further evaluations.

[1] `http://www.boost.org/libs/math/`

[2] For the complete list of available distributions, see: `http://www.boost.org/libs/math/doc/html/math_toolkit/dist_ref/dists.html`

We then evaluated our method for generating pseudo-random numbers based on a simple linear interpolation. We directly filled the internal IECDF table of our generator with evaluated values of the exact ICDF values provided by the library. We can then evaluate how many samples are necessary in order to reach a certain maximal u-error. We evaluated the u-error in 1000 different points chosen randomly according to a uniform distribution and stored the maximal value of u-error. Results are presented in Figure 9.5 in case of a normal distribution. Results for other distributions are similar.

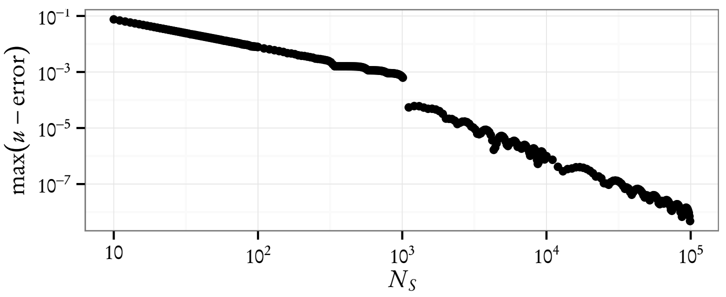


Fig. 9.5: Number of samples N_S needed for achieving a maximal u-error using an exact ICDF

We then evaluated the complete toolchain of our pseudo-random number generator, namely generate observations from a known distribution, store them in our generator, and generating pseudo-random numbers based on those observations. The observations are generated using the inverse transform method with the exact ICDF provided by the library boost.

We evaluate the u-error in 1000 uniformly distributed points in the u-scale, as presented in Figure 9.6, with a number of sample N_S of 50 and 250. A normal distribution was used for generating those plots. As we can see on the figure, with a small number of sample N_S the tails of the distribution, where u is near 0 and 1, have a larger error than the middle, due to the linear interpolation which does not fit the tails of a normal distribution. Such pitfall is to be expected because we sampled the IECDF curve with a fixed interval size.

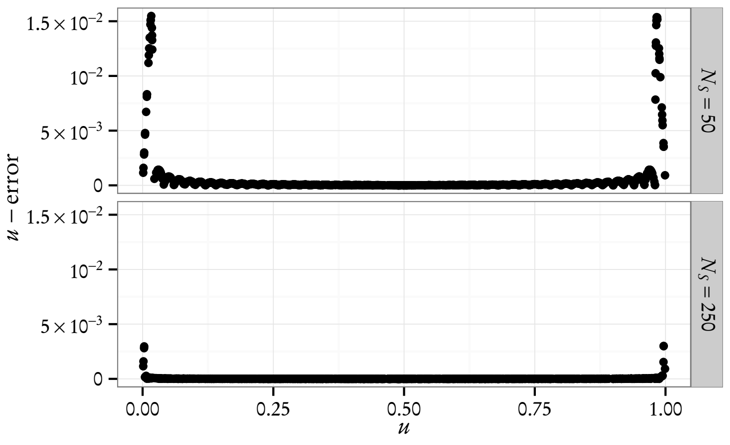


Fig. 9.6: u-error for N_S = 50 (top) and N_S = 250 (bottom)

Finally, we evaluated the number of sample N_S needed for reaching a maximal u-error as presented in Figure 9.7. We can see that we reach a similar accuracy than for Figure 9.5 for N_S < 500 and we reach a minimal achievable error of 10^{-3} for the $N_S \geq 500$.

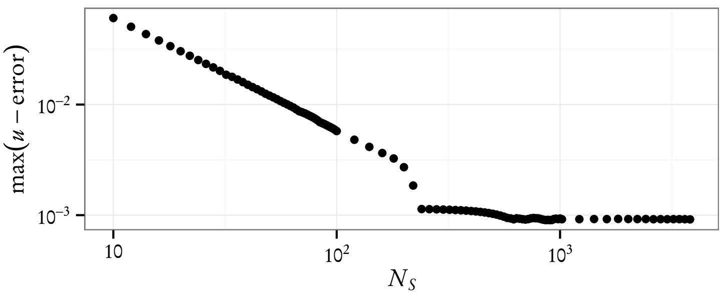


Fig. 9.7: Number of samples N_S needed for achieving a maximal u-error using observations of the distribution

Discrete distributions

Because of the discontinuities in the CDF of discrete distributions, the u-error previously used needs to be modified for measuring the error of discrete distributions. We modify the u-error and introduce here the ud-error ϵ_{ud} which applies to discrete distribution with the following equation:

$$\epsilon_{ud} = |F(F^{-1}(u)) - F(G^{-1}(u))| \tag{9.4}$$

We evaluated the ud-error with a binomial distribution. Results are presented on Figure 9.8, where we can see that errors happen around the discontinuity points, while for the rest the error is exactly 0. This result can be explained by the fact that we used here regularly spaced intervals to sample the ECDF, and not variable intervals which could match the discontinuity points.

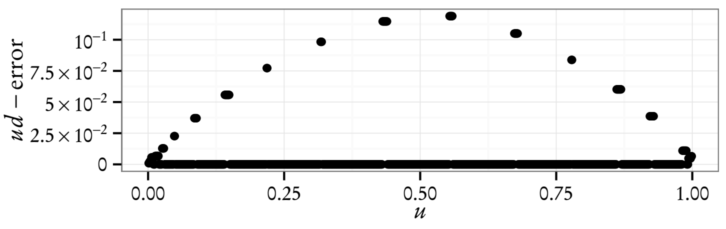


Fig. 9.8: ud-error for N_S = 100 with a binomial distribution

Finally, we made a comparison between different interpolation methods to see if the method we chose was relevant enough, using a two-sample location test of the null hypothesis that the means of two populations are equal, as presented by Law and Kelton in [167, Chap. 10], and detailed in Equation (3.5). This method was detailed earlier in Section 3.5.1. It is similar to the evaluation of the absolute x-error ϵ_x presented by Hörmann and Leydold in [142] and defined as:

$$\epsilon_x = |F^{-1}(u) - G^{-1}(u)| \tag{9.5}$$

We used here the distribution of request message size from the LNBL-FTP-PKT trace, which can be seen in Figure 9.10, and the interpolation methods provided by GNU-R [18], a programming language and software environment for statistical computing. Like for the u-error, we evaluated the generated number from the inverse transform sampling method in

1000 uniformly distributed points in the u-scale, using 3 different size of samples N_S. Results are presented on Figure 9.9.

The use of a piecewise constant interpolation brings better results than other interpolation method as the difference is around 0 for the evaluated N_S values. This result confirms our choice of a piecewise constant interpolation for discrete distributions.

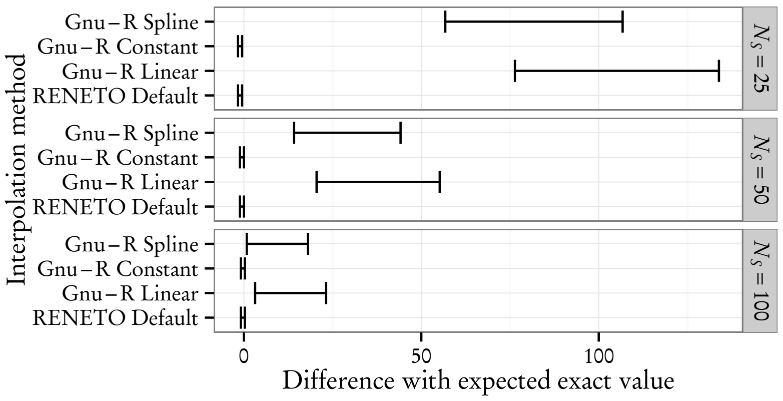


Fig. 9.9: Comparison of interpolation function for a discrete distribution according to the method presented in Equation (3.5)

9.5.3 Traffic generation

For this evaluation, we generate traffic between a certain number of clients, and one server. Each client is directly connected to the server with an 100 Mbit/s Ethernet connection.

Generation of FTP Control traffic

For this part, we generated traffic according to the FTP Control model extracted from the LNBL-FTP-PKT trace and 50 clients were simulated.

Figure 9.10 presents the different parameters of the model using the Empirical Cumulative Distribution Function as a representation. On the same plots, we have the model of the original capture (plain line), as well as the model of the simulation (dashed line).

With a first visual estimation on Figure 9.10, we see that the simulation reproduces accurately the parameters recorded by the analysis tool. In order to give a more quantitative comparison of the various distributions presented in Figure 9.10, we performed a two sample Kolmogorov-Smirnov test for each parameter (see [93, pages 309–314]). This statistical test is a two-sided test for the null hypothesis that the two cumulative distribution functions are drawn from the same continuous distribution. This test quantifies the distance between two cumulative distribution functions.

Results are presented in Figure 9.11 with different simulation runs, meaning that the pseudo-random number generator has a different seed at each run. We can see from the p value and with a significance level of $\alpha = 0.05$, that the difference between the model and the simulation is not significant enough to say that they have a different distribution, except for

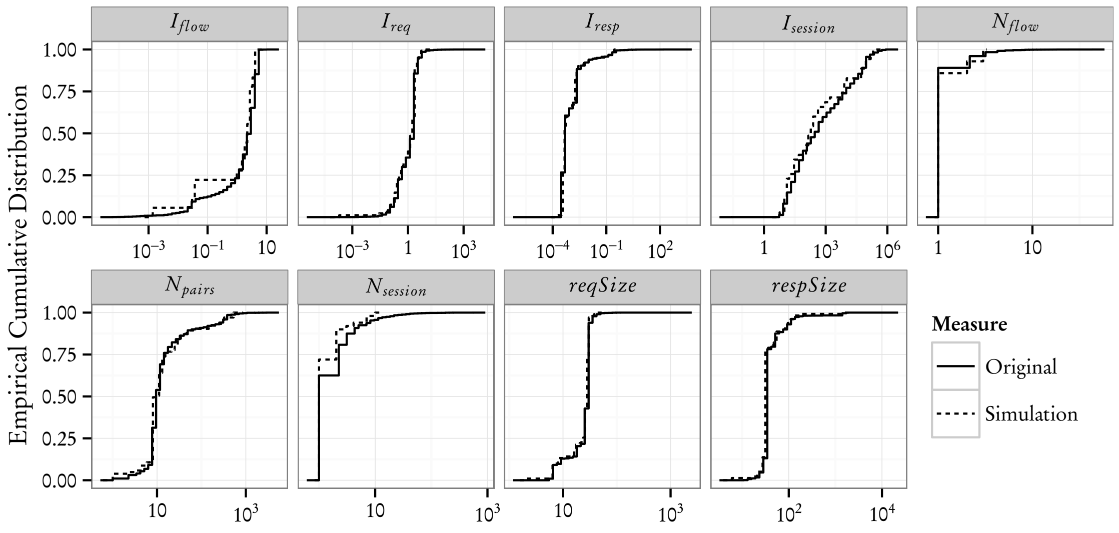


Fig. 9.10: Empirical Cumulative Distributions of the different parameters of the model after analysis of the LNBL-FTP-PKT trace, and simulation based on the model. Times are given in seconds, and message sizes in Bytes

the I_{resp} and I_{req} parameters which are highly dependent on where the trace is made on the network as our model does not account the topology of the original trace.

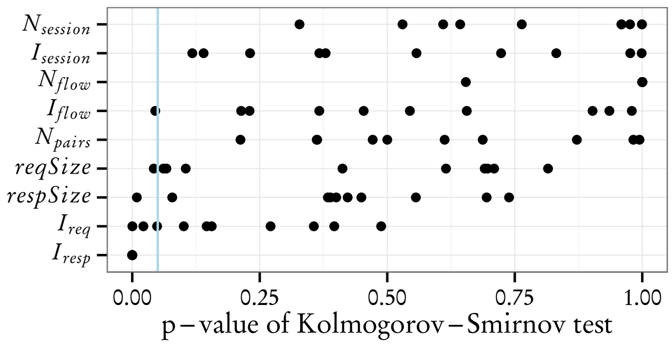


Fig. 9.11: Result of a two sample Kolmogorov-Smirnov test between original trace and simulation with 10 simulation runs

Generation of SNMP traffic

For this part, we generated SNMP traffic according to the SNMP model extracted from our own trace. This trace has the advantage of containing some correlation between packet size and inter-arrival time. We show on Figure 9.12 this correlation between packet size and inter-arrival time, where we see the benefit of this introduced correlation for the simulation. Generating packets in the lower-right part of the plot (large packets with a small inter-arrival time) would generate more bandwidth utilization than in the original capture.

Finally, we also investigate the ability of our model to reproduce long-range dependency (LRD) or self-similarity found in the initial trace as Ethernet traffic is well known to be self-

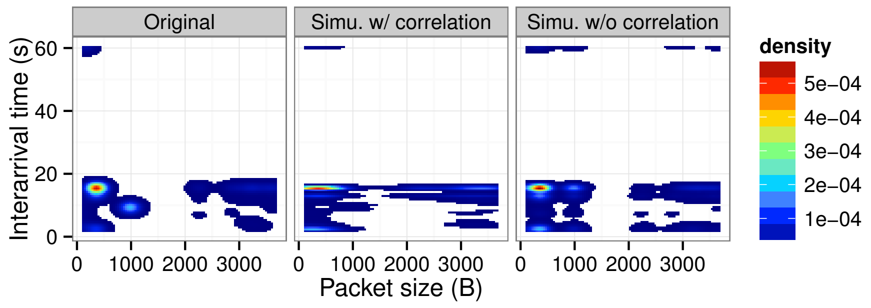


Fig. 9.12: Correlation between inter-arrival time and packet size for the SNMP trace

similar (see for instance the work from Leland *et al.* in [171]). Self-similarity describes the fact that traffic "spikes" look similar at different time scales.

As proposed by the author of Swing in [235], we use here the Logscale Diagram Estimate (or LDE) introduced by Abry *et al.* in [19]. This method is based on discrete wavelet transform. In simple terms, this diagram shows the variance in the traffic arrival process at different timescales. This diagram and associated tool was used in order to match the packet arrival time series and compare the original capture with and the simulated traffic.

As suggested by Abry *et al.* in [19], we can compute the scaling exponent α of the LRD process by performing a linear regression on the curve. The analysis of the original traffic resulted in $\alpha = 2.25$, which suggests self-similar traffic with a Hurst parameter $H = (\alpha - 1)/2 = 0.625$, as α is greater than 1 following the numerical evaluations proposed in [19].

The LDE of the capture and the 10 runs of the simulation is presented on Figure 9.13. Each simulation run corresponds to a different seed for the pseudo-random number generator. We can see from the plot that the seed has a limited influence on the variance.

As exposed in the Figure 9.12, we separated here the plots to see if the use of the correlation between parameters improves the quality of the simulated traffic regarding self-similarity. We can see that the use of the correlation brings more precision to the simulation than without, although the difference is small.

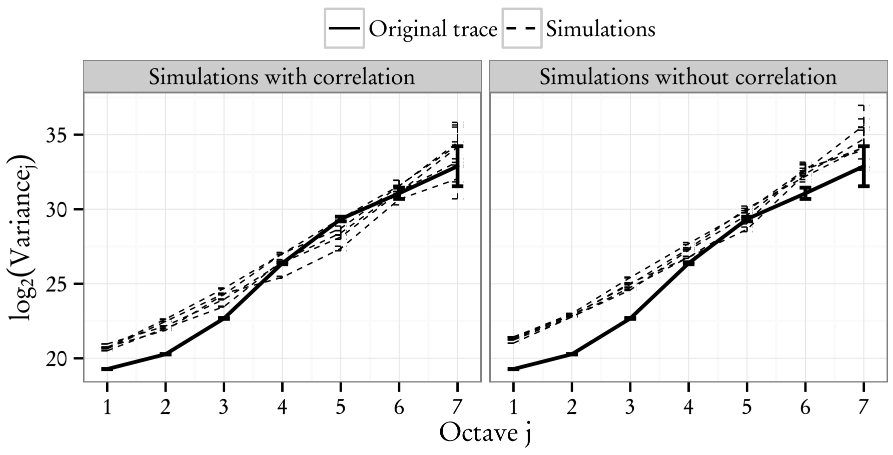


Fig. 9.13: Logscale Diagram Estimate of the original traffic and the generated traffic in case of the SNMP capture

9.6 Conclusion on realistic user and application modeling

We developed in this chapter a simple mathematical model able to characterize the realistic behavior of higher-layer protocols and users (Research Objective **O2.5**). Our model is based on four layers akin to the OSI layers, namely: *(i) messages* corresponding to layer 4 (TCP or UDP) messages; *(ii) flows* which are groups of messages exchanged between two TCP/IP endpoints (*i.e.* network sockets) around the same time period; *(iii) sessions* corresponding to a group of flows ocurring around the same time and initiated by the same source; *(iv) users* which reflect humans (or digital agents) starting the different sessions. Each layer of the model is characterized by various random variables. One contribution of our work is to consider some of the random variables as correlated in order to better reflect characteristics of network protocols.

In order to give realistic statistical distributions to the random variables describing our model, we proposed to extract empirical distributions from traffic captures. This process has been automated via our tool called RENETO Trace Analyzer. In order to validate if our four layer model is indeed able to reproduce realistic traffic behavior, we used discrete-event simulation. We developed algorithms able to replicate the different layers of the protocol. Those algorithms were then implemented in a second tool called RENETO Trace Generator which benefits from the output produced by our network capture analyzer.

Our method and tools were evaluated against two different network protocols: FTP Control for a TCP-based protocol and SNMP for a UDP-based one. We first evaluated our internal pseudo-random number generator implementation in order to see if the generated statistical distributions matched the input distribution. We then showed that the generated traffic was comparable to the traffic of the original capture using two different approaches. The first approach was to extract the protocol model from the synthetic trace and compare the extracted statistical distributions to the ones of the input protocol model. The second approach was to compare the long-range dependency of both input and synthetic traces using the Logscale Diagram Estimate, a popular method to evaluate the long-range dependency or self-similarity of network traces. Using those two approaches, we showed the benefit of the correlation between packet size and inter-arrival time on UDP traffic.

We note that the message layer of the four layer model presented in this chapter is actually similar to the analytical ON/OFF model proposed in Chapter 8. While the method developed in this chapter is mainly targeted at discrete-event simulation, an interesting extension of this work would be to add the flow, session and user layers to the ON/OFF model presented in Chapter 8 in order to incorporate this realistic traffic behavior into the analytical framework proposed in Chapters 7 and 8.

Key insights and contributions

Model of network protocol We proposed in this chapter a simple mathematical model able to characterize realistic network protocols and traffic.

Importance of correlation in the model One contribution was to show that some protocols have correlated parameters which have to be taken into account for increasing the accuracy of the method.

Tool for extracting the parameters from a trace We proposed a tool able to extract the statistical distributions of the different random variables of the model of specific protocols based on the analysis of a network capture.

Tool for reproducing the model in simulations We also proposed algorithms and their associated implementation for reproducing our protocol model inside a discrete-event simulator.

10. TCP TRANSFERS WITH STOCHASTIC GUARANTEES

10.1 Introduction

We have formalized and extended in Chapter 8 a framework for the performance evaluation of ON/OFF TCP and UDP flows sharing multiple bottlenecks on Ethernet networks, but with the limitation that we only evaluated mean performances. While such performance index is certainly useful for applications falling in the soft real-time category, it is not considered sufficient for firm real-time. Hence, we propose in this chapter to investigate stochastic bounds for elastic traffic by reusing the results from Chapters 7 and 8. The bounds we develop here are similar to ones presented earlier in Section 2.3.2, namely:

$$\Pr(\textit{delay} \geq \textit{bound}) \leq \epsilon \tag{10.1}$$

$$\Pr(\textit{queue size} \geq \textit{bound}) \leq \epsilon \tag{10.2}$$

$$\Pr(\textit{transfer time} \geq \textit{bound}) \leq \epsilon \tag{10.3}$$

with ϵ the probability that the bound is violated. ϵ is a small value generally between 10^{-3} and 10^{-9} as illustrated earlier in Table 2.3. Another formulation of Equations (10.1) to (10.3) is to say that a performance parameter (delay, queue size, etc.) will be below a required bound with a probability of $1 - \epsilon$ (*i.e.* 99.999 % or 99.999 999 % for instance).

One straightforward solution to give such stochastic bound would be to use Markov's inequality in conjunction with the results from Chapter 8, namely:

$$\Pr(X \geq \textit{bound}) \leq \frac{\mathbb{E}[X]}{\textit{bound}} \tag{10.4}$$

As the right-hand side of the inequality does only depend on mean performances, we can directly take the results from Chapter 8 to give stochastic bounds. Nevertheless, bounds based on Markov's inequality are generally loose and hence we propose here an alternate stochastic extension of Chapter 7 using stochastic network calculus.

While stochastic network calculus generally focuses on packet-level guarantees as presented in Section 2.3.2, we propose to use it at a flow-level. We make use of the General Processor Sharing (GPS) service discipline in order to emulate the bandwidth sharing of elastic traffic. The contributions of this chapter are threefold. First, we extend the work from Jiang *et al.* [154] in order to give tighter bounds on flows bounded by a generalized Stochastically Bounded Burstiness (gSBB) process traversing a GPS server. Then, we propose a bound for a hybrid scheduling combining priority and GPS scheduling as studied in Chapter 4, representing a combination between hard real-time traffic with low-latency and firm and soft elastic traffic as well as a bound for Packetized GPS (PGPS). Finally, we apply those results to elastic TCP traffic modeled using a compound Poisson process.

Structure of this chapter

In Section 10.2, we present related work. In Section 10.3, we first present the stochastic network calculus framework used in this chapter. Using this formulation, we develop a bound on flows traversing a GPS server in Section 10.4 and various extensions, namely PGPS and hybrid priority-GPS scheduling. We develop a simple stochastic flow model for ON/OFF TCP flows in Section 10.5. With Section 10.6, we compare the bounds developed here with a numerical evaluation. Finally, Section 10.7 summarizes and concludes this chapter.

10.2 Related work

The challenge of dimensioning a network for elastic traffic or TCP flows has been an active area of research since the late 1990's. While a large body of work on network dimensioning for TCP focuses on allocations for long-lived flows, work on short flows has received less attention. A closed-queuing network has been used by Berger and Kogan in [60] in order to give a statistical bound on the minimum available bandwidth. Similarly, Bonald *et al.* proposed a simple dimensioning rule on mean bandwidth requirements in [64]. Heyman *et al.* used Engset's model to predict the mean performances of ON/OFF TCP flows in [138]. More recently, Hoekstra *et al.* used the notion of effective service time in conjunction with a M/G/1 queuing model in [140] in order to derive the tail probability on the transfer time for TCP flows.

On the general principle of predicting the transfer time of TCP flows, Avrachenkov *et al.* proposed to use a two-level processor sharing scheduler in [36] in order to optimize the response time of short TCP flows. Chen and Vicat-Blanc Primet proposed to extend the IntServer/RSVP mechanism with specific time windows for bulk TCP transfers in [87].

A frequent model used for characterizing the bandwidth sharing of TCP is the processor sharing. Various asymptotic results on the tail of sojourn time in processor sharing have been published, such as for instance the works from Guillemin *et al.* in [129], Mandjes and Zwart in [179] or Boxma *et al.* in [67]. More generally on the tails of sojourn time in queuing networks, Boxma and Zwart proposed in [66] a survey with various scheduling disciplines. Finally, on the question of modeling the behavior of statistical bandwidth sharing, Ben Fredj *et al.* and Roberts presented surveys of various models based on queuing theory or various notions of fairness in [57] and [208].

Regarding service guarantees using the GPS scheduling discipline, an early work from Parekh and Gallager [197] introduced various notions and provided deterministic bounds on flows shaped using a leaky bucket. This work then served as a basis to Zhang *et al.* in [255] to give stochastic bounds on flows in an early formulation of stochastic network calculus. They proposed to model flows as exponentially bounded burstiness (EBB) stochastic processes, which was introduced by Yaron and Sidi in [247]. With the development of stochastic network calculus and its various traffic envelopes, GPS has subsequently been studied with other stochastic bounding processes. For instance, Weibull Bounded Burstiness (WBB) processes have been proposed and used by Yu *et al.* in [249] to characterize the performances of traffic which exhibit long range dependency (LRD). Example of LRD traffic are Ethernet as shown by Leland *et al.* in [171], or Internet traffic as shown by Crovella and Bestavros in [94]. In order to give a more comprehensive formulation of stochastic bounding processes, generalized Stochastically Bounded Burstiness (gSBB) has been proposed by Jiang *et al.* in [154]. In their work, GPS was briefly studied using sessions modeled as gSBB processes, but without using the full potential of GPS as explained later in this chapter.

10.3 Stochastic network calculus

We give in this section an overview of stochastic network calculus and introduce various notations and definitions necessary for establishing the stochastic bounds described in Section 10.4. The mathematical framework and models described here borrows some notations and definitions of the work from Jiang *et al.* in [154]. We consider here a discrete time system starting at time $t = 0$.

10.3.1 Flow model

In stochastic network calculus, a flow is described as follows:

Definition 10.1 (Input process [154]). *We denote by $A(s,t)$ the traffic amount arriving in the time interval $(s,t]$ such that:*

$$A(s,t) = \sum_{i=s+1}^{t} a(i) \tag{10.5}$$

where $a(t)$ corresponds to the amount of traffic arriving during the interval $(t-1;t]$.

$a(t)$ corresponds here to a non-negative real random variable.

In order to bound this stochastic process, the notion of virtual backlog is used:

Definition 10.2 (Virtual backlog [154]). *We define the virtual backlog of flow A as:*

$$\hat{A}(t;\rho) = \sup_{0 \le s \le t} \{A(s,t) - \rho \cdot (t-s)\} \tag{10.6}$$

ρ is a constant value, generally representing a rate.

This definition is similar to the so-called *virtual-backlog-centric* stochastic arrival curve [152].

Using those two definitions, a flow can be stochastically bounded using the following definition:

Definition 10.3 (Generalized Stochastically Bounded Burstiness (gSBB) [154, Definition 2]). *A flow is said to have generalized Stochastically Bounded Burstiness – noted gSBB – with upper rate ρ and bounding function f, denoted by $A \ll \langle f, \rho \rangle$, if and only if for all $t \ge 0$ and all $x \ge 0$, there holds:*

$$\Pr\left(\hat{A}(t;\rho) > x\right) \le f(x) \tag{10.7}$$

with f a non-negative non-increasing function defined on $\mathbb{R}^+$.

Using those definitions, we can give the gSBB characterization of the superposition of two flows:

Theorem 10.1 (Superposition [154, Theorem 4]). *Assuming two flows $A_1 \ll \langle f_1, \rho_1 \rangle$ and $A_2 \ll \langle f_2, \rho_2 \rangle$, their superposition is also gSBB, such that:*

$$A_1 + A_2 \ll \langle g, \rho_1 + \rho_2 \rangle \tag{10.8}$$

with

$$g_p(x) = [f_1(px) + f_2((1-p)x)]_1$$

where $p \in (0,1)$ and $[x]_1 = \min(1, x)$.

Note that Theorem 10.1 does not make any assumptions on independence. A proof of this theorem is given in [248, Theorem 3].

10.3.2 Server model

We consider in this section a work-conserving server with a constant service rate C serving multiple flows. Flows are described using the notation introduced in the previous subsection. We denote by $D(t)$ as the maximal delay for traversing the server for elements arriving at time t and $Q(t)$ the backlog of the server at time t.

The following theorems can be established on $D(t)$ and $Q(t)$.

Theorem 10.2 (Delay in a work-conserving server [154, Theorem 7]). *Assume that $A \ll \langle f, \rho \rangle$ is the (possibly aggregate) input process of a work-conserving server with service rate $C > \rho$ (stability condition), then:*

$$\Pr(D(t) \geq k) \leq f(k(C-\rho)), \textit{ for } k \in \mathbb{N}^* \tag{10.9}$$

Note that there is no assumption in this theorem on which service discipline is used by the work-conserving server.

In case N flows with input process $A_i \ll \langle f_i, \rho_i \rangle$ share a work-conserving server with rate C following the First-In-First-Out (FIFO) service discipline, the following bound can be defined:

Theorem 10.3 (Delay in a FIFO server [154, Theorem 8]). *Under FIFO, the maximal delay for each source and for the whole aggregate satisfies for any $k > 0$:*

$$\Pr(D(t) \geq k) \leq g(kC - \rho), \textit{ for } k \in \mathbb{N}^* \tag{10.10}$$

with

$$\rho = \sum_{i=1}^{N} \rho_i \quad \textit{and} \quad g(x) = \left[\sum_{i=1}^{N} f_i(p_i x)\right]_1$$

where $\{p_i\}_{i \in [1;N]} \in (0,1)$ satisfies $\sum_{i=1}^{N} p_i = 1$.

The proofs of Theorems 10.2 and 10.3 are given in [154, Theorems 7 and 8].

10.4 Server model for Generalized Processor Sharing

We establish the following conditions for the theorems in this section. We study here a work-conserving server with constant rate C following the Generalized Processor Sharing service discipline. We assume that N flows with input process $A_i \ll \langle f_i, \rho_i \rangle$ share the server, with $\sum_{i=1}^{N} \rho_i < C$ (stability condition). Each flow i has a weight parameter noted here $\phi_i > 0$. We assume without loss of generality that the weights are normalized, such that $\sum_{i=1}^{N} \phi_i = 1$.

Let $A_i^{out}(s,t)$ denote the amount of traffic received by flow i during the time interval $(s+1;t]$. With a server following the GPS service, we have from [197, Equation 1]:

$$\frac{A_i^{out}(s,t)}{A_j^{out}(s,t)} \geq \frac{\phi_i}{\phi_j}, \text{ for } 1 \leq j \leq N, 0 \leq s \leq t \tag{10.11}$$

10.4.1 A first bound

As noted in the introduction, the following stochastic delay bound for GPS was proposed by Jiang *et al.* in [154]:

Theorem 10.4 (GPS bound [154, Theorems 11 and 12]). *Under the conditions noted at the beginning of Section 10.4, the maximum delay of flow i, noted $D_i(t)$, admits the following bounds for $k > 0$:*

1. *If $\phi_i C > \rho_i$:*
$$\Pr(D_i(t) \geq k) \leq f_i\left(\phi_i C k - \rho_i\right) \tag{10.12}$$
2. *Otherwise, use Theorem 10.2.*

We note that this bound can be improved, as the case of $\phi_i C \leq \rho_i$ results in not taking into account the properties of the GPS service discipline. We propose in the rest of this section to extend this theorem.

10.4.2 A second bound using GPS feasible ordering

In order to take into account the properties of the GPS service discipline, we first introduce the notion of *feasible ordering* as defined in [197, Equation 28], :

Definition 10.4 (Feasible ordering). *Under the conditions noted at the beginning of Section 10.4, there exists an ordering among the flows such that after relabeling the flows, we have:*

$$\rho_i < \psi_i \left(C - \sum_{j=1}^{i-1} \rho_j \right) \tag{10.13}$$

with $\psi_i = \frac{\phi_i}{\sum_{j=i}^{N} \phi_j}$

We follow here the same reasoning as in the work from Zhang *et al.* in [255] and Yu *et al.* in [249] for improving the bound described in Theorem 10.4. We decompose the GPS server

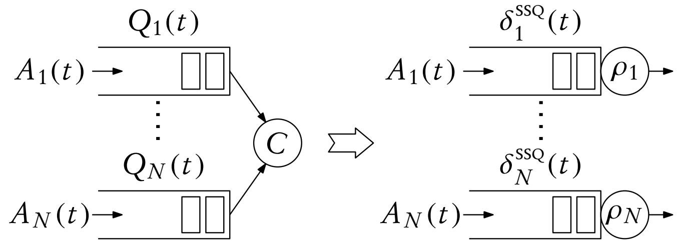


Fig. 10.1: Decomposition of the GPS system into N SSQs

into a set of N servers with rate $\rho_1, \dots, \rho_N$ so that each queue has its own dedicated single server queue (SSQ) as illustrated in Figure 10.1. Let $\delta_i^{SSQ}(t)$ denote the queue length of the SSQ at time t, such that:

$$\delta_i^{SSQ}(t) = \sup_{0 \le s \le t} \{A_i(s,t) - \rho_i(t-s)\} = \hat{A}_i(t, \rho_i)$$

Following [255, Lemma 3], if the SSQs are ordered following a feasible ordering we have:

Lemma 10.1. *For any t:*

$$Q_i^{GPS}(t) \le \delta_i^{SSQ}(t) + \psi_i \sum_{j=1}^{i-1} \delta_j^{SSQ}(t) \tag{10.14}$$

Using Theorem 10.1 and Lemma 10.1, we can bound the individual queue sizes:

Theorem 10.5. *Under the conditions noted at the beginning of Section 10.4 and the sessions following a feasible ordering, we get that the queue size of flow i is characterized by:*

$$\Pr(Q_i^{GPS}(t) \ge k) \le g_i^{GPS}(k) \tag{10.15}$$

with

$$g_i^{GPS}(x) = f_i(p_i x) + \sum_{j=1}^{i-1} f_j\left(\frac{p_j}{\psi_i} x\right) \tag{10.16}$$

for any $\{p_j\}_{j \in [1;i]} \in (0,1)$ satisfying $\sum_{j=1}^{i} p_j = 1$.

Proof. Applying Lemma 10.1, we have:

$$Q_i^{GPS}(t) \le \hat{A}_i(t, \rho_i) + \psi_i \sum_{j=1}^{i-1} \hat{A}_j(t, \rho_j)$$

We use here the method used in the proof of [248, Theorem 3], method which is also used for proving Theorem 10.1. For any $\{p_j\}_{j \in [1;i]} \in (0,1)$, satisfying $\sum_{j=1}^{i} p_j = 1$, if $\hat{A}_i(t, \rho_i) \le p_i k$ and $\psi_i \hat{A}_j(t, \rho_j) \le p_j k$ for all $j = 1, \dots, i-1$, we have $Q_i \le k$. Hence,

$$\left\{Q_i^{GPS}(t) \ge k\right\} \subseteq \left\{\hat{A}_i(t, \rho_i) \ge p_i k\right\} \cup \left(\bigcup_{j=1}^{i-1} \left\{\psi_i \hat{A}_j(t, \rho_j) \ge p_j k\right\}\right) \tag{10.17}$$

Therefor,

$$\Pr(Q_i^{\text{GPS}}(t) \geq k) \leq \Pr(\hat{A}_i(t,\rho_i) \geq p_i k) + \sum_{j=1}^{i-1} \Pr\left(\hat{A}_j(t,\rho_j) \geq \frac{p_j}{\psi_i} k\right) \tag{10.18}$$

By applying the definition of gSBB from Equation (10.7), we get the expression in Equation (10.15). □

Using Theorem 10.5, we can then bound the delay:

Lemma 10.2. *With the same conditions than Theorem 10.5, we get that the delay of flow i is characterized by:*

$$\Pr(D_i^{GPS}(t) > k) < g_i^{GPS}(\phi_i C k - \rho_i) \tag{10.19}$$

with g_i^{GPS} defined in Equation (10.16).

Proof. As flow i is guaranteed a rate of $\phi_i C$, we get:

$$D_i^{\text{GPS}}(t) \leq \left\lceil \frac{Q_i^{\text{GPS}}(t)}{\phi_i C} \right\rceil$$

Applying this relation with Theorem 10.3, we get:

$$\Pr(D_i^{\text{GPS}}(t) \geq k) \leq \Pr(Q_i^{\text{GPS}}(t) \geq k\phi_i C - \rho_i)$$

We then apply Theorem 10.5 to conclude this proof. □

We note that under the condition that flow i admits the relation $\rho_i < \phi_i C$, Theorem 10.4 and Lemma 10.2 give the same bound as the flows can be ordered such that $i = 1$.

We propose now various extensions of Lemma 10.2.

10.4.3 Hybrid priority and GPS scheduler

We propose in this section to study a hybrid scheduler similar to the one studied in Chapter 4 and illustrated here in Figure 10.2. Suppose that we have N sources sharing a work-conserving server with service rate C following a strict priority discipline, and M sources sharing the left-over bandwidth with a GPS service discipline. Let $A_j \ll \langle f_j^{\text{SPQ}}, \rho_j^{\text{SPQ}} \rangle$ be the input process for source j among the N flows and $D_j^{\text{SPQ}}(t)$ the delay for source j. We define similarly $B_n \ll \langle f_n^{\text{GPS}}, \rho_n^{\text{GPS}} \rangle$ for the M flows, $D_n^{\text{GPS}}(t)$ the delay for source n and ϕ_i the associated GPS weight.

Since the N flows have always the priority over the other M flows and we use here a fluid model, the M flows do not have any influence on the performance of the N flows. The bound on $D_j^{\text{SPQ}}(t)$ can hence be given by using [154, Theorem 10], namely:

$$\Pr(D_j^{\text{SPQ}}(t) \geq k) \leq \begin{cases} f_j^{\text{SPQ}}(kC - \rho_1^{\text{SPQ}}) & \text{if } j = 1 \\ g_j^{\text{SPQ}}(k(C - r_j)) & \text{otherwise} \end{cases} \tag{10.20}$$

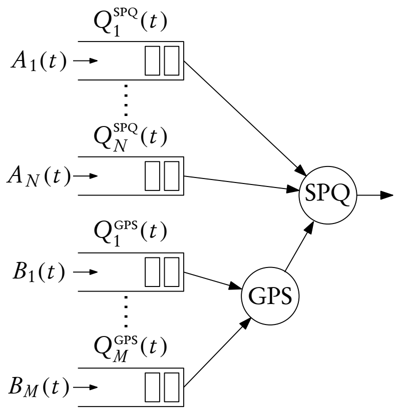


Fig. 10.2: Scheduling architecture with mixed priority and fair queuing policies

with $r_j^{\text{SPQ}} = \sum_{i=1}^{j} \rho_i^{\text{SPQ}}$ and

$$g_j^{\text{SPQ}}(x) = \left[\sum_{i=1}^{j} f_i^{\text{SPQ}}(p_{ji}x)\right]_1 \tag{10.21}$$

with $p_{ji} \in (0,1)$ and $\sum_{i=1}^{j} p_{ji} = 1$.

For the bound on $D_n^{\text{GPS}}(t)$, we can define the following Lemma:

Lemma 10.3. *Under the conditions specified at the beginning of this section, and that the sessions follow a feasible ordering, we get that the bound on D_n^{GPS} is given by:*

$$\Pr(D_n^{GPS}(t) \geq k) \leq \Pr(D_n^{SPQ}(t) \geq pk) + g_n^{GPS}((1-p)\phi_n Ck - \rho_n) \tag{10.22}$$

$\forall p \in (0,1)$ *and with $g_n^{GPS}(x)$ defined in Equation* (10.16).

Proof. Flow n is guaranteed a rate of $\phi_n C$ when the N flows are idle, such that:

$$D_n^{\text{GPS}}(t) \leq D_N^{\text{SPQ}}(t) + \left\lceil \frac{Q_n^{\text{GPS}}(t)}{\phi_n C} \right\rceil \tag{10.23}$$

For any $0 < p < 1$, if $D_N^{\text{SPQ}}(t) < pk$ and $\left\lceil \frac{Q_n^{\text{GPS}}(t)}{\phi_n C} \right\rceil < (1-p)k$, then we have $D_n^{\text{GPS}}(t) < k$. Hence,

$$\{D_n^{\text{GPS}}(t) \geq k\} \subseteq \left\{D_N^{\text{SPQ}}(t) \geq pk\right\} \cup \left\{\left\lceil \frac{Q_n^{\text{GPS}}(t)}{\phi_n C} \right\rceil \geq (1-p)k\right\} \tag{10.24}$$

Therefore by taking the probabilities and applying Theorem 10.3 to the second term of the right-hand side of the equation, we get:

$$\Pr(D_n^{\text{GPS}}(t) \geq k) \leq \Pr\left(D_N^{\text{SPQ}}(t) \geq pk\right) + \Pr\left(Q_n^{\text{GPS}}(t) \geq (1-p)k\phi_n C - \rho_n\right) \tag{10.25}$$

Applying Theorem 10.5 on the second term of the right-hand side of the inequality concludes the proof. □

10.4.4 Packetized Generalized Processor Sharing

The bounds derived in Section 10.4 assume that the traffic is fluid, namely that it is indefinitely divisible. Because in practice the notion of packets is used – which is not indefinitely divisible – Packetized GPS (PGPS) is traditionally used. An example of algorithm is the Worst-Case Fair Weighted Fair Queuing (WF^{2}Q) algorithm [58].

It was shown by Parekh and Gallager in [197] that PGPS does not fall behind GPS by more than one maximum-size packet, noted here L_{max}. Hence, the following relation can be established:

$$Q_i^{PGPS}(t) \leq Q_i^{\text{GPS}}(t) + L_{max} \tag{10.26}$$

Theorem 10.6. *Under the same conditions than Theorem 10.5, we then have:*

$$\Pr(Q_i^{PGPS}(t) \geq k) \leq f_i(p_i(k - L_{max})) + \sum_{j=1}^{i-1} f_j\left(\frac{p_j}{\psi_i}(k - L_{max})\right) \tag{10.27}$$

Proof. Following Equation (10.26), we have:

$$\begin{aligned}\Pr(Q_i^{PGPS}(t) \geq k) &\leq \Pr(Q_i^{\text{GPS}}(t) + L_{max} \geq k)\\ &= \Pr(Q_i^{\text{GPS}}(t) \geq k + L_{max})\end{aligned}$$

We then apply Theorem 10.5 to conclude the proof. □

Following the same method than the one used for Lemma 10.2, a bound on the delay for PGPS can be also given.

10.5 Stochastic flow model for TCP

We mainly focused in Section 10.4 on server models and let the question of flow characterization mostly open by using the notion of generalized Stochastically Bounded Burstiness (gSBB) defined in Definition 10.3. We now focus in this section on the question of how to model flows.

Our approach is as follows. We concentrate here on flow-level characteristics as in Chapters 7 and 8, and not on packet-level. In order to simulate the bandwidth sharing of elastic flows, instead of focusing on the packet-layer where a feedback-based mechanism would be needed, we make use of the GPS service discipline for "emulating" it. This means that we assume that flows adapt instantaneously their bandwidth with regards to the number of active flows. While this assumption is valid when packet scheduling is used, it is not entirely true when end-to-end congestion protocols are used, such as TCP. Nevertheless, it is a good approximation for larger timescales (*i.e.* generally one or two orders of magnitude larger than packet latencies).

10.5.1 Arrival process

The goal of this section is to define the gSBB characterization of the studied flows. Since we look at the flow-level, we assume that messages of random size L_i are sent at different random points in time noted t_i. This principle is illustrated in Figure 10.3. The $\{L_i\}_{i\in\mathbb{N}^*}$ correspond here to application layer messages, such as HTTP requests or files, where in general they need to be split into multiple TCP segments, *i.e.* $L_i \gg$ *TCP segment size*. The bound on delay presented in Section 10.4 will then correspond to the transfer duration of a higher-layer message as highlighted earlier in Equation (10.3).

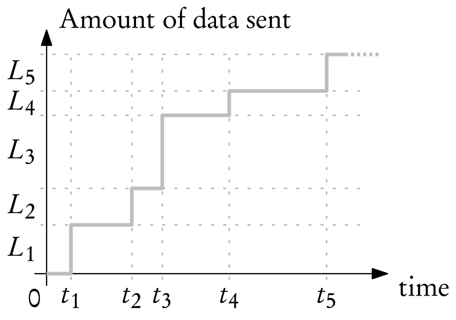


Fig. 10.3: Illustration of a compound Poisson process

Assuming that the $\{L_i\}_{i\in\mathbb{N}^*}$ are independent and follow the distribution F and $\{t_{i+1} - t_i\}_{i\in\mathbb{N}^*}$ are also independent and follow the distribution G with positive value, we have a so-called Lévy process. Restricting to the case where $L_i \geq 0, \forall i$, we have a stochastic process similar to that of a $G/G/1$ queue.

In order to simplify the evaluation made in this chapter, we focus here on the case where G follows an exponential distribution with parameter λ. This specific Lévy process is then called a compound Poisson process, and it is defined as:

Definition 10.5 (Compound Poisson process). *A compound Poisson process is a stochastic process with jumps arriving according to a Poisson process $\{N(t), t \geq 0\}$ with parameter λ and the size of the jumps follow the general distribution F. We denote by CPP(λ, F) such stochastic process.*

A compound Poisson process has arrival process:

$$A_{cpp}(s,t) = \sum_{i=1}^{\nu(s,t)} L_i = \sum_{i=1}^{N(t-s)} L_i \tag{10.28}$$

with $\nu(s,t)$ corresponding to the number of arrival during the interval $(s,t]$, and $\{L_i\}_{i\geq 1}$ are independent and identically distributed random variables with distribution F.

It can be shown that the moment-generating function of the arrival process of the compound Poisson process *CPP*(λ, F) can be defined as:

$$\log \mathbb{E}\left[e^{\theta A_{cpp(s,t)}}\right] = \lambda(t-s)(M_F(\theta) - 1), \forall\theta \in \mathbb{R} \tag{10.29}$$

When $\{L_i\}_{i\in\mathbb{N}}$ follow an exponential distribution with parameter μ, we have:

$$\frac{1}{\theta}\log \mathbb{E}\left[e^{\theta A_{cpp(s,t)}}\right] = \frac{\lambda}{\mu - \theta}(t-s), \forall\theta \in \mathbb{R} \tag{10.30}$$

It can then be shown (see [151, Section 2.3.4]) that A_{cpp} admits a gSBB characterization, such that $A_{cpp} \ll \langle f_{cpp}, \rho_{cpp} \rangle$ with:

$$f_{cpp}(x) = e^{-\theta\alpha} e^{-\theta x} \tag{10.31}$$

$$\rho_{cpp} = \frac{\lambda}{\mu - \theta} + \alpha \tag{10.32}$$

for any $\alpha \geq 0$ and $\theta > 0$.

10.5.2 GPS weights selection

As noted in the introduction of this section, the bandwidth sharing of the different flows is here assured by the GPS service discipline, but we let the question of the weights choice open. We propose here two approaches:

1. We set them manually in order to satisfy some specific requirements;
2. We can use a realistic TCP bandwidth sharing following a method analogous to the one developed in Section 8.4.

We note that when using the first approaches, actual GPS (or PGPS) schedulers might be used to enforce the bandwidth sharing prescribed by the requirements.

We detail here the second approach. We define the following steps, similar to the one presented in Section 8.4:

Step A For each server of the topology, we compute the steady-state bandwidth of its traversing flows as if the other flows of the network were idle. We make use here of the method developed in Chapter 7 for getting the realistic flow bandwidth sharing.

Step B The bandwidth shares computed in Step A will then serve as the respective weights of the flows for each GPS server. We then use the traditional methods of stochastic network calculus for analyzing networks and the bounds developed in Section 10.4 to get the stochastic bounds.

A few remarks should be noted for this second approach. As the TCP bandwidth sharing between the flows might vary due interfering traffic at other points of the network, we only get approximations of the bounds. Also, those bounds are not *guaranteed* as there are no elements in the network enforcing that the flows behave as expected (*i.e.* that the flows are TCP-friendly). One benefit of this method compared to the one developed in Chapter 8 is that it is less computationally expensive.

10.6 Numerical evaluation

We will now evaluate numerically in this section the results presented in Section 10.4.

10.6.1 Comparison of the bounds from Sections 10.4.1 and 10.4.2

We evaluate a scenario with three flows following a compound Poisson process with exponential message sizes. We make use of the gSBB characterization of a compound Poisson process detailed in Equations (10.31) and (10.32) and use the values noted in Table 10.1 with $C = 100\,\text{bit/s}$. We note that the flows are already ordered following a feasible ordering.

Tab. 10.1: Flows parameters used for the evaluation

Flow	**1**	**2**	**3**
Arrival rate λ_i (s^{-1})	0.1	0.5	0.8
Message rate μ_i (bit^{-1})	2×10^{-2}	2×10^{-2}	2×10^{-2}
GPS weight ϕ_i	0.5	0.3	0.2

The results of the evaluation are presented in Figure 10.4. For simplification purpose, we took $\alpha = 0$, $p_k = 1/i$, and optimized according to the θ parameter for the numerical evaluations. As expected, Equations (10.12) and (10.19) give the same bound for flow 1 (as $\rho_1 < \phi_1 C$). For flow 2 we have $\rho_2 \geq \phi_2 C$, and the bound given by Equation (10.19) is tighter than the bound given by Equation (10.12). Finally, the bound Equation (10.12) for flow 3 gives a tighter bound than Equation (10.19), mainly due to the fact that the assumptions used for Section 10.4.2 does not take into account the statistical bandwidth sharing between the different flows.

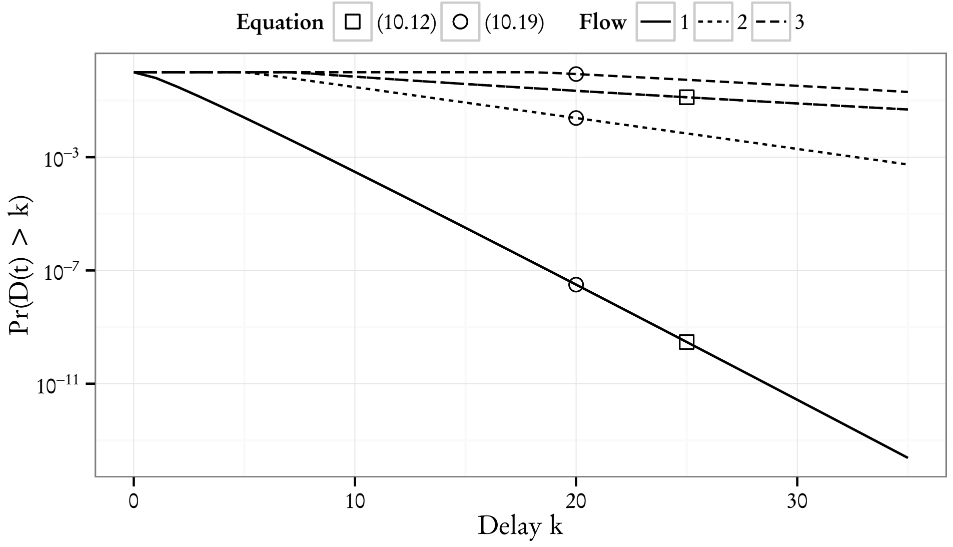


Fig. 10.4: Numerical evaluation of the bounds of Equations (10.12) and (10.19)

10.6.2 Comparison with simulations

We propose to evaluate here the tightness of the bound presented in Equation (10.19) against simulations made with OMNeT++ [17] and its framework INET [14]. We focus here on the scenario presented in Section 10.6.1.

We simulated an Ethernet dumbbell topology similar to the one presented in Figure 7.8, with three client-server pairs (one for each flow). The switches in the network use the Deficit Round Robin (DRR) scheduling policy, which corresponds to a simpler version of Packetized GPS, with the weights detailed in Table 10.1. At the application layer, flows are simulated following a compound Poisson process with increments following an exponential distribution as explained in Section 10.5.1 with the parameters taken in Table 10.1, but adapted in order to take into account the encapsulation overhead. We use here TCP Reno for the transport layer. We note that we scaled C and μ_i from Table 10.1 in order to use standard Ethernet link speeds, and that because of the relatively large file size, the slow-start phase of TCP can be neglected here for the numerical evaluation.

Transfer times corresponds to the difference between the last ACK is received from the server, and the TCP three-way handshake is performed. We follow a Monte-Carlo approach with 20 runs. Each run has a duration of 10 000 s in order to be able to capture rare-events.

Results of the simulations are compared to the bounds from Equation (10.19) in Figure 10.5. As expected, the transfer durations in the simulation are below the computed bounds for the three studied flows. We notice that the bounds are relatively loose, but they are still within the same order of magnitude.

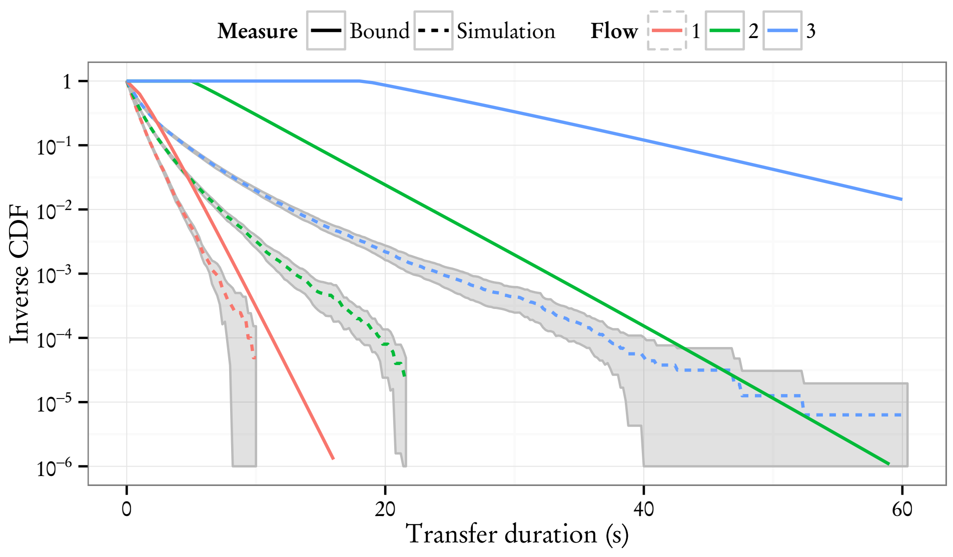


Fig. 10.5: Comparison of the stochastic bound on transfer duration from Equation (10.19) with OMNeT++ simulations. The gray ribbon corresponds to a 0.95 % confidence interval for the simulations.

10.7 Conclusion on stochastic guarantees

Using a traffic model based on the notion of generalized Stochastically Bounded Burstiness (gSBB), we proposed in this chapter a new stochastic bound on sessions scheduled by a GPS server. We extended a previous analysis of GPS using flows following a gSBB process which did not take into account some important properties of GPS. By using the notion of feasible ordering introduced in [197], we provided a better bound for some cases as shown with a numerical evaluation. We also extended this work to a hybrid strict priority and GPS server similar to the one studied in Chapter 4, as well as Packetized GPS, a service discipline used in practice in packet networks.

Using those bounds we proposed a method for applying stochastic network calculus to TCP-based traffic and gave a bound on the transfer time of a message. By leveraging on the results developed in Chapters 7 and 8, we derived an approach to assign weights to the GPS servers of the network reproducing realistic bandwidth sharing of TCP traffic.

Via numerical evaluation, we compared the bound developed here and showed that our new bound provides tighter bounds in some cases. We also evaluated the tightness of the bound against discrete-event simulations performed with OMNeT++.

Regarding future research directions, some more advanced Lévy processes than the compound Poisson process with message sizes following an exponential distribution would be useful for modeling more realistic traffic patterns of the application layer. An example would be message sizes following a Pareto distribution, as often shown in statistical analysis of network captures. Secondly, we made here the assumption that the TCP bandwidth sharing can be approximated by GPS. While it is true for large timescales, more advanced models would be necessary for including the effect of slow-start for instance. Finally, traffic patterns following the request-reply paradigm as developed in Section 8.5.3 would prove useful for studying real network protocols.

Key insights and contributions

Extension of stochastic bound for GPS and PGPS service discipline We proposed in this chapter a new stochastic bound on session following a gSBB characterization served by a server with constant rate with GPS or PGPS service discipline.

Stochastic bound for hybrid priority and GPS scheduling We extended the bound previously described to the analytical study of hybrid scheduling as presented in Chapter 4. This extension is valuable for network engineering as such scheduling architecture is often present in COTS switches under the name low-latency queuing.

Application to TCP bandwidth sharing We applied this bound to TCP in order to get bounds on transfer time (Research Objective **O2.4**). We propose a simple characterization of application layer behavior based on a Lévy process. We also provided a method leveraging on the work from Chapters 7 and 8 for assigning the GPS weights.

Part V

SUMMARY AND CONCLUSION

11. TOWARDS FLOW-BASED MIXED-CRITICALITY NETWORKS

This chapter gives a global review over the results and contributions presented in this thesis around two aspects. The first aspect treated here is the comparison between our contributions and the state-of-the-art. The second aspect concentrates on a set of global recommendations on how to use the results in a more general context of network engineering.

Structure of this chapter

In Section 11.1, we first focus on packet scheduling architectures as studied in Part II for Research Objective **O1**. We then concentrate in Section 11.2 on performance evaluate frameworks for packet and flows as investigated in Parts III and IV for Research Objectives **O2.1** to **O2.5**. Finally, Section 11.3 is dedicated to modeling of higher layers (application and user), which is part of Research Objective **O2.5**.

11.1 Packet scheduling for mixed-criticality networks

We described in Section 2.4 our vision of an ideal industrial Ethernet network. In such network, one should be able to mix real-time traffic with requirements regarding end-to-end latency and jitter (abbreviated "RT" later on), with so-called best-effort flows where coarser and less strict requirements are used (abbreviated "BE" later on). We addressed this point in Chapters 3 to 5, where various packet scheduling algorithms were used and evaluated using discrete event simulation in two scenarios used in airplanes.

The following packet scheduling architectures were studied:

- Well-known schedulers, namely Priority Queuing (PQ), Fair Queuing (FQ) and a hybrid Priority/Fair Queuing (PQ+FQ), evaluated in Chapters 3 and 4;
- A proposition by the IEEE Audio/Video Bridging (AVB) Task Group which uses a shaping function called Credit Based Shaper (CBS) for audio and video traffic and priority queuing for the rest (CBS+PQ), studied in Chapter 3;
- A novel algorithm called Time-Aware Deficit Round Robin (TADRR), which mixes the idea of time-triggered traffic with fair queuing, proposed and studied in Chapter 5.

Table 11.1 presents an overview over the benefits and drawbacks of those different architectures. We use purely performance based comparison criteria as well as practical use ones.

Criteria \ Architectures	**PQ**	**FQ**	**PQ+FQ**	**CBS+PQ**	**TADRR**
Latency (RT)	+	+	+	∘	++
Isolation between RT and BE	+	∘	+	+	++
Bandwidth sharing (BE)	−	+	+	−	+
Implementation complexity	+	−	−	∘	−
Configuration complexity	+	∘	∘	∘	−
Availability in COTS switches	++	+	+	∘	−−

Tab. 11.1: Comparison between the different architectures studied in Chapters 3 to 5. Notation detailed in Table 11.2

Symbol	−−	−	∘	+	++	∅
Signification	Very Bad	Bad	Neutral	Good	Very Good	Not applicable

Tab. 11.2: Notation used in Tables 11.1, 11.3 and 11.4

The following criteria where used for the comparison:

Latency (RT) We evaluate here the performance in term of latency that the scheduler can offer for the real-time traffic. Because the TADRR algorithm is able to eliminate queuing delay for real-time packets, it offers the best performances compared to the other algorithms. We showed in Chapter 5 than performances can be improved by more than an order of magnitude when using TADRR. Schedulers using priority queuing for the real-time packets (*i.e.* PQ and PQ+FQ) generally offer good performances as those packets will always have the priority over best-effort traffic. We showed in Chapter 3 that a correctly configured fair queuing algorithm can also offer similar latencies for real-time traffic. Finally, the AVB architecture has a small impact on the latency of real-time traffic due to its shaping function as showed in Chapter 3.

Isolation between RT and BE We evaluate here the impact that the best-effort traffic load can have on the performances of the real-time traffic. As for the previous point, because the TADRR algorithm completely separates both traffic classes using time windows, the best-effort traffic load will have no impact on the real-time performances. All architectures using a priority mechanism for separating the two classes of traffic (*i.e.* PQ, PQ+FQ and CBS+PQ) will offer a good isolation as the real-time traffic will only be delayed by one maximum-sized packet in the worst-case. Regarding the fair-queuing algorithm, we showed in Chapters 3 and 4 that we have a good isolation between the two classes of traffic, but with a larger impact that with the priority-based separation.

Bandwidth sharing (BE) Because we consider that the best-effort traffic is mostly composed of elastic flows, one point to consider is the bandwidth sharing between those flows or classes of traffic. In this case, architectures using a fair queuing algorithm for serving the best-effort classes (*i.e.* FQ, PQ+FQ and TADRR) will offer the best flexibility and fairly distribute the available bandwidth. The scheduler using a priority-based algorithm (*i.e.* PQ and CBS+PQ) will give all the bandwidth to the highest-priority class and potentially starve the ones with lower priority.

Implementation complexity When using those scheduling architectures in real devices, the implementation complexity plays an important role, mostly because the correct behaviors of those algorithms have to be verified. The algorithm for priority based schedulers is the simplest to implement. The CBS+PQ architecture offers a fairly unambiguous algorithm to implement. The TADRR algorithm is also reasonably straightforward to implement, but special care have to be taken in order to correctly synchronize the time windows across the network. Finally, architectures based on fair queuing (*i.e.* FQ and PQ+FQ) generally have more complex algorithms, where special attention to details have to be used depending on which variant of the fair queuing algorithm is used.

Configuration complexity Another essential part of using those architectures is the configuration of the schedulers. As for the implementation, priority based schedulers are generally straightforward to configure when using latency requirements. Fair queuing based architectures usually offer weight parameters where mixing latency and bandwidth requirements involves more effort. The CBS algorithm does require setting a bandwidth limit, which is a function of the input flows, but guidelines are given by the IEEE AVB standards. Finally, because offline scheduling is needed for TADDR in addition to the fair queuing weights configuration, it is the most difficult one to configure.

Availability in COTS switches One final point that we consider is the actual availability of the architecture in commercial off-the-shelves (COTS) devices. While almost all switches offer priority queuing, the availability of fair queuing architectures is generally scarcer. Due to its standardization by the IEEE and the potential mass-market use, the CBS+PQ architecture is starting to be available in some equipments and it availability should grow in the future. Finally, TADRR is not available in COTS switches, but other similar architectures based on time-triggered traffic are available (ex: Time-Triggered Ethernet) or will be in the soon future (ex: IEEE Time Sensitive Networking).

Various details and references about the comments made here are detailed in Chapters 3 to 5.

Conclusion of the comparison

From Table 11.1 we note that there is no clear winner which fulfills all criteria presented in this section. It means that some compromises have to made based on which parameters are the most important. If the bandwidth sharing between the different classes of the best-effort traffic is less significant, the best solution would be to use priority queuing due to its performance, ease of use and large availability. If it does play a more decisive role, the best solution would be to use a hybrid priority/fair queuing architecture, but it comes at the cost of higher implementation and configuration complexity. Finally, if the performance of the real-time traffic is critical as illustrated in Chapter 5, the best solution would be to use the TADRR architecture, which comes at the cost of higher implementation and configuration complexity, and custom hardware.

11.2 A framework for packet-level and flow-level Quality-of-Service

We reviewed previously in this thesis various mathematical frameworks targeted either at packets performances and used for avionic networks (in Section 2.3), or targeted at flows performances and TCP (in Section 6.3). The following frameworks were studied:

- In Section 2.3, deterministic and stochastic network calculus (DNC and SNC) were reviewed in the context of real-time traffic used in avionic networks, and it was shown that it is not really adapted to the evaluation of TCP;
- In Section 6.3, various frameworks for the performance evaluation of TCP flows and its bandwidth sharing were reviewed, namely:
  - The network utility maximization approach (NUM), which describes the bandwidth sharing of TCP as an optimization problem;
  - The fluid models (FM), which describes the TCP window size evolution using differential equations;
  - The queuing theory approach (QT), either describing packet-level behavior as well-known queues, or flow-level behavior using a Processor Sharing queue;
  - The fixed-point approach (FP), using packet-level behavior of TCP describing its bandwidth as a function of round-trip time and drop probability with models of queues.

Based on the review made in Section 6.3, we made various contributions to the so-called fixed-point approach via separate iterations in Chapters 7, 8 and 10, noted hereafter C1 (Chapter 7), C2 (Chapter 8) and C3 (Chapter 10). While we already compared the different frameworks presented in Section 6.3 with Table 6.4, we propose here to extend it to deterministic and stochastic network calculus, as well as the various contributions we made in Chapters 7, 8 and 10. This comparison is summarized in Table 11.3.

The following criteria were used for the comparison:

End-to-end packet-level QoS Because the requirements for our target network are a mix between packet-level and flow-level requirements, we are interested in a model able to characterize the end-to-end performances of packets (ex: latency, jitter). The DNC, SNC and QT approaches are well suited for this task. We also note that due to our use of SNC in Chapter 10, C3 is also able to deal with packet-level performances as illustrated for instance in Section 10.4.3. All the other approaches are tailored for flow-level performances.

End-to-end flow-level QoS As noted in the previous point, an ideal approach should be able to also characterize end-to-end performances of flows (ex: bandwidth, transfer duration), more precisely TCP flows. The frameworks reviewed in Section 6.3 (*i.e.* NUM, FM, QT and FP) as well as the contributions from Chapters 7, 8 and 10 (*i.e.* C1, C2 and C3) are appropriate for this function. We note that while the FM framework is targeted at flows, the current methods which are used for solving the system of differential equations is generally not suitable for characterizing QoS attributes. Finally, the DNC and SNC are not well-suited for TCP-based traffic as illustrated in Sections 2.3.1 and 2.3.2.

Criteria \ Approaches	Deterministic NC	Stochastic NC	Utility max. (NUM)	Fluid models (FM)	Queuing theory (QT)	Fixed-point (FP)	FP+Chapter 7 (C1)	FP+Chapter 8 (C2)	FP+SNC+Chapter 10 (C3)
End-to-end packet-level QoS	+	+	−	−	+	−	−	−	+
End-to-end flow-level QoS	−	−	+	○	+	+	+	+	+
Applicability to dynamic behaviors	−	+	○	+	+	−	−	+	+
Statistical performances	−	++	+	−	++	−	−	+	++
Realistic TCP bandwidth sharing	∅	∅	−	+	○	+	+	++	+
Nonintuitive behavior of TCP	∅	∅	+	+	−	−	+	+	+
Appl. to multiple bottleneck topo.	+	+	○	+	○	+	+	+	+
Computational complexity	○	○	○	○	○	○	○	−	○
Ease of use	++	○	+	+	++	+	+	+	○

Tab. 11.3: Comparison between the different mathematical frameworks reviewed in Sections 2.3 and 6.3 and proposed in this thesis in Chapters 7, 8 and 10. Notation detailed in Table 11.2.

Applicability to dynamic behaviors As flows and packets activity generally fluctuates in a real network (ex: active/idle behavior), a good approach should be able to take those fluctuations into account. While not initially designed for this purpose, we extended the FP approach in Chapters 8 and 10 (C2 and C3) to be able to study such effect. Other frameworks such as the SNC, FM and QT ones can also be used for dynamic flow behaviors. We note that the NUM approach is able to characterize ON/OFF flows performances, but it is limited to single bottleneck topologies.

Statistical performances In relation to the previous point, when modeling the dynamic behavior of flows and packets, a good method should be able to give statistical characteristics of the QoS parameters such as mean, variance or distribution. The best approaches are able to give statistical distributions (ex: tail), such as the SNC or QT ones. While the FP approach does not initially support such characterization, we extended it in Chapters 8 and 10 to means (C2) and statistical distributions (C3). The NUM approach does only support mean performances. Finally, as noted for the *End-to-end flow-level QoS* criteria, due to current methods used for solving the differential equations, the FM approach does not support statistical performances.

Realistic TCP bandwidth sharing We examine here how well the approaches map the bandwidth sharing of multiple TCP flows to the reality. One clear winner here is our contribution described in Chapter 8 (C2), as we have a realistic model of both the slow-start and congestion-avoidance phases of TCP. Other methods focus only on the congestion-avoidance phase (*i.e.* FM, FP, C1 and C3), meaning they are more appropriate for flows where the slow-start phase is negligible. Finally, some methods use only idealized behavior of TCP (*i.e.* NUM and QT).

Nonintuitive behavior of TCP As seen in Chapter 7, TCP has some non-intuitive behavior on some topologies, such as for instance in presence of cross-traffic. Frameworks taking into account such effects will be more accurate. While such effect was not taken into account initially in the FP approach, our contribution in Chapter 7 (C1) was to take into account the cross-traffic effect. This effect was also included in to NUM and FM approaches, but not in the QT one.

Applicability to multiple bottleneck topologies As we study networks where multiple bottlenecks may be present, a good framework should be able to evaluate those topologies. While almost all methods compared here have such property, the NUM and QT approaches were marked as neutral because of some restrictions. For the NUM approach, only single bottleneck topologies can be studied when ON/OFF TCP flows are studied. Similarly for the QT approach at a flow level, only single bottleneck topologies can be studied. More detailed explanations are given in Section 6.3.

Computational complexity Regarding computational complexity, there is no clear winner between the different approaches. While some frameworks can *simplify* networks when flows share the same properties – an assumption which is not always applicable in reality – all approaches are generally linear or quadratic with the number of flows and linear with the number of nodes. Despite this, our contribution described in Chapter 8 is the worst one as it is exponential.

Ease of use One final point to consider is the ease of use of the framework from an engineering perspective. Most methods reviewed here are fairly straightforward to use. Compared to the other frameworks, the DNC and QT approaches are ahead of the others due to the availability of tools (open-source and proprietary) and well-established literature. The SNC and framework contributed in Chapter 10 are less straightforward due to the more involved theories and care to details needed to use them on real networks (ex: statistical independence).

Various details and references about the comments made here are detailed in Sections 2.3 and 6.3 and Parts III and IV.

Conclusion of the comparison

As in Section 11.1, we note from Table 11.3 that there is no clear winner which fulfills all criteria presented in this section. One promising approach is the hybrid one developed in Chapter 10 (C3) as it is the only one able to deal with both packet-level and flow-level QoS at the same time and applicable to multiple bottlenecks topologies. This means that it is the best suited for our general goal of a mixed packet and flow requirements. Compared to the other frameworks, it comes at the cost of a less simple use and it does not take into account the slow-start phase of TCP. The queuing theory framework (QT) is also interesting with respect to this double use and thanks to its wide range of literature. But mixing both packet-level and flow-level QoS at the same time with queuing theory is not straightforward and can be limited to the single bottleneck case.

Regarding methods essentially focused on TCP flows, the framework developed in Chapter 8 (C2) is well positioned due to its inclusion of slow-start, but it comes at the cost of a high computational complexity. If one is not interested in dynamic flow behaviors, the framework developed in Chapter 7 (C1) and the network utility maximization (NUM) are interesting alternatives due to their better computational complexity and the support of a non-intuitive TCP behavior. Those two approaches differentiate themselves on the type of network which is used (inclusion or not of queuing delay as seen in Section 7.8.6) and TCP bandwidth sharing (*i.e.* realistic or idealized).

Regarding methods only focused on packet level QoS, the deterministic and stochastic network calculi (DNC and SNC) are still the best suited as presented in Section 2.3. Their main advantages are that they provide bounds and they are adopted by the aeronautic industry (for DNC).

Finally, Table 11.3 is good way to have an overview over the contributions and various iterations made in Parts III and IV compared to the original fixed-point framework (FP). Our initial contribution in Chapter 7 (C1) brought the TCP cross-traffic non-intuitive behavior to the fixed-point framework, without any change to the other properties of the method except a slightly larger computation complexity due to the increased number of parameters to take into account. Our second contribution in Chapter 8 (C2) brought the study of dynamic active/idle flow behavior with a more realistic model of TCP (slow-start phase), but it came at the cost of a far larger computation complexity compared to the initial method. Lastly, our final contribution to the fixed-point framework in Chapter 10 (C3) brought a more detailed statistical characterization of the performances (tail), with a better computation complexity than the one from Chapter 8, but at the cost of a less realistic TCP model.

11.3 A realistic model for application layer behavior

We described in Sections 2.3 and 6.3 and proposed in this thesis in Chapters 7, 8 and 10 various mathematical framework for modeling the performance of packets or flows. While such models are useful for the performance evaluation of networks, they do not address the question of how to choose the parameters and distribution functions used by the models in order to characterize realistic applications behaviors. We addressed this challenge in Chapter 9 and proposed a model and a set of tools called RENETO.

We propose to compare here our solution to a portion of the ones presented in Section 9.2, namely: Harpoon [225], Swing [235], Tmix [136], and LiTGen [209]. We chose to restrict our comparison to those models are they are the nearest to the approach we decided to choose, which is to abstract out the packet-layer behavior and to have a protocol-agnostic method. We refer to Section 9.2 for a larger overview on application layer modeling. The comparison between the different methods and tools and our contribution from Chapter 9 is summarized in Table 11.4.

The following criteria were used for the comparison:

Based on empirical analysis of real traffic In order to get realistic traffic behavior, we argue that a good method should be based on the analysis of real traffic. While all approaches presented here are based either on the analysis of traffic dumps or other related methods (ex: NetFlow), we note that Harpoon uses a more coarse method which may not match

Criteria \ Approaches	**Harpoon [225]**	**Swing [235]**	**Tmix [136]**	**LiTGen [209]**	**RENETO (Ch. 9)**
Based on empirical analysis of real traffic	○	+	+	+	+
Support of TCP applications	+	+	+	+	+
Support of UDP applications	○	+	−	−	+
Application-layer agnostic	+	+	+	+	+
Online model update	+	−	−	−	−
Bidirectional communications	−	+	+	+	+
User modeling	−	+	−	−	+
Correlation between parameters	−	−	−	+	+
Traffic generation in real networks	+	+	−	∅	−
Traffic generation in simulators	○	−	+	∅	+

Tab. 11.4: Comparison between the different models and tools for application layer modeling

over shorter intervals (ex: below 300 s).

Support of TCP applications As we mainly studied TCP-based flows in Parts III and IV, we are interested in models and tools supporting TCP-based applications. All models compared in Table 11.4 support such functionality.

Support of UDP applications While the main focus of Parts III and IV was to study the behavior of TCP, we should still consider models of UDP-based applications. The only tools supporting UDP are Swing and RENETO. We note that Harpoon has also some support of UDP, but with the limitation that some predefined models have to be used (constant packet rate and periodic or exponentially distributed ping-pong).

Application-layer agnostic As we noted in Section 9.2, some models and tools are targeted at one specific protocol or class of protocols (ex: HTTP, VoIP). All the tools compared in Table 11.4 use a more generic approach, able to accommodate a broader family of protocols.

Online model update One challenge that we did not address with RENETO is the concept that the empirical analysis of the real traffic can be made online. By using such online analysis, seasonality can be taken into account for instance. The only tool in our comparison which applies this concept is Harpoon.

Bidirectional communications One design principle often used in network protocols is the request/request paradigm, where clients send requests for some file or action, and servers reply to this request. We argue that a realistic model should account for this behavior, as done by all tools compared here except Harpoon.

User modeling Knowing how a protocol is used from a user perspective should give a better overview of protocol use. The model used in Swing and RENETO do take into account user behavior, while the other models (Harpoon, Tmix, LiTGen) work on a per-user view.

Correlation between parameters The models compared in Table 11.4 are based on the selection of different parameters (ex: packet size, inter-arrival time) and the empirical characterization of those parameters using statistical analysis. While most works consider those different parameters to be independent from each other (*i.e.* Harpoon, Swing, Tmix), one interesting concept is to consider a portion of to be correlated as done in LiTGen and RENETO. We showed in Figures 9.12 and 9.13 for instance the benefits of viewing the packet size and the inter-arrival times as correlated parameters.

Traffic generation in real networks or in simulators Being able to reproduce statistically similar traffic in various settings is a useful concept for performance evaluation. Two approaches were used in the models compared in Table 11.4. First, one can generate traffic in real networks as done in Harpoon and Swing, and evaluate protocols in real settings. Secondly, one can also use simulators as done in Tmix (for ns-2) and RENETO (for OMNeT++), giving more freedom with regards to size of the topology or easier troubleshooting for instance. We note that Harpoon has been extended to a flow-level simulator called *fs* [226], but it is more limited than more conventional simulators (*i.e.* ns-2 and OMNeT++). Regarding LiTGen, it is not clear from the publications describing this tool how the traffic generation is made.

Conclusion of the comparison

We note from the comparison made in Table 11.4 that the two most promising approaches and tools fulfilling most of the criteria listed earlier are Swing and our own approach from Chapter 9 (*i.e.* RENETO). The main difference between the two approaches is the concept of using correlation between some parameters of the model, a useful benefit in some cases as shown in Chapter 9. Another distinction between the two approaches is on the traffic generation. On one hand, Swing targets real networks, a valid method when wanting to evaluate performances in a real setting. On the other hand, our method focuses on simulation, as evaluating in a real network is not always possible or desirable due to hardware cost, scalability, or availability.

Regarding the other approaches, each one offers a different benefit. The idea of using an online method used in Harpoon is useful is one wishes to reproduce effects such as trend or seasonality. As for RENETO, LiTGen also proposed the idea of using correlation between some parameters of the model and showed its benefits in various settings. Finally, Tmix is an interesting alternative to RENETO if one wishes to use traffic generation in another simulator (ns-2).

Finally, one last step not discussed in this comparison is how to use those models in one of the analytical frameworks presented in Section 11.2. For TCP-based applications, all models reviewed here are based on the same underlying model, which is the superposition of multiple (unidirectional or bidirectional) ON/OFF flows. Such traffic pattern can generally be studied with the frameworks reviewed in Section 11.2, with eventually some difficulties with regards to the behavior of applications or users.

12. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

This chapter concludes and summarizes the key results of this thesis. We then provide an overview of future research questions in the domain of mathematical frameworks for the performance evaluation of networks with a focus of flow level and its associated guarantees and Quality-of-Service (QoS) characteristics.

12.1 Summary and conclusions

As introduced in Chapter 1, the different research results presented in this thesis were constructed around two objectives.

12.1.1 Research Objective O1: Evaluate mechanisms for Ethernet networks with mixed-criticality

The first research objective addresses the question of mechanisms for enforcing and guaranteeing Quality-of-Service in Ethernet networks, namely packet scheduling algorithms. Research Objective **O1** was divided into three smaller objectives:

O1.1 Investigate the impact of packet scheduling algorithms in the context of avionic networks and identify the ones enabling the mix of real-time with best-effort traffic.

O1.2 Evaluate the benefits of the newly proposed scheduling architecture from the IEEE which address the challenge of mixing audio and video flows with best-effort traffic.

O1.3 Investigate a new packet-forwarding and scheduling architecture for enhancing the performance of periodic real-time traffic mixed with best-effort traffic.

Research Objective O1.1

For Research Objective **O1.1**, various performance evaluations were performed on the two network topologies presented in Sections 2.2.1 and 2.2.2, namely an industrial AFDX network dedicated to real-time traffic and a new Ethernet network for the cabin, dedicated to a mix between real-time, multimedia and best-effort traffic. The method which was used for these evaluations was discrete-event simulations. We found that a good compromise for mixing traffic with different time requirements is to use a hybrid scheduling which treats real-time traffic with priority scheduling and non real-time or elastic with some fair-queuing mechanism to provide a fair bandwidth sharing and eventually class-based QoS. Regarding the separation between the two types of traffic with this hybrid scheduling, we showed that it prevents non real-time traffic from affecting the real-time one in large proportions. Such mechanism is also easily implemented in hardware, with some already existing variations in today's commercial off-the-shelf (COTS) switches and routers.

Research Objective O1.2

For Research Objective **O1.2**, we evaluated a scheduling architecture proposed by the IEEE for mixing audio/video traffic with normal traffic in Ethernet networks using a novel shaper algorithm called *Credit Based Shaper* (CBS). This architecture is based on traditional strict-priority scheduling where audio and video traffic have the two highest priorities but they are shaped in order to avoid starvation. Our aim was to see if using this scheduling architecture could be applied to avionic networks where real-time flows would replace the originally intended audio/video traffic. Using discrete-event simulations, this scheduling architecture was compared to other more common scheduling architecture. We showed that from a performance point of view (*i.e.* end-to-end latency and jitter), the non real-time flows were advantaged compared to other schedulers, but at the price of higher latencies for the real-time one. Despite this fact, thanks to a provable deterministic latency bound as well as its possible widespread deployment in future affordable COTS devices, this scheduling architecture is interesting for an industrial use.

Research Objective O1.3

While we focused for the two previous research objectives on commonly used and non-specialized packet scheduling algorithms, we proposed to investigate for Research Objective **O1.3** a non work-conserving scheduling architecture designed specifically for mixing low end-to-end latencies for real-time traffic with a class-based fair-queuing algorithm. By using such a non work-conserving scheduler, packets can be scheduled in a way that queuing delay can be eliminated, and thus provide the best possible end-to-end latencies for real-time traffic. We used a solution based on a time partitioning schedule (akin to Time-Division Multiple Access – TDMA), where periodic time-slots are reserved for the transmission of real-time traffic. Because the time partitioning has to be shared by all the switches of the network, we used a special master-slave protocol to achieve a simplistic synchronization mechanism while preventing the use of more complex clock synchronization protocol. In combination with this architecture, we proposed the so-called Time-Aware Deficit Round Robin (TADRR) packet scheduling algorithm, an extension of the well-known Deficit Round Robin (DRR). Our algorithm prevents queuing delay during the time slots reserved for real-time traffic, while it works as the traditional DRR during the remaining time-slots for the best-effort traffic, therefor enabling a fair bandwidth sharing among different classes of traffic.

12.1.2 Research Objective O2: Develop mathematical models for network engineering with elastic traffic

We recall here the definition of deterministic communications used in the industry introduced in Chapter 2, namely formally proven performance bounds (end-to-end latency and jitter in the case of guarantees at packet-level) with enforcements in the network. With Research Objective **O1**, we gave some insights on the second part of the definition for deterministic communications, specifically which mechanisms can be used to enforce traffic properties in the network and which impact do they have the performance of the flows. The second research objective addresses the first part of the definition of deterministic communications, namely the question of a mathematical framework for the performance evaluation of networks with real-time and elastic traffic. The goal is to determine which existing analytical model fits our needs for performance evaluation and extend it to support more advanced models. Research Objective **O2** was divided into five smaller objectives:

O2.1 Identify performance evaluation frameworks as well as Quality-of-Service attributes which are relevant for Ethernet networks with elastic traffic such as TCP based flows.

O2.2 Propose an analytical framework which is relevant for the evaluation of networks with UDP and TCP flows at the network layer.

O2.3 Extend this framework to the evaluation of mean performances of short flows.

O2.4 Extend this approach to give statistical guarantees on tail performances.

O2.5 Propose a method to give realistic characteristics to short flows in order to map application layer performances.

Research Objective O2.1

We noted in Section 2.3 that the current mathematical frameworks used in the industry for providing packet guarantees were not adapted to the study of elastic and TCP-based traffic. The goal of Research Objective **O2.1** was then to investigated which performance evaluation framework would be the most suitable for network with real-time and elastic traffic with an emphasis on the evaluation of TCP-based traffic. Various analytical frameworks were considered in Chapter 6 and compared against a categorization proposed in Section 6.3.1. The following approaches were compared: *(i)* Network utility maximization; *(ii)* Fluid models with stochastic differential equations; *(iii)* Queuing theory based models; *(iv)* Multi-layer fixed-point models. We established that the most promising approach is the multi-layer fixed-point models as it fits multiple key points for studying Ethernet networks and it provides a promising inclination regarding extension.

A second aim of Research Objective **O2.1** was to determine which performance attributes are relevant for elastic flows. It was established that:

Packet and flow-level view Traditional packet-level guarantees such as end-to-end delay and jitter are less relevant for elastic traffic. The notion of flow, namely a group of packet forming application-layer message or even group of flow, is more important as it related to application layer requirements.

Key attributes One key Quality-of-Service characteristic for elastic traffic is the duration needed to transmit a message or duration to perform a request-reply exchange.

Research Objective O2.2

Following the results of Research Objective **O2.1** and the evaluation of various approaches for the performance evaluation of networks with TCP, Research Objective **O2.2** was dedicated to formalize the multi-layer fixed-point models approach and fill some missing characteristics. This approach is based on early models of the bandwidth of TCP flows as a function of latency and drop probability. Using those models in conjunction with models of queues characterizing drop probability and latency as a function of the traversed flows, complete networks with multiple bottlenecks can be evaluated. The results of a performance evaluation with this approach are: mean bandwidth of infinite TCP and UDP flows, average queue size, average drop probability and link utilization.

Among our contributions for this research objective are:

Ethernet networks specific characteristics We provided and formalized a framework and methods for the performance evaluation of Ethernet networks using the multi-layer fixed-point models approach. Due to some characteristics of the studied Ethernet networks, namely importance of queuing delay and switches using FIFO queues, simple or hybrid packet scheduling algorithms, one of our contributions was to adapt and extend existing models for providing accurate results on our use-cases.

Modeling of unexpected behavior of TCP We proposed a method to model a non-intuitive behavior of TCP historically known as "ACK compression". Instead of modeling TCP flows as unidirectional – as mostly done in related works – we took into account TCP acknowledgements and their influence on the bandwidth of the data packets. By doing so, we increased the accuracy of the method.

Performance evaluation tool for this analytical framework After conducting a survey on tools available for the evaluation of networks using the multi-layer fixed-point models approach, we established that there was a lack of tools taking into account the specificities of Ethernet networks (*i.e.* importance of queuing delay). Hence we proposed our own tool called PETFEN – Performance Evaluation Tool for Flow-level network modeling of Ethernet Networks – implementing the formalism and contributions made for this research objective. This tool and our global approach was then compared against another similar and its was shown that our tool produced more accurate results.

Research Objective O2.3

We noted in Research Objective **O2.1** that one shortcoming of the performance evaluation framework we chose was that it was limited to infinite flows. As this does not actually match real protocol behavior, one of our contributions presented in Chapter 8 was to propose a stochastic extension of flow-level network modeling in order to be able to study short flows. Compared to related approaches, one advantage of using the one developed in Chapter 8 is that we can take advantage of the results from Research Objective **O2.2** in order to have realistic models of TCP bandwidth sharing as well as extended models of network elements like packet scheduling for instance.

The results of a performance evaluation with this extended stochastic approach are the same as the non-stochastic one, *i.e.* mean bandwidth of TCP and UDP flows, average queue size, average drop probability and link utilization, with the extension of average time to complete the transmission of a file for short flows and average number of active flows.

For this purpose we used a simple active/idle or ON/OFF model of a flow. The ON phase corresponds to the full transmission of a file, with a size following a general random distribution with known mean. The OFF phase corresponds to an idle phase of duration following a general random distribution with known mean, representing for instance the processing of the file.

Among our contributions for this research objective are:

Stochastic extension of flow-level modeling We proposed methods to leverage the results of flow-level network modeling developed in Research Objective **O2.2** in order to study short UDP and TCP flows. Due to those existing results, the methods we proposed for this research objective are able to use realistic bandwidth sharing models, on networks with multiple bottlenecks, and with various packet scheduling algorithms.

Extension to short TCP flow with slow-start In order to increase the accuracy of our model for ON/OFF TCP flows where slow-start plays an important role, an extension of our framework was proposed using a well-established model of TCP slow-start. We compared with a numerical evaluation our simple and advanced model, and established when the increase on accuracy of this advanced model outweighs the computational complexity.

Extension to bidirectional TCP flows and first step toward application level We also proposed an extension of our stochastic ON/OFF model to study network protocols based on the bidirectional request-reply paradigm. This extension is useful for numerous protocols used in practice such as HTTP, DNS, IMAP, POP, or FTP.

Research Objective O2.4

We developed in Research Objective **O2.3** methods and a mathematical framework for the performance evaluation of active/idle or ON/OFF TCP and UDP flows, in order to characterize mean performances, such as mean time to transfer a file. While such mean performance characterization might be enough for services without strict requirements (like best-effort or soft real-time), we proposed with Research Objective **O2.4** to characterize tail performances. Using such characterization, more stringent applications can be used, as requirement can fulfilled with a certain probability, generally in the order of 99.999 % to 99.999 999 999 %.

In order to achieve such performance characterizations, we used stochastic network calculus. The approach we proposed for evaluating TCP flow is as follows. Since we are not interested in packet-level performances, we proposed to model the bandwidth sharing of TCP using Generalized Processor Sharing (GPS). Flows are characterized using the generalized Stochastically Bounded Burtiness (gSBB). We modeled TCP flows as Lévy processes, where the increments in the stochastic processes correspond to application layer messages.

Leveraging on the results from Research Objectives **O2.2** and **O2.3**, we proposed a method based on the results from Chapters 7 and 8 for assigning weights to the GPS servers according to realistic models of TCP.

Among our contributions for this research objective are:

Extension of stochastic bound for GPS and PGPS service discipline We proposed a new stochastic bound on session following a gSBB characterization served by a server with constant rate with GPS or PGPS service discipline in Chapter 10.

Stochastic bound for hybrid priority and GPS scheduling We extended the bound previously described to the analytical study of hybrid scheduling as presented in Chapter 4. This extension is valuable for network engineering as such scheduling architecture is often present in COTS switches under the name low-latency queuing.

Application to TCP bandwidth sharing We applied this bound to TCP in order to get bounds on transfer time. We propose a simple characterization of application layer behavior based on a Lévy process. We also provided a method leveraging on the work from Research Objectives **O2.2** and **O2.3** for assigning the GPS weights following realistic TCP bandwidth sharing.

Research Objective O2.5

The performance evaluation frameworks developed in Chapters 7, 8 and 10 are inherently based on two simple views of network protocols, namely a first one where flows always have data to send (for Chapter 7), and a second one where flows alternate between active and idle periods of random durations in an ON/OFF pattern (for Chapter 8). Some extensions of the ON/OFF flow model were proposed in Chapter 8 to approach a more realistic view of network protocols, but a key challenge is still present: which statistical distributions and parameters should be given to the model in order to have realistic models.

Hence, one of our contributions for Research Objective **O2.5** was to propose an extended model of network protocol behavior and a method of how to set the parameters of the model by analyzing network captures. The model is based on a layered view of network protocols, similar to the OSI layer, with messages, flows, sessions and users layers. Each layer can be viewed with an ON/OFF behavior, where the time between two active phases and the number of components are described by general random variables. The distributions of those random variables are then characterized by analyzing network captures using a tool we proposed.

Among our contributions for this research objective are:

Model of network protocols We proposed for this research objective a simple mathematical model able to characterize realistic network protocols and traffic based on TCP and UDP. Our model is based on four layers: messages, flows, sessions and users.

Importance of correlation in the model One contribution for Research Objective **O2.5** was to show that some protocols have correlated parameters which have to be taken into account for increasing the accuracy of the method. In the case of UDP traffic, we showed that packet size and packet inter-arrival time are correlated in case of SNMP traffic.

Tool for extracting the parameters from a trace We proposed a tool able to extract the statistical distributions of the different random variables of the model of specific protocols based on the analysis of network captures.

Tool for reproducing the model in simulations In order to show if our protocol model and empirical statistical distributions are able to reproduce realistic network traffic, we also proposed algorithms and their associated implementation for reproducing our protocol model inside a discrete-event simulator.

12.2 Future research directions

We studied in this thesis various methods and mathematical frameworks for the performance evaluation of elastic flows with a focus on flow-layer performances and evaluated various mechanisms used in switches for enforcing performance requirements. Yet several open research questions are still open. We already mentioned throughout this thesis various drawbacks of the developed methods which could be addressed, as well as improvements and possible extensions. We focus in the rest of this section on broader open research questions.

Towards stochastic flow-level modeling and application-level performances We developed in this thesis various analytical models for the performance evaluation of infinite and short flows. Yet, those models are mostly based on simplistic views of traffic behavior, namely a simple unidirectional active/idle or ON/OFF pattern (with some proposed extensions), which is not always straightforward to map to real network protocols. While it is understandable from a pure mathematical perspective to keep those models simple in order to get tractable results, network engineers often need more advanced behaviors, due to more elaborate patterns or interaction and synchronization between different applications in the network. Getting one layer above for instance, *i.e.* going toward application-level performances, would benefit network dimensioning methods and applications to real use-cases. A parallel with the gap between Quality-of-Service and Quality-of-Experience can also be made here, where the former focuses on quantitative packets and flows performances, while the latter targets more qualitative end-user perceptions which might not always be correlated to the former.

A deterministic TCP variant We studied in this thesis various approaches to characterize the performance of TCP-based traffic with the assumption that TCP performs in a certain predictable way. Yet from a purely practical point of view, this predictable way is not always accurate due to non-intuitive behaviors of TCP which are often neglected in performance evaluation frameworks. We treated one example of such behavior in Chapter 7 with the impact of cross-traffic, but there are other ones like for instance Incast which we did not treat here. One approach to address this challenge would be to develop new congestion control algorithms having a provable and deterministic behavior, and where the interaction and bandwidth sharing between multiple flows would also be provable and deterministic.

A standardized flow-based network architecture We focused in this thesis on flow-level requirements with statistical bandwidth sharing. In order to get adoption of such view of requirements by the industry, one needs enforcement as noted in our definition of *deterministic* in Section 2.2.1. One example would be to use Packetized Generalized Processor Sharing (PGPS) scheduling with support of numerous flows, which cannot be found in today's COTS devices. While such enforcement can be implemented in custom-made hardware, one lever to popularize it and find it in more COTS equipment would be to have a standardized platform. Such flow-level platform has also been advocated by various researchers in the past with also some solutions, but without global adoption. We note that the IntServ architecture was a proposition for a similar flow-based platform, but it did not get enough traction due to scalability issues. A similar problem is present nowadays with packet-level guarantees, but this challenge is being actively addressed by the IEEE with its Audio/Video Bridging (AVB) and Time-Sensitive Networking (TSN) technologies. With the current rise of Software Defined Networking (SDN) defining a flexible platform with common specifications for packet forwarding and flow handling, one could envision a “Software Defined Quality-of-Service” for building responsive flow-level policies.

APPENDIX

A. LIST OF PUBLICATIONS

Publications of the author related to this dissertation are subsequently listed in the chronological order of the date of their appearance.

[120] **Fabien Geyer**, Stefan Schneele, and Georg Carle. RENETO, a Realistic Network Traffic Generator for OMNeT++/INET. In *Proceedings of the 6th International Conference on Simulation Tools and Techniques*, SIMUTOOLS 2013, pages 73–81, March 2013. ISBN 978-1-4503-2464-9. doi:10.4108/icst.simutools.2013.251697

[122] **Fabien Geyer**, Stefan Schneele, Matthias Heinisch, and Peter Klose. Simulation and Performance Evaluation of an Aircraft Cabin Network Node. In Frank Thielecke Otto von Estorff, editor, *Proceedings of the 4th International Workshop on Aircraft System Technologies*, AST 2013, pages 323–332. Shaker-Verlag, April 2013. ISBN 978-3-8440-1850-9

[119] **Fabien Geyer**, Emanuel Heidinger, Stefan Schneele, and Alexander von Bodisco. Evaluation of Audio/Video Bridging Forwarding Method in an Avionics Switched Ethernet Context. In *Proceedings of the 18th IEEE Symposium on Computers and Communications*, ISCC 2013, pages 711–716. IEEE, July 2013. doi:10.1109/ISCC.2013.6755032

[121] **Fabien Geyer**, Stefan Schneele, and Georg Carle. Practical Performance Evaluation of Ethernet Networks with Flow-Level Network Modeling. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS 2013, pages 253–262, December 2013. doi:10.4108/icst.valuetools.2013.254367

[125] **Fabien Geyer**, Stefan Schneele, and Wolfgang Fischer. Performance Evaluation of an Ethernet-Based Cabin Network Architecture Supporting a Low-Latency Service. In *Proceedings of the 6th International Workshop Nets4Cars/Nets4Trains/Nets4Aircraft*, LNCS 8435, pages 69–80. Springer, May 2014. doi:10.1007/978-3-319-06644-8_7

[123] **Fabien Geyer**, Stefan Schneele, and Georg Carle. Towards Stochastic Flow-Level Network Modeling: Performance Evaluation of Short TCP Flows. In *Proceedings of the 39th IEEE Conference on Local Computer Networks*, LCN 2014, pages 462–465. IEEE, September 2014. doi:10.1109/LCN.2014.6925817

[124] **Fabien Geyer**, Stefan Schneele, and Georg Carle. PETFEN: A Performance Evaluation Tool for Flow-Level Network Modeling of Ethernet Networks. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS 2014, December 2014. doi:10.4108/icst.valuetools.2014.258166

B. TOPOLOGIES DETAILS

For reproducibility purpose, we present here the details of the four random topologies depicted in Figure 7.19 and used for the numerical evaluation in Sections 7.8.5 and 7.8.6. Those topologies were generated according to Algorithm 7.7 and follow the PETFEN topology description format presented in Section 7.7.3.

Listing B.1: Full description of the topology presented in Figure 7.19a

```
; -- Nodes definition
(node N00) (node N01) (node N02) (node N03) (node N04) (node N05) (node N06)
(node N07) (node N08) (node N09) (node N10) (node N11) (node N12) (node N13)
(node N14) (node N15) (node N16) (node N17) (node N18) (node N19) (node N20)
(node N21) (node N22) (node N23) (node N24) (node N25) (node N26) (node N27)
(node N28) (node N29) (node N30) (node N31) (node N32) (node N33) (node N34)
(node N35) (node N36) (node N37) (node N38) (node N39) (node N40) (node N41)
(node N42) (node N43) (node N44) (node N45) (node N46) (node N47) (node N48)
(node N49) (node N50) (node N51) (node N52) (node N53) (node N54) (node N55)
(node N56) (node N57) (node N58) (node N59) (node N60) (node N61) (node N62)
(node N63) (node N64) (node N65) (node N66) (node N67) (node N68) (node N69)
(node N70) (node N71) (node N72) (node N73) (node N74) (node N75) (node N76)
(node N77) (node N78) (node N79) (node N80) (node N81) (node N82) (node N83)
(node N84) (node N85) (node N86)

; -- Links definition
(link N00 N01 (Mbps 100)) (link N00 N19 (Mbps 100)) (link N00 N25 (Mbps 100))
(link N00 N26 (Mbps 100)) (link N00 N32 (Mbps 100)) (link N01 N02 (Mbps 100))
(link N01 N03 (Mbps 100)) (link N01 N11 (Mbps 100)) (link N01 N12 (Mbps 100))
(link N01 N18 (Mbps 100)) (link N03 N04 (Mbps 100)) (link N03 N05 (Mbps 100))
(link N03 N06 (Mbps 100)) (link N03 N07 (Mbps 100)) (link N03 N08 (Mbps 100))
(link N03 N09 (Mbps 100)) (link N03 N10 (Mbps 100)) (link N12 N13 (Mbps 100))
(link N12 N14 (Mbps 100)) (link N12 N15 (Mbps 100)) (link N12 N16 (Mbps 100))
(link N12 N17 (Mbps 100)) (link N19 N20 (Mbps 100)) (link N19 N21 (Mbps 100))
(link N19 N22 (Mbps 100)) (link N19 N23 (Mbps 100)) (link N19 N24 (Mbps 100))
(link N26 N27 (Mbps 100)) (link N26 N28 (Mbps 100)) (link N26 N29 (Mbps 100))
(link N26 N30 (Mbps 100)) (link N26 N31 (Mbps 100)) (link N32 N33 (Mbps 100))
(link N32 N34 (Mbps 100)) (link N32 N42 (Mbps 100)) (link N32 N75 (Mbps 100))
(link N32 N81 (Mbps 100)) (link N34 N35 (Mbps 100)) (link N34 N36 (Mbps 100))
(link N34 N37 (Mbps 100)) (link N34 N38 (Mbps 100)) (link N34 N39 (Mbps 100))
(link N34 N40 (Mbps 100)) (link N34 N41 (Mbps 100)) (link N42 N43 (Mbps 100))
(link N42 N51 (Mbps 100)) (link N42 N52 (Mbps 100)) (link N42 N53 (Mbps 100))
(link N42 N54 (Mbps 100)) (link N42 N61 (Mbps 100)) (link N42 N69 (Mbps 100))
(link N43 N44 (Mbps 100)) (link N43 N45 (Mbps 100)) (link N43 N46 (Mbps 100))
(link N43 N47 (Mbps 100)) (link N43 N48 (Mbps 100)) (link N43 N49 (Mbps 100))
(link N43 N50 (Mbps 100)) (link N54 N55 (Mbps 100)) (link N54 N56 (Mbps 100))
(link N54 N57 (Mbps 100)) (link N54 N58 (Mbps 100)) (link N54 N59 (Mbps 100))
(link N54 N60 (Mbps 100)) (link N61 N62 (Mbps 100)) (link N61 N63 (Mbps 100))
(link N61 N64 (Mbps 100)) (link N61 N65 (Mbps 100)) (link N61 N66 (Mbps 100))
(link N61 N67 (Mbps 100)) (link N61 N68 (Mbps 100)) (link N69 N70 (Mbps 100))
(link N69 N71 (Mbps 100)) (link N69 N72 (Mbps 100)) (link N69 N73 (Mbps 100))
```

```
(link N69 N74 (Mbps 100)) (link N75 N76 (Mbps 100)) (link N75 N77 (Mbps 100))
(link N75 N78 (Mbps 100)) (link N75 N79 (Mbps 100)) (link N75 N80 (Mbps 100))
(link N81 N82 (Mbps 100)) (link N81 N83 (Mbps 100)) (link N81 N84 (Mbps 100))
(link N81 N85 (Mbps 100)) (link N81 N86 (Mbps 100))

; -- Flows definition
(tcpflow N02:F0 N02 N57) (tcpflow N04:F0 N04 N08) (tcpflow N05:F0 N05 N23)
(tcpflow N06:F0 N06 N13) (tcpflow N07:F0 N07 N44) (tcpflow N08:F0 N08 N59)
(tcpflow N09:F0 N09 N22) (tcpflow N10:F0 N10 N84) (tcpflow N11:F0 N11 N58)
(tcpflow N13:F0 N13 N74) (tcpflow N14:F0 N14 N79) (tcpflow N15:F0 N15 N30)
(tcpflow N16:F0 N16 N55) (tcpflow N17:F0 N17 N11) (tcpflow N18:F0 N18 N08)
(tcpflow N20:F0 N20 N51) (tcpflow N21:F0 N21 N46) (tcpflow N22:F0 N22 N25)
(tcpflow N23:F0 N23 N11) (tcpflow N24:F0 N24 N16) (tcpflow N25:F0 N25 N63)
(tcpflow N27:F0 N27 N68) (tcpflow N28:F0 N28 N78) (tcpflow N29:F0 N29 N51)
(tcpflow N30:F0 N30 N38) (tcpflow N31:F0 N31 N52) (tcpflow N33:F0 N33 N72)
(tcpflow N35:F0 N35 N11) (tcpflow N36:F0 N36 N84) (tcpflow N37:F0 N37 N63)
(tcpflow N38:F0 N38 N70) (tcpflow N39:F0 N39 N64) (tcpflow N40:F0 N40 N51)
(tcpflow N41:F0 N41 N49) (tcpflow N44:F0 N44 N14) (tcpflow N45:F0 N45 N38)
(tcpflow N46:F0 N46 N44) (tcpflow N47:F0 N47 N83) (tcpflow N48:F0 N48 N39)
(tcpflow N49:F0 N49 N33) (tcpflow N50:F0 N50 N38) (tcpflow N51:F0 N51 N82)
(tcpflow N52:F0 N52 N35) (tcpflow N53:F0 N53 N47) (tcpflow N55:F0 N55 N67)
(tcpflow N56:F0 N56 N86) (tcpflow N57:F0 N57 N36) (tcpflow N58:F0 N58 N16)
(tcpflow N59:F0 N59 N49) (tcpflow N60:F0 N60 N06) (tcpflow N62:F0 N62 N20)
(tcpflow N63:F0 N63 N20) (tcpflow N64:F0 N64 N25) (tcpflow N65:F0 N65 N36)
(tcpflow N66:F0 N66 N82) (tcpflow N67:F0 N67 N52) (tcpflow N68:F0 N68 N33)
(tcpflow N70:F0 N70 N47) (tcpflow N71:F0 N71 N41) (tcpflow N72:F0 N72 N28)
(tcpflow N73:F0 N73 N08) (tcpflow N74:F0 N74 N23) (tcpflow N76:F0 N76 N22)
(tcpflow N77:F0 N77 N68) (tcpflow N78:F0 N78 N51) (tcpflow N79:F0 N79 N18)
(tcpflow N80:F0 N80 N68) (tcpflow N82:F0 N82 N58) (tcpflow N83:F0 N83 N05)
(tcpflow N84:F0 N84 N10) (tcpflow N85:F0 N85 N17) (tcpflow N86:F0 N86 N78)
```

Listing B.2: Full description of the topology presented in Figure 7.19b

```
; -- Nodes definition
(node N00) (node N01) (node N02) (node N03) (node N04) (node N05) (node N06)
(node N07) (node N08) (node N09) (node N10) (node N11) (node N12) (node N13)
(node N14) (node N15) (node N16) (node N17) (node N18) (node N19) (node N20)
(node N21) (node N22) (node N23) (node N24) (node N25) (node N26) (node N27)
(node N28) (node N29) (node N30) (node N31) (node N32) (node N33) (node N34)
(node N35) (node N36) (node N37)

; -- Links definition
(link N00 N01 (Mbps 100)) (link N00 N02 (Mbps 100)) (link N00 N27 (Mbps 100))
(link N00 N33 (Mbps 100)) (link N02 N03 (Mbps 100)) (link N02 N09 (Mbps 100))
(link N02 N15 (Mbps 100)) (link N02 N16 (Mbps 100)) (link N02 N17 (Mbps 100))
(link N02 N25 (Mbps 100)) (link N02 N26 (Mbps 100)) (link N03 N04 (Mbps 100))
(link N03 N05 (Mbps 100)) (link N03 N06 (Mbps 100)) (link N03 N07 (Mbps 100))
(link N03 N08 (Mbps 100)) (link N09 N10 (Mbps 100)) (link N09 N11 (Mbps 100))
(link N09 N12 (Mbps 100)) (link N09 N13 (Mbps 100)) (link N09 N14 (Mbps 100))
(link N17 N18 (Mbps 100)) (link N17 N19 (Mbps 100)) (link N17 N20 (Mbps 100))
(link N17 N21 (Mbps 100)) (link N17 N22 (Mbps 100)) (link N17 N23 (Mbps 100))
(link N17 N24 (Mbps 100)) (link N27 N28 (Mbps 100)) (link N27 N29 (Mbps 100))
(link N27 N30 (Mbps 100)) (link N27 N31 (Mbps 100)) (link N27 N32 (Mbps 100))
(link N33 N34 (Mbps 100)) (link N33 N35 (Mbps 100)) (link N33 N36 (Mbps 100))
(link N33 N37 (Mbps 100))

; -- Flows definition
(tcpflow N01:F0 N01 N08) (tcpflow N04:F0 N04 N35) (tcpflow N05:F0 N05 N13)
```

```
(tcpflow N06:F0 N06 N36) (tcpflow N07:F0 N07 N36) (tcpflow N08:F0 N08 N36)
(tcpflow N10:F0 N10 N13) (tcpflow N11:F0 N11 N29) (tcpflow N12:F0 N12 N06)
(tcpflow N13:F0 N13 N04) (tcpflow N14:F0 N14 N34) (tcpflow N15:F0 N15 N04)
(tcpflow N16:F0 N16 N15) (tcpflow N18:F0 N18 N31) (tcpflow N19:F0 N19 N36)
(tcpflow N20:F0 N20 N23) (tcpflow N21:F0 N21 N20) (tcpflow N22:F0 N22 N24)
(tcpflow N23:F0 N23 N04) (tcpflow N24:F0 N24 N29) (tcpflow N25:F0 N25 N13)
(tcpflow N26:F0 N26 N18) (tcpflow N28:F0 N28 N35) (tcpflow N29:F0 N29 N14)
(tcpflow N30:F0 N30 N10) (tcpflow N31:F0 N31 N35) (tcpflow N32:F0 N32 N34)
(tcpflow N34:F0 N34 N26) (tcpflow N35:F0 N35 N25) (tcpflow N36:F0 N36 N07)
(tcpflow N37:F0 N37 N26)
```

Listing B.3: Full description of the topology presented in Figure 7.19c

```
; -- Nodes definition
(node N00) (node N01) (node N02) (node N03) (node N04) (node N05) (node N06)
(node N07) (node N08) (node N09) (node N10) (node N11) (node N12) (node N13)
(node N14) (node N15) (node N16) (node N17) (node N18) (node N19) (node N20)
(node N21) (node N22) (node N23) (node N24) (node N25) (node N26) (node N27)
(node N28) (node N29) (node N30) (node N31) (node N32) (node N33) (node N34)
(node N35) (node N36) (node N37) (node N38) (node N39) (node N40) (node N41)
(node N42) (node N43) (node N44) (node N45) (node N46) (node N47) (node N48)
(node N49) (node N50) (node N51) (node N52) (node N53) (node N54) (node N55)

; -- Links definition
(link N00 N01 (Mbps 100)) (link N00 N19 (Mbps 100)) (link N00 N26 (Mbps 100))
(link N00 N27 (Mbps 100)) (link N00 N34 (Mbps 100)) (link N01 N02 (Mbps 100))
(link N01 N03 (Mbps 100)) (link N01 N09 (Mbps 100)) (link N01 N10 (Mbps 100))
(link N01 N11 (Mbps 100)) (link N01 N18 (Mbps 100)) (link N03 N04 (Mbps 100))
(link N03 N05 (Mbps 100)) (link N03 N06 (Mbps 100)) (link N03 N07 (Mbps 100))
(link N03 N08 (Mbps 100)) (link N11 N12 (Mbps 100)) (link N11 N13 (Mbps 100))
(link N11 N14 (Mbps 100)) (link N11 N15 (Mbps 100)) (link N11 N16 (Mbps 100))
(link N11 N17 (Mbps 100)) (link N19 N20 (Mbps 100)) (link N19 N21 (Mbps 100))
(link N19 N22 (Mbps 100)) (link N19 N23 (Mbps 100)) (link N19 N24 (Mbps 100))
(link N19 N25 (Mbps 100)) (link N27 N28 (Mbps 100)) (link N27 N29 (Mbps 100))
(link N27 N30 (Mbps 100)) (link N27 N31 (Mbps 100)) (link N27 N32 (Mbps 100))
(link N27 N33 (Mbps 100)) (link N34 N35 (Mbps 100)) (link N34 N45 (Mbps 100))
(link N34 N46 (Mbps 100)) (link N34 N47 (Mbps 100)) (link N35 N36 (Mbps 100))
(link N35 N37 (Mbps 100)) (link N35 N38 (Mbps 100)) (link N35 N39 (Mbps 100))
(link N35 N40 (Mbps 100)) (link N40 N41 (Mbps 100)) (link N40 N42 (Mbps 100))
(link N40 N43 (Mbps 100)) (link N40 N44 (Mbps 100)) (link N47 N48 (Mbps 100))
(link N47 N49 (Mbps 100)) (link N47 N50 (Mbps 100)) (link N47 N51 (Mbps 100))
(link N51 N52 (Mbps 100)) (link N51 N53 (Mbps 100)) (link N51 N54 (Mbps 100))
(link N51 N55 (Mbps 100))

; -- Flows definition
(tcpflow N02:F0 N02 N26) (tcpflow N04:F0 N04 N14) (tcpflow N05:F0 N05 N45)
(tcpflow N06:F0 N06 N22) (tcpflow N07:F0 N07 N10) (tcpflow N08:F0 N08 N30)
(tcpflow N09:F0 N09 N06) (tcpflow N10:F0 N10 N33) (tcpflow N12:F0 N12 N20)
(tcpflow N13:F0 N13 N06) (tcpflow N14:F0 N14 N09) (tcpflow N15:F0 N15 N54)
(tcpflow N16:F0 N16 N12) (tcpflow N17:F0 N17 N31) (tcpflow N18:F0 N18 N05)
(tcpflow N20:F0 N20 N33) (tcpflow N21:F0 N21 N24) (tcpflow N22:F0 N22 N31)
(tcpflow N23:F0 N23 N22) (tcpflow N24:F0 N24 N45) (tcpflow N25:F0 N25 N31)
(tcpflow N26:F0 N26 N48) (tcpflow N28:F0 N28 N38) (tcpflow N29:F0 N29 N30)
(tcpflow N30:F0 N30 N09) (tcpflow N31:F0 N31 N55) (tcpflow N32:F0 N32 N28)
(tcpflow N33:F0 N33 N15) (tcpflow N36:F0 N36 N15) (tcpflow N37:F0 N37 N14)
(tcpflow N38:F0 N38 N37) (tcpflow N39:F0 N39 N05) (tcpflow N41:F0 N41 N17)
(tcpflow N42:F0 N42 N41) (tcpflow N43:F0 N43 N23) (tcpflow N44:F0 N44 N53)
(tcpflow N45:F0 N45 N42) (tcpflow N46:F0 N46 N28) (tcpflow N48:F0 N48 N50)
```

```
(tcpflow N49:F0 N49 N50) (tcpflow N50:F0 N50 N32) (tcpflow N52:F0 N52 N32)
(tcpflow N53:F0 N53 N32) (tcpflow N54:F0 N54 N42) (tcpflow N55:F0 N55 N28)
```

Listing B.4: Full description of the topology presented in Figure 7.19d

```
; -- Nodes definition
(node N00) (node N01) (node N02) (node N03) (node N04) (node N05) (node N06)
(node N07) (node N08) (node N09) (node N10) (node N11) (node N12) (node N13)
(node N14) (node N15) (node N16) (node N17) (node N18) (node N19) (node N20)
(node N21) (node N22) (node N23) (node N24) (node N25) (node N26) (node N27)
(node N28) (node N29) (node N30) (node N31) (node N32) (node N33)

; -- Links definition
(link N00 N01 (Mbps 100)) (link N00 N09 (Mbps 100)) (link N00 N10 (Mbps 100))
(link N00 N18 (Mbps 100)) (link N00 N19 (Mbps 100)) (link N01 N02 (Mbps 100))
(link N01 N03 (Mbps 100)) (link N01 N04 (Mbps 100)) (link N01 N05 (Mbps 100))
(link N01 N06 (Mbps 100)) (link N01 N07 (Mbps 100)) (link N01 N08 (Mbps 100))
(link N10 N11 (Mbps 100)) (link N10 N12 (Mbps 100)) (link N10 N13 (Mbps 100))
(link N10 N14 (Mbps 100)) (link N10 N15 (Mbps 100)) (link N10 N16 (Mbps 100))
(link N10 N17 (Mbps 100)) (link N19 N20 (Mbps 100)) (link N19 N21 (Mbps 100))
(link N19 N28 (Mbps 100)) (link N19 N29 (Mbps 100)) (link N21 N22 (Mbps 100))
(link N21 N23 (Mbps 100)) (link N21 N24 (Mbps 100)) (link N21 N25 (Mbps 100))
(link N21 N26 (Mbps 100)) (link N21 N27 (Mbps 100)) (link N29 N30 (Mbps 100))
(link N29 N31 (Mbps 100)) (link N29 N32 (Mbps 100)) (link N29 N33 (Mbps 100))

; -- Flows definition
(tcpflow N02:F0 N02 N25) (tcpflow N03:F0 N03 N23) (tcpflow N04:F0 N04 N08)
(tcpflow N05:F0 N05 N13) (tcpflow N06:F0 N06 N27) (tcpflow N07:F0 N07 N28)
(tcpflow N08:F0 N08 N11) (tcpflow N09:F0 N09 N08) (tcpflow N11:F0 N11 N07)
(tcpflow N12:F0 N12 N26) (tcpflow N13:F0 N13 N32) (tcpflow N14:F0 N14 N03)
(tcpflow N15:F0 N15 N12) (tcpflow N16:F0 N16 N30) (tcpflow N17:F0 N17 N22)
(tcpflow N18:F0 N18 N08) (tcpflow N20:F0 N20 N23) (tcpflow N22:F0 N22 N14)
(tcpflow N23:F0 N23 N03) (tcpflow N24:F0 N24 N04) (tcpflow N25:F0 N25 N15)
(tcpflow N26:F0 N26 N28) (tcpflow N27:F0 N27 N05) (tcpflow N28:F0 N28 N33)
(tcpflow N30:F0 N30 N15) (tcpflow N31:F0 N31 N12) (tcpflow N32:F0 N32 N13)
(tcpflow N33:F0 N33 N23)
```

C. RENETO TOOL

C.1 RENETO XML-based file format

Because our approach is based on two distinct tools, we need an exchange file format between the two tools, which was based on Extensible Markup Language (XML).

The XML description of a RENETO traffic model is described in Listing C.1. For each parameter presented in Table 9.1, we define a tag and save the inverse of the empirical distribution function of the parameter.

Listing C.1: XML file format used in RENETO

```
<?xml version="1.0"?>
<reneto_model ...>
    <numSession>...</numSession>
    <interSession>...</interSession>
    <numFlow>...</numFlow>
    <interFlow>...</interFlow>
    <numPairs>...</numPairs>
    <reqSize>...</reqSize>
    <respSize>...</respSize>
    <interReq>...</interReq>
    <interResp>...</interResp>
    <reqSize_interReq>...</reqSize_interReq>
    <respSize_interResp>...</respSize_interResp>
</reneto_model>
```

D. GLOSSARY AND NOTATIONS

Definitions noted in italic mean that it is an acronym.

Mathematical notations and terms

$\mathbb{E}[X]$	Expected value (or also mean or first moment) of random variable X
$M_X(t)$	Moment-generating function of the random variable X evaluated at $t \in \mathbb{R}^+$
$[x]^+$	$\max(x, 0), \forall x \in \mathbb{R}$
$[x]_1$	$\min(x, 1), \forall x \in \mathbb{R}$
$\lvert\mathcal{X}\rvert$	Cardinally or number of elements of the set $\mathcal{X}$
μ_X	Arithmetic mean of the set X: $\mu_X = {}^1\!/\!_{\lvert X\rvert} \sum_i x_i$
s_X^2	Unbiased estimate of the variance of the set X: $s_X^2 = \frac{1}{\lvert X\rvert-1} \sum_i (x_i - \mu_X)^2$
$\lfloor x \rfloor$	Floor function: largest integer not greater than x
$\lceil x \rceil$	Ceiling function: smallest integer not less than x
$\mathbb{I}\{x\}$	Indicator function: 1 if x is true and 0 otherwise
CDF	*Cumulative Distribution Function*
ECDF	*Empirical Cumulative Distribution Function*
ICDF	*Inverse Cumulative Distribution Function*
IECDF	*Inverse Empirical Cumulative Distribution Function*
LDE	*Logscale Diagram Estimate*
LRD	*Long Range Dependency*

Variable names commonly used through the thesis

C	Link capacity (in bit/s)
F_i	Flow i

Network and computer related terms

AIMD	*Additive Increase Multiplicative Decrease*
AVB	*Audio/Video Bridging*
AQM	*Active Queue Management*
BE	*Best Effort*

CBS	*Credit Based Shaper*, defined in the IEEE 802.1Qav standard [2]
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
DiffServ	*Differentiated Services*
DNC	*Deterministic Network Calculus*
DRR	*Deficit Round Robin*
FCFS	*First Come First Served*
FIFO	*First In First Out*
FQ	*Fair Queuing*
FTP	*File Transfer Protocol*
GPS	*Generalized Processor Sharing*
HTTP	*Hypertext Transfer Protocol*
HTML	*HyperText Markup Language*
IEEE	*Institute of Electrical and Electronics Engineers*
IFG	*Inter-Frame Gap* (Ethernet)
IntServ	*Integrated Services*
LAN	*Local Area Network*
MAC	*Medium Access Control*
MSRP	*Multiple Stream Reservation Protocol*
OSI	*Open Systems Interconnection*
PCP	*Priority Code Point*, as defined the IEEE 802.1Q standard
PGPS	*Packetized Generalized Processor Sharing*
PHY	*Physical Layer* of the OSI model
QoS	*Quality-of-Service*
RED	*Random Early Detection*
RR	*Round Robin*
RSVP	*Resource Reservation Protocol*
RT	*Real-Time*
RTT	*Round-Trip Time*
SCFQ	*Self-Clocked Fair Queuing*
SFD	*Start of Frame Delimiter* (Ethernet)
SPQ	*Strict Priority Queuing*
SMTP	*Simple Mail Transfer Protocol*
SNC	*Stochastic Network Calculus*

SNMP	*Simple Network Management Protocol*
STP	*Spanning Tree Protocol*
TADRR	*Time Aware Deficit Round Robin*
TCP	*Transport Control Protocol*
TDMA	*Time-Division Multiple Access*
TSN	*Time Sensitive Networking*
UDP	*User Datagram Protocol*
VoIP	*Voice over IP*
VLAN	*Virtual Local Area Network* (IEEE 802.1Q)
WAN	*Local Area Network*
WLAN	*Wireless Local Access Network*
WFQ	*Weighted Fair Queuing*
WF^{2}Q	*Worst-Case Fair Weighted Fair Queuing*
WRR	*Weighted Round Robin*
XML	*eXtensible Markup Language*

Technical jargon from the aeronautic industry

ACD	*Aircraft Control Domain*
AFDX	*Avionics Full-Duplex Switched Ethernet*
AISD	*Airline Information Services Domain*
ARINC	*Aeronautical Radio, Incorporated*
BAG	*Bandwidth Allocation Gap*
CCS	*Cabin Core Sub-domain*, part of the ACD domain
COTS	*Commercial Off-The-Shelves*
DAL	*Design Assurance Level*
ES	*End-System*
HMI	*Human-Machine Interface*
IFE	*In-Flight Entertainment*
PIESD	*Passenger Information and Entertainment Services Domain*
PODD	*Passenger Owned Devices Domain*
PSU	*Passenger Service Unit*
s_{min}	Minimum packet size of a Virtual Link
s_{max}	Maximum packet size of a Virtual Link
VL	*Virtual Link*

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS AND LISTINGS

BIBLIOGRAPHY

[1] IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pages 1–277, June 2004. doi:10.1109/IEEESTD.2004.94569.

[2] IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams. *802.1Qav-2009 (Amendment to 802.1Q-2005)*, December 2009. doi:10.1109/IEEESTD.2009.5375704.

[3] IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP). *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, pages 1–119, September 2010. doi:10.1109/IEEESTD.2010.5594972.

[4] IEEE Standard for Local and Metropolitan Area Networks: Port-Based Network Access Control. *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)*, pages 1–205, February 2010. doi:10.1109/IEEESTD.2010.5409813.

[5] IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks. *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)*, pages 1–1365, August 2011. doi:10.1109/IEEESTD.2011.6009146.

[6] IEEE Standard for Ethernet. *IEEE Std 802.3-2012*, December 2012. doi:10.1109/IEEESTD.2012.6419735.

[7] Road vehicles – FlexRay communications system – Part 1: General information and use case definition. *ISO 17458-1:2013*, pages 1–24, 2013.

[8] IEEE 802.1Qbu Task Group - Frame Preemption, Accessed 2015/07/30. URL `http://www.ieee802.org/1/pages/802.1bu.html`.

[9] IEEE 802.1Qbv Task Group - Enhancements for Scheduled Traffic, Accessed 2015/07/30. URL `http://www.ieee802.org/1/pages/802.1bv.html`.

[10] IEEE Study Group Distinguished Minimum Latency Traffic in a Converged Traffic Environment (DMLT), Accessed 2015/07/30. URL `http://www.ieee802.org/3/DMLT/`.

[11] IEEE Time-Sensitive Networking Task Group, Accessed 2015/07/30. URL `http://www.ieee802.org/1/pages/tsn.html`.

[12] IEEE 802.1 Audio/Video Bridging Task Group, Accessed 2015/07/30. URL http://www.ieee802.org/1/pages/avbridges.html.

[13] GENI – Global Environment for Network Innovations, Accessed 2015/07/30. URL https://www.geni.net.

[14] INET Framework for OMNeT++/OMNEST, Accessed 2015/07/30. URL http://inet.omnetpp.org/.

[15] ns-2, Network Simulator (ver. 2.35), Accessed 2015/07/30. URL http://nsnam.isi.edu/nsnam/index.php/Main_Page.

[16] ns-3 Discrete-Event Network Simulator, Accessed 2015/07/30. URL http://www.nsnam.org/.

[17] OMNeT++ Network Simulation Framework, Accessed 2015/07/30. URL http://www.omnetpp.org/.

[18] The R Project for Statistical Computing, Accessed 2015/07/30. URL http://www.r-project.org/.

[19] Patrice Abry, Patrick Flandrin, Murad S. Taqqu, and Darryl Veitch. Wavelets for the Analysis, Estimation and Synthesis of Scaling Data. In Kihong Park and Walter Willinger, editors, *Self-Similar Network Traffic and Performance Evaluation*, pages 39–88. Wiley, 2000. ISBN 978-0-471-31974-0. doi:10.1002/047120644X.ch2.

[20] Aeronautical Radio Inc. ARINC Specification 636: Onboard Local Area Network (OLAN), January 1993.

[21] Aeronautical Radio Inc. ARINC Specification 646: Ethernet Local Area Network (ELAN), December 1995.

[22] Aeronautical Radio Inc. ARINC Specification 664P5: Aircraft Data Network, Part 5 - Network Domain Characteristics and Interconnection, April 2005.

[23] Aeronautical Radio Inc. ARINC Specification 628P0-2: Cabin Equipment Interfaces - Part 0 - Cabin Management and Entertainment System - Overview, December 2008.

[24] Aeronautical Radio Inc. ARINC Specification 664P7-1: Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network, September 2009.

[25] Aeronautical Radio Inc. ARINC Specification 746-6: Cabin Communications Systems (CCS), November 2011.

[26] Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock. Host-to-Host Congestion Control for TCP. *IEEE Communications Surveys and Tutorials*, 12(3):304–342, May 2010. ISSN 1553-877X. doi:10.1109/SURV.2010.042710.00114.

[27] Rajeev Agrawal, Rene L. Cruz, Clayton Okino, and Rajendran Rajan. Performance Bounds for Flow Control Protocols. *IEEE/ACM Transactions on Networking*, 7(3):310–323, June 1999. ISSN 1063-6692. doi:10.1109/90.779197.

[28] Marco Ajmone Marsan, Michele Garetto, Paolo Giaccone, Emilio Leonardi, Enrico Schiattarella, and Alessandro Tarello. Using Partial Differential Equations to Model TCP Mice and Elephants in Large IP Networks. *IEEE/ACM Transactions on Networking*, 13(6):1289–1301, December 2005. ISSN 1063-6692. doi:10.1109/TNET.2005.860102.

[29] Giuliana Alderisi, Gaetano Patti, and Lucia Lo Bello. Introducing Support for Scheduled Traffic over IEEE Audio Video Bridging Networks. In *Proceedings of the 18th IEEE Conference on Emerging Technologies Factory Automation*, ETFA, pages 1–9, September 2013. doi:10.1109/ETFA.2013.6647943.

[30] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). *ACM SIGCOMM Computer Communication Review*, 41(4):63–74, October 2011. doi:10.1145/1851275.1851192.

[31] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009. URL `http://www.ietf.org/rfc/rfc5681.txt`.

[32] Mark Allman and Aaron Falk. On the Effective Evaluation of TCP. *ACM SIGCOMM Computer Communication Review*, 29(5):59–70, October 1999. ISSN 0146-4833. doi:10.1145/505696.505703.

[33] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. TCP Network Calculus: The case of large delay-bandwidth product. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1 of *INFOCOM 2002*, pages 417–426. IEEE, June 2002. doi:10.1109/INFCOM.2002.1019284.

[34] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, April 1994. doi:10.1016/0304-3975(94)90010-8.

[35] Allan T. Andersen and Bo Friis Nielsen. A Markovian Approach for Modeling Packet Traffic with Long-Range Dependence. *IEEE Journal on Selected Areas in Communications*, 16(5):719–732, June 1998. doi:10.1109/49.700908.

[36] Konstantin Avrachenkov, Urtzi Ayesta, Patrick Brown, and Eeva Nyberg. Differentiation Between Short and Long TCP Flows: Predictability of the Response Time. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM 2004, pages 762–773. IEEE, March 2004. doi:10.1109/INFCOM.2004.1356965.

[37] Urtzi Ayesta, Konstantin E. Avratchenkov, Eitan Altman, Chadi Barakat, and Parijat Dube. Multilevel Approach for Modeling TCP/IP. In *Proceedings of the 8th International Teletraffic Congress (ITC 18)*, September 2003.

[38] François Baccelli and Dohy Hong. TCP is Max-Plus Linear and what it tells us on its throughput. In *ACM SIGCOMM Computer Communication Review*, volume 30, pages 219–230. ACM, October 2000. doi:10.1145/347057.347548.

[39] François Baccelli and Dohy Hong. Flow Level Simulation of Large IP Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM 2003*, pages 1911–1921. IEEE, March 2003. doi:10.1109/INFCOM.2003.1209213.

[40] François Baccelli and Dohy Hong. Interaction of TCP Flows As Billiards. *IEEE/ACM Transactions on Networking*, 13(4):841–853, August 2005. ISSN 1063-6692. doi:10.1109/TNET.2005.852883.

[41] François Baccelli and David R. McDonald. A stochastic model for the throughput of non-persistent TCP flows. *Performance Evaluation*, 65(6–7):512–530, June 2008. ISSN 0166-5316. doi:10.1016/j.peva.2007.12.006.

[42] Andrea Bacioccola, Claudio Cicconetti, and Giovanni Stea. User-level Performance Evaluation of VoIP Using ns-2. In *Proceedings of the 1st International ICST Workshop on Network Simulation Tools*, pages 20:1–20:10, October 2007. doi:10.4108/nstools.2007.2014.

[43] Andrea Baiocchi, Angelo P. Castellani, and Francesco Vacirca. YeAH-TCP: Yet Another Highspeed TCP. In *Proceedings of the 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet)*, volume 7, pages 37–42, 2007.

[44] F. Baker, J. Polk, and M. Dolly. A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic. RFC 5865 (Proposed Standard), May 2010. URL `http://www.ietf.org/rfc/rfc5865.txt`.

[45] Anand Balachandran, Goeffrey M. Voelker, Paramvir Bahl, and P. Venkat Rangan. Characterizing User Behavior and Network Performance in a Public Wireless LAN. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 195–205. ACM, June 2002. doi:10.1145/511334.511359.

[46] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, and M. Sooriyabandara. TCP Performance Implications of Network Path Asymmetry. RFC 3449 (Best Current Practice), December 2002. URL `http://www.ietf.org/rfc/rfc3449.txt`.

[47] Wei Bao, Vincent W. S. Wong, and Victor C. M. Leung. A Model for Steady State Throughput of TCP CUBIC. In *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6. IEEE, December 2010. doi:10.1109/GLOCOM.2010.5684172.

[48] Mario Barbera, Alfio Lombardo, Giovanni Schembra, and Andrea Trecarichi. Fluid flow analysis of TCP flows in a DiffServ environment. *European Transactions on Telecommunications*, 17(5):505–524, September 2006. doi:10.1002/ett.1093.

[49] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 26(1):151–160, June 1998. ISSN 0163-5999. doi:10.1145/277858.277897.

[50] Forest Baskett, K. Mani Chandy, Richard R. Muntz, and Fernando G. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, April 1975. doi:10.1145/321879.321887.

[51] Henri Bauer. *Analyse pire cas de flux hétérogènes dans un réseau embarqué avion*. PhD thesis, Université de Toulouse, October 2011.

[52] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. Impact of QoS mechanisms on end-to-end delays for an avionics switched Ethernet. In *Proceedings of the 2nd Junior Researcher Workshop on Real-Time Computing*, 2008.

[53] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach. *IEEE Transactions on Industrial Informatics*, 6(4):521–533, November 2010. ISSN 1551-3203. doi:10.1109/TII.2010.2055877.

[54] Michael A. Beck and Jens B. Schmitt. The DISCO Stochastic Network Calculator Version 1.0 – When Waiting Comes to an End. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, pages 282–285. ICST, December 2013. doi:10.4108/icst.valuetools.2013.254378.

[55] 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 *Proceedings of the 2nd IEEE International Conference on Cloud Networking (CloudNet)*, pages 190–194, November 2013. doi:10.1109/CloudNet.2013.6710576.

[56] William H. Bell, David G. Cameron, A. Paul Millar, Luigi Capozza, Kurt Stockinger, and Floriano Zini. OptorSim – A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17 (4):403–416, November 2003. doi:10.1177/10943420030174005.

[57] Slim Ben Fredj, Thomas Bonald, Alexandre Proutiere, Gwénaël Régnié, and James W. Roberts. Statistical Bandwidth Sharing: A Study of Congestion at Flow Level. *ACM SIGCOMM Computer Communication Review*, 31(4):111–122, August 2001. doi:10.1145/964723.383068.

[58] Jon C. R. Bennett and Hui Zhang. WF^{2}Q: Worst-case Fair Weighted Fair Queueing. In *Proceedings of the 15th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1 of *INFOCOM 1996*, pages 120–128, March 1996. doi:10.1109/INFCOM.1996.497885.

[59] Jon C. R. Bennett and Hui Zhang. Hierarchical Packet Fair Queueing Algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675–689, October 1997. ISSN 1063-6692. doi:10.1109/90.649568.

[60] Arthur W. Berger and Yaakov Kogan. Dimensioning Bandwidth for Elastic Traffic in High-Speed Data Networks. *IEEE/ACM Transactions on Networking*, 8(5):643–654, October 2000. doi:10.1109/90.879350.

[61] Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks*. Prentice-Hall International, 2nd edition, 1992.

[62] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475 (Informational), December 1998. URL `http://www.ietf.org/rfc/rfc2475.txt`. Updated by RFC 3260.

[63] Mathias Bohge and Martin Renwanz. A realistic VoIP traffic generation and evaluation tool for OMNeT++. In *Proceedings of the 1st International Workshop on OMNeT++*, March 2008. doi:10.4108/ICST.SIMUTOOLS2008.3003.

[64] Thomas Bonald, Philippe Olivier, and James Roberts. Dimensioning high speed IP access networks. In *Proceedings of the 8th International Teletraffic Congress (ITC 18)*, volume 5, pages 241–250, August 2003. doi:10.1016/S1388-3437(03)80169-5.

[65] Anne Bouillard, Bertrand Cottenceau, Bruno Gaujal, Laurent Hardouin, Sébastien Lagrange, and Mehdi Lhommeau. COINC Library: a toolbox for the Network Calculus. In *Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS*, number 35. ICST, October 2009. doi:10.4108/ICST.VALUETOOLS2009.8045.

[66] Onno Boxma and Bert Zwart. Tails in scheduling. *ACM SIGMETRICS Performance Evaluation Review*, 34(4):13–20, March 2007. doi:10.1145/1243401.1243406.

[67] Onno J. Boxma, Nidhi Hegde, and Rudesindo Núñez-Queija. Exact and approximate analysis of sojourn times in finite discriminatory processor sharing queues. *International Journal of Electronics and Communications (AEU)*, 60(2):109–115, February 2006. ISSN 1434-8411. doi:10.1016/j.aeue.2005.11.007.

[68] Marc Boyer. NC-maude: a rewriting tool to play with network calculus. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6415 of *Lecture Notes in Computer Science*, pages 137–151. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16557-3.

[69] Marc Boyer, Jörn Migge, and Marc Fumey. PEGASE - A Robust and Efficient Tool for Worst-Case Network Traversal Time Evaluation on AFDX. In *Proceedings of SAE Aerotech 2011*, January 2011. doi:10.4271/2011-01-2711.

[70] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. URL `http://www.ietf.org/rfc/rfc2205.txt`. Updated by RFCs 2750, 3936, 4495, 5946, 6437, 6780.

[71] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13 (8):1465–1480, October 1995. doi:10.1109/49.464716.

[72] Eli Brosh, Salman Abdul Baset, Vishal Misra, Dan Rubenstein, and Henning Schulzrinne. The Delay-Friendliness of TCP for Real-Time Traffic. *IEEE/ACM Transactions on Networking*, 18(5):1478–1491, October 2010. ISSN 1063-6692. doi:10.1109/TNET.2010.2050780.

[73] Tian Bu and Don Towsley. Fixed Point Approximations for TCP behavior in an AQM Network. In *ACM SIGMETRICS Performance Evaluation Review*, volume 29, pages 216–225. ACM, June 2001. doi:10.1145/384268.378786.

[74] Łukasz Budzisz, Rade Stanojević, Robert Shorten, and Fred Baker. A Strategy for Fair Coexistence of Loss and Delay-Based Congestion Control Algorithms. *IEEE Communications Letters*, 13(7):555–557, July 2009. doi:10.1109/LCOMM.2009.090305.

[75] Carlo Caini and Rosario Firrincieli. TCP Hybla: a TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22 (5):547–566, August 2004. doi:10.1002/sat.799.

[76] Jin Cao, William S. Cleveland, Yuan Gao, Kevin Jeffay, F. Donelson Smith, and Michele Weigle. Stochastic Models for Generating Synthetic HTTP Source Traffic. In *Proceedings of the twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM 2004*, pages 1546–1557. IEEE, 2004. doi:10.1109/INFCOM.2004.1354568.

[77] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling TCP Latency. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM 2000*, pages 1742–1751. IEEE, March 2000. doi:10.1109/INFCOM.2000.832574.

[78] Gonzalo Carvajal, Chun Wah Wu, and Sebastian Fischmeister. Evaluation of Communication Architectures for Switched Real-time Ethernet. *IEEE Transactions on Computers*, 63(1):218–229, January 2014. doi:10.1109/TC.2012.177.

[79] Giuliano Casale, E. Zhang Zhang, and Evgenia Smirni. KPC-Toolbox: Simple Yet Effective Trace Fitting Using Markovian Arrival Processes. In *Proceedings of the 5th International Conference on Quantitative Evaluation of Systems, 2008 (QEST)*, pages 83–92. IEEE, September 2008. doi:10.1109/QEST.2008.33.

[80] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *Proceedings of the 10th International Conference on Computer Modeling and Simulation*, UKSIM '08, pages 126–131, April 2008. ISBN 978-0-7695-3114-4. doi:10.1109/UKSIM.2008.28.

[81] Claudio Casetti and Michela Meo. A New Approach to Model the Stationary Behavior of TCP Connections. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1 of *INFOCOM 2000*, pages 367–375. IEEE, March 2000. doi:10.1109/INFCOM.2000.832207.

[82] Cheng-Shang Chang. On Deterministic Traffic Regulation and Service Guarantees: A Systematic Approach by Filtering. *IEEE Transactions on Information Theory*, 44(3): 1097–1110, May 1998. doi:10.1109/18.669173.

[83] Cheng-Shang Chang. *Performance Guarantees in Communication Networks*. Springer, 2000. ISBN 978-1-4471-0459-9. doi:10.1007/978-1-4471-0459-9.

[84] Cheng-Shang Chang and Zhen Liu. A Bandwidth Sharing Theory for a Large Number of HTTP-Like Connections. *IEEE/ACM Transactions on Networking*, 12(5):952–962, October 2004. doi:10.1109/TNET.2004.836118.

[85] Hussein Charara. *Évaluation des performances temp réel de réseaux embarqués avioniques*. PhD thesis, Institut National Polytechnique de Toulouse, November 2007.

[86] Hussein Charara, Jean-Luc Scharbarg, Jérôme Ermont, and Christian Fraboul. Methods for bounding end-to-end delays on an AFDX network. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, July 2006. doi:10.1109/ECRTS.2006.15.

[87] Bin Bin Chen and P. Vicat-Blanc Primet. Supporting bulk data transfers of high-end applications with guaranteed completion time. In *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, pages 575–580. IEEE, June 2007. doi:10.1109/ICC.2007.100.

[88] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, pages 73–82. ACM, August 2009. doi:10.1145/1592681.1592693.

[89] Mung Chiang, Steven H. Low, A. Robert Calderbank, and John C. Doyle. Layering as Optimization Decomposition: A Mathematical Theory of Network Architectures. *Proceedings of the IEEE*, 95(1):255–312, January 2007. doi:10.1109/JPROC.2006.887322.

[90] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003. ISSN 0146-4833. doi:10.1145/956993.956995.

[91] Florin Ciucu, Almut Burchard, and Jörg Liebeherr. A Network Service Curve Approach for the Stochastic Analysis of Networks. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):279–290, June 2005. ISSN 0163-5999. doi:10.1145/1071690.1064251.

[92] John W. Cohen. The Multiple Phase Service Network with Generalized Processor Sharing. *Acta Informatica*, 12(3):245–284, October 1979. ISSN 0001-5903. doi:10.1007/BF00264581.

[93] William Jay Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, 1971.

[94] Mark E. Crovella and Azer Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997. doi:10.1109/90.650143.

[95] Rene L. Cruz. A Calculus for Network Delay, Part I. Network Elements in Isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991. doi:10.1109/18.61109.

[96] Rene L. Cruz. A Calculus for Network Delay, Part II. Network Analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991. doi:10.1109/18.61110.

[97] Joan Adrià Ruiz De Azua and Marc Boyer. Complete Modelling of AVB in Network Calculus Framework. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 55:55–55:64. ACM, October 2014. ISBN 978-1-4503-2727-5. doi:10.1145/2659787.2659810.

[98] Brunonas Dekeris, Tomas Adomkus, and Aurelijus Budnikas. Analysis of QoS Assurance Using Weighted Fair Queueing (WQF) Scheduling Discipline with Low Latency Queue (LLQ). In *Proceedings of the 28th International Conference on Information Technology Interfaces*, ITI, pages 507–512, June 2006. doi:10.1109/ITI.2006.1708533.

[99] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and Simulation of a Fair Queueing Algorithm. *ACM SIGCOMM Computer Communication Review*, 19(4):1–12, August 1989. ISSN 0146-4833. doi:10.1145/75247.75248.

[100] Luc Devroye. *Non-Uniform Random Variate Generation*, volume 4. Springer-Verlag New York, 1986. ISBN 978-0387963051.

[101] Amogh Dhamdhere and Constantine Dovrolis. Open Issues in Router Buffer Sizing. *ACM SIGCOMM Computer Communication Review*, 36(1):87–92, January 2006. ISSN 0146-4833. doi:10.1145/1111322.1111342.

[102] M. Duke, R. Braden, W. Eddy, E. Blanton, and A. Zimmermann. A Roadmap for Transmission Control Protocol (TCP) Specification Documents. RFC 7414 (Informational), February 2015. URL `http://www.ietf.org/rfc/rfc7414.txt`.

[103] Mohamed A. El-Gendy, Abhijit Bose, and Kang G. Shin. Evolution of the Internet QoS and Support for Soft Real-Time Applications. *Proceedings of the IEEE*, 91(7):1086–1104, July 2003. ISSN 0018-9219. doi:10.1109/JPROC.2003.814615.

[104] Tore Olaus Engset. Die Wahrscheinlichkeitsrechnung zur Bestimmung der Wählerzahl in automatischen Fernsprechämtern. *Elektrotechnische Zeitschrift*, 39:304–306, August 1918.

[105] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A First Look at Traffic on Smartphones. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 281–287. ACM, November 2010. ISBN 978-1-4503-0483-2. doi:10.1145/1879141.1879176.

[106] Kevin R. Fall and W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 2nd edition, 2011. ISBN 978-0321336316.

[107] Max Felser. Real-Time Ethernet - Industry Prospective. *Proceedings of the IEEE*, 93(6): 1118–1129, June 2005. ISSN 0018-9219. doi:10.1109/JPROC.2005.849720.

[108] Markus Fidler. An End-to-End Probabilistic Network Calculus with Moment Generating Functions. In *Proceedings of the 14th IEEE International Workshop on Quality of Service (IWQoS)*, pages 261–270, June 2006. doi:10.1109/IWQOS.2006.250477.

[109] Markus Fidler. A Survey of Deterministic and Stochastic Service Curve Models in the Network Calculus. *IEEE Communications Surveys and Tutorials*, 12(1):59–86, 2010. ISSN 1553-877X. doi:10.1109/SURV.2010.020110.00019.

[110] Victor Firoiu and Marty Borden. A Study of Active Queue Management for Congestion Control. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM 2000*, pages 1435–1444. IEEE, March 2000. doi:10.1109/INFCOM.2000.832541.

[111] Victor Firoiu, Ikjun Yeom, and Xiaohui Zhang. A Framework for Practical Performance Evaluation and Traffic Engineering in IP Networks. In *Proceedings of the IEEE International Conference on Telecommunications*, 2001.

[112] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003. URL http://www.ietf.org/rfc/rfc3649.txt.

[113] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348 (Proposed Standard), September 2008. URL http://www.ietf.org/rfc/rfc5348.txt.

[114] Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, August 1999. doi:10.1109/90.793002.

[115] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993. doi:10.1109/90.251892.

[116] Sally Floyd and Van Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995. doi:10.1109/90.413212.

[117] Cheng Peng Fu and Soung C. Liew. TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks. *IEEE Journal on Selected Areas in Communications*, 21 (2):216–228, February 2003. doi:10.1109/JSAC.2002.807336.

[118] Fei Ge and Liansheng Tan. Network Utility Maximization in Two-way Flow Scenario. *ACM SIGCOMM Computer Communication Review*, 44(2):13–19, April 2014. doi:10.1145/2602204.2602207.

[119] Fabien Geyer, Emanuel Heidinger, Stefan Schneele, and Alexander von Bodisco. Evaluation of Audio/Video Bridging Forwarding Method in an Avionics Switched Ethernet Context. In *Proceedings of the 18th IEEE Symposium on Computers and Communications*, ISCC 2013, pages 711–716. IEEE, July 2013. doi:10.1109/ISCC.2013.6755032.

[120] Fabien Geyer, Stefan Schneele, and Georg Carle. RENETO, a Realistic Network Traffic Generator for OMNeT++/INET. In *Proceedings of the 6th International Conference on Simulation Tools and Techniques*, SIMUTOOLS 2013, pages 73–81, March 2013. ISBN 978-1-4503-2464-9. doi:10.4108/icst.simutools.2013.251697.

[121] Fabien Geyer, Stefan Schneele, and Georg Carle. Practical Performance Evaluation of Ethernet Networks with Flow-Level Network Modeling. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS 2013, pages 253–262, December 2013. doi:10.4108/icst.valuetools.2013.254367.

[122] Fabien Geyer, Stefan Schneele, Matthias Heinisch, and Peter Klose. Simulation and Performance Evaluation of an Aircraft Cabin Network Node. In Frank Thielecke Otto von Estorff, editor, *Proceedings of the 4th International Workshop on Aircraft System Technologies*, AST 2013, pages 323–332. Shaker-Verlag, April 2013. ISBN 978-3-8440-1850-9.

[123] Fabien Geyer, Stefan Schneele, and Georg Carle. Towards Stochastic Flow-Level Network Modeling: Performance Evaluation of Short TCP Flows. In *Proceedings of the*

39th IEEE Conference on Local Computer Networks, LCN 2014, pages 462–465. IEEE, September 2014. doi:10.1109/LCN.2014.6925817.

[124] Fabien Geyer, Stefan Schneele, and Georg Carle. PETFEN: A Performance Evaluation Tool for Flow-Level Network Modeling of Ethernet Networks. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS 2014, December 2014. doi:10.4108/icst.valuetools.2014.258166.

[125] Fabien Geyer, Stefan Schneele, and Wolfgang Fischer. Performance Evaluation of an Ethernet-Based Cabin Network Architecture Supporting a Low-Latency Service. In *Proceedings of the 6th International Workshop Nets4Cars/Nets4Trains/Nets4Aircraft*, LNCS 8435, pages 69–80. Springer, May 2014. doi:10.1007/978-3-319-06644-8_7.

[126] R. J. Gibbens, S. K. Sargood, C. Van Eijl, F. P. Kelly, H. Azmoodeh, R. N. Macfadyen, and N. W. Macfadyen. Fixed-Point Models for the End-to-End Performance Analysis of IP Networks. In *Proceedings of the 13th ITC Specialist Seminar: IP Traffic Measurement, Modeling and Management*, pages 10/1–10/8, September 2000.

[127] S. Jamaloddin Golestani. A Self-Clocked Fair Queueing Scheme for Broadband Applications. In *Proceedings of the 13th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM 1994, pages 636–646, June 1994. doi:10.1109/INFCOM.1994.337677.

[128] Jérôme Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques*. PhD thesis, Institut National Polytechnique de Toulouse, September 2004.

[129] Fabrice Guillemin, Philippe Robert, and Bert Zwart. Tail asymptotics for processor-sharing queues. *Advances in Applied Probability*, 36(2):525–543, June 2004. doi:10.1239/aap/1086957584.

[130] Roch A. Guérin and Vicent Pla. Aggregation and Conformance in Differentiated Service Networks: A Case Study. *ACM SIGCOMM Computer Communication Review*, 31 (1):21–32, January 2001. ISSN 0146-4833. doi:10.1145/382176.383018.

[131] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, July 2008. doi:10.1145/1400097.1400105.

[132] Hassan Hassan, Olivier Brun, Jean-Marie Garcia, and David Gauchard. Integration of Streaming and Elastic Traffic: A Fixed Point Approach. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques*, SIMUTOOLS 2008, pages 25:1–25:10. ICST, March 2008. ISBN 978-963-9799-20-2. doi:10.4108/ICST.SIMUTOOLS2008.2931.

[133] David A. Hayes and Grenville Armitage. Improved Coexistence and Loss Tolerance for Delay Based TCP Congestion Control. In *Proceedings of the 35th IEEE Conference on Local Computer Networks*, LCN 2010, pages 24–31. IEEE, October 2010. doi:10.1109/LCN.2010.5735714.

[134] David A. Hayes and Grenville Armitage. Revisiting TCP Congestion Control Using Delay Gradients. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, volume 6641 of *Lecture Notes in Computer Science*, pages 328–341. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-20798-3_25.

[135] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582 (Proposed Standard), April 2012. URL http://www.ietf.org/rfc/rfc6582.txt.

[136] Felix Hernandez-Campos, Kevin Jeffay, and F. Donelson Smith. Modeling and generating TCP application workloads. In *Proceedings of the 4th International Conference on Broadband Communications, Networks and Systems (BROADNETS 2007)*, pages 280–289, September 2007. doi:10.1109/BROADNETS.2007.4550436.

[137] Martin Heusse, Sears A. Merritt, Timothy X. Brown, and Andrzej Duda. Two-way TCP Connections: Old Problem, New Insight. *ACM SIGCOMM Computer Communication Review*, 41(2):5–15, April 2011. ISSN 0146-4833. doi:10.1145/1971162.1971164.

[138] Daniel P. Heyman, T. V. Lakshman, and Arnold L. Neidhardt. A New Method for Analysing Feedback-Based Protocols with Applications to Engineering Web Traffic over the Internet. In *ACM SIGMETRICS Performance Evaluation Review*, volume 25, pages 24–38. ACM, June 1997. doi:10.1145/258623.258670.

[139] Rich Hickey. The Clojure Programming Language. In *Proceedings of the 2008 Symposium on Dynamic Languages*. ACM, July 2008. doi:10.1145/1408681.1408682.

[140] Geert Jan Hoekstra, Robert D. van der Mei, and Natalia Diaz-Feraren. Engineering Elastic Traffic in TCP-Based Networks: Processor Sharing and Effective Service Time. In *Proceedings of the 25th International Teletraffic Congress (ITC 25)*, pages 1–8. IEEE, September 2013. doi:10.1109/ITC.2013.6662958.

[141] Chao-Ju Hou and Khan G. Shin. Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems. *IEEE Transactions on Computers*, 46(12):1338–1356, December 1997. ISSN 0018-9340. doi:10.1109/12.641934.

[142] Wolfgang Hörmann and Josef Leydold. Continuous Random Variate Generation by Fast Numerical Inversion. *ACM Transactions on Modeling and Computer Simulation*, 13 (4):347–362, October 2003. doi:10.1145/945511.945517.

[143] Yu Hua and Xue Liu. Scheduling design and analysis for end-to-end heterogeneous flows in an avionics network. In *Proceedings of the 30th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM 2011, pages 2417–2425, April 2011. doi:10.1109/INFCOM.2011.5935062.

[144] Jahanzaib Imtiaz, Jürgen Jasperneite, and Lixue Han. A Performance Study of Ethernet Audio Video Bridging (AVB) for Industrial Real-time Communication. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation*, ETFA, pages 1–8, September 2009. doi:10.1109/ETFA.2009.5347126.

[145] Jahanzaib Imtiaz, Jürgen Jasperneite, and Sebastian Schriegel. A Proposal to Integrate Process Data Communication to IEEE 802.1 Audio Video Bridging (AVB). In *Proceedings of IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, September 2011. doi:10.1109/ETFA.2011.6059004.

[146] Adam Jacobs, John Wernicke, Sarl Oral, Burt Gordon, and Alan George. Experimental Characterization of QoS in Commercial Ethernet Switches for Statistically Bounded Latency in Aircraft Networks. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN 2004, pages 190–197, November 2004. doi:10.1109/LCN.2004.56.

[147] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992. URL `http://www.ietf.org/rfc/rfc1323.txt`. Obsoleted by RFC 7323.

[148] Van Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM Computer Communication Review*, volume 18, pages 314–329. ACM, August 1988. doi:10.1145/52325.52356.

[149] Van Jacobson. Modified TCP congestion avoidance algorithm. end2end-interest mailing list, April 1990.

[150] Sam Jansen and Anthony McGregor. Simulation with Real World Network Stacks. In *Proceedings of the 2005 Winter Simulation Conference*. IEEE, December 2005. doi:10.1109/WSC.2005.1574539.

[151] Yuming Jiang. A Note on Applying Stochastic Network Calculus, May 2010.

[152] Yuming Jiang and Yong Liu. *Stochastic Network Calculus*. Springer, September 2008. ISBN 978-1-84800-127-5. doi:10.1007/978-1-84800-127-5.

[153] Yuming Jiang and Qi Yao. Impact of FIFO Aggregation on Delay Performance of a Differentiated Services Network. In Hyun-Kook Kahng, editor, *Information Networking*, volume 2662 of *Lecture Notes in Computer Science*, pages 948–957. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40827-7. doi:10.1007/978-3-540-45235-5_93.

[154] Yuming Jiang, Qinghe Yin, Yong Liu, and Shengming Jiang. Fundamental Calculus on Generalized Stochastically Bounded Bursty Traffic for Communication Networks. *Computer Networks*, 53(12):2011–2021, August 2009. doi:10.1016/j.comnet.2009.03.004.

[155] Kristján Valur Jónsson. HttpTools: a toolkit for simulation of web hosts in OMNeT++. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, SIMUTools '09, pages 70:1–70:8, March 2009. ISBN 978-963-9799-45-5. doi:10.4108/ICST.SIMUTOOLS2009.5589.

[156] Jan Kamieth, Till Steinbach, Franz Korf, and Thomas C. Schmidt. Design of TDMA-based In-Car Networks: Applying Multiprocessor Scheduling Strategies on Time-Triggered Switched Ethernet Communication. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–9, September 2014. doi:10.1109/ETFA.2014.7005119.

[157] Frank P. Kelly, Aman K. Maulloo, and David K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, March 1998. doi:10.1057/palgrave.jors.2600523.

[158] Tom Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, April 2003. doi:10.1145/956981.956989.

[159] Arzad A. Kherani and Anurag Kumar. Stochastic Models for Throughput Analysis of Randomly Arriving Elastic Flows in the Internet. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2 of *INFOCOM 2002*, pages 1014–1023, June 2002. doi:10.1109/INFCOM.2002.1019349.

[160] Geunhyung Kim and Cheeha Kim. Deterministic Edge-to-Edge Delay Bounds for a Flow in a DiffServ Network Domain. In Hyun-Kook Kahng and Shigeki Goto, editors, *Information Networking. Networking Technologies for Broadband and Mobile Networks*, volume 3090 of *Lecture Notes in Computer Science*, pages 370–379. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23034-2. doi:10.1007/978-3-540-25978-7_38.

[161] Leonard Kleinrock. *Queueing Systems, Volume 2: Computer Applications*. Wiley-Interscience, January 1976. ISBN 978-0-471-49111-8.

[162] Hermann Kopetz and Günter Grünsteidl. TTP – A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, pages 524–533, June 1993. doi:10.1109/FTCS.1993.627355.

[163] Aleksandar Kuzmanovic and Edward W. Knightly. TCP-LP: A Distributed Algorithm for Low Priority Data Transfer. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM 2003*, pages 1691–1701. IEEE, March 2003. doi:10.1109/INFCOM.2003.1209192.

[164] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet Inter-Domain Traffic. *ACM SIGCOMM Computer Communication Review*, 40(4):75–86, August 2010. ISSN 0146-4833. doi:10.1145/1851275.1851194.

[165] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6. ACM, October 2010. doi:10.1145/1868447.1868466.

[166] Pasi Lassila, Hans van den Berg, Michel Mandjes, and Rob Kooij. An integrated packet/flow model for TCP performance analysis. In *Proceedings of the 18th International Teletraffic Congress - ITC-18*, volume 5, pages 651–660. Elsevier, 2003. doi:10.1016/S1388-3437(03)80214-7.

[167] Averill M. Law and David W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3rd edition, 2000.

[168] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001. ISBN 978-3-540-42184-9. doi:10.1007/3-540-45318-0.

[169] Rodolfo I. Ledesma Goyzueta and Yu Chen. A Deterministic Loss Model Based Analysis of CUBIC. In *Proceedings of the IEEE International Conference on Computing, Networking and Communications (ICNC)*, pages 944–949. IEEE, January 2013. doi:10.1109/ICCNC.2013.6504217.

[170] Douglas Leith and Robert Shorten. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of the 2nd International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet)*, volume 2004, February 2004.

[171] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994. ISSN 1063-6692. doi:10.1109/90.282603.

[172] Jörg Liebeherr, Yashar Ghiassi-Farrokhfal, and Almut Burchard. On the Impact of Link Scheduling on End-to-End Delays in Large Networks. *IEEE Journal on Selected Areas in Communications*, 29(5):1009–1020, May 2011. ISSN 0733-8716. doi:10.1109/JSAC.2011.110511.

[173] Hyung-Taek Lim, Daniel Herrscher, and Firas Chaari. Performance comparison of IEEE 802.1Q and IEEE 802.1 AVB in an Ethernet-based in-vehicle network. In *Proceedings of the 8th International Conference on Computing Technology and Information Management*, volume 1 of *ICCM*, pages 1–6, April 2012.

[174] Shao Liu, Tamer Başar, and Ravi Srikant. TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks. *Performance Evaluation*, 65(6): 417–440, June 2008. doi:10.1016/j.peva.2007.12.007.

[175] Yong Liu, Francesco Lo Presti, Vishal Misra, Donald F. Towsley, and Yu Gu. Fluid Models and Solutions for Large-Scale IP Networks. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):91–101, June 2003. doi:10.1145/781027.781039.

[176] Zhen Liu, Philippe Nain, and Don Towsley. Exponential Bounds for a Class of Stochastic Processes with Application to Call Admission Control in Networks. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, volume 1, pages 156–161. IEEE, December 1994. doi:10.1109/CDC.1994.411029.

[177] Steven H. Low. A Duality Model of TCP and Queue Management Algorithms. *IEEE/ACM Transactions on Networking*, 11(4):525–536, August 2003. doi:10.1109/TNET.2003.815297.

[178] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, pages 90–102. ACM, November 2009. doi:10.1145/1644893.1644904.

[179] Michel Mandjes and Bert Zwart. Large deviations of sojourn times in processor sharing queues. *Queueing Systems*, 52(4):237–250, April 2006. doi:10.1007/s11134-006-5567-6.

[180] Shiwen Mao and Shivendra S. Panwar. A Survey of Envelope Processes and Their Applications in Quality of Service Provisioning. *IEEE Communications Surveys and Tutorials*, 8(3):2–20, July 2006. doi:10.1109/COMST.2006.253272.

[181] Steven Martin and Pascale Minet. Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, April 2006. doi:10.1109/IPDPS.2006.1639424.

[182] Steven Martin and Pascale Minet. Worst case end-to-end response times of flows scheduled with FP/FIFO. In *Proceedings of the 2006 International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, ICN/ICONS/MCL, April 2006. doi:10.1109/ICNICONSMCL.2006.231.

[183] Saverio Mascolo, Claudio Casetti, Mario Gerla, Medy Y. Sanadidi, and Ren Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 287–297. ACM, July 2001. doi:10.1145/381677.381704.

[184] Laurent Massoulié and James Roberts. Bandwidth Sharing: Objectives and Algorithms. *IEEE/ACM Transactions on Networking*, 10(3):320–328, June 2002. ISSN 1063-6692. doi:10.1109/TNET.2002.1012364.

[185] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, June 1997. doi:10.1145/263932.264023.

[186] Makoto Matsumoto and Takuji Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998. doi:10.1145/272991.272995.

[187] John Meier, Sung Kim, Alan George, and Sarp Oral. Gigabit COTS Ethernet Switch Evaluation for Avionics. In *Proceedings of the 1st Workshop on High-Speed Local Networks*, HSLN, pages 739–740, November 2002. doi:10.1109/LCN.2002.1181856.

[188] Marco Mellia and Hui Zhang. TCP Model for Short Lived Flows. *IEEE Communications Letters*, 6(2):85–87, February 2002. doi:10.1109/4234.984705.

[189] Jörn Migge. *L'ordonnancement sous contraintes temps-réel un modèle à base de trajectoires*. PhD thesis, INRIA Sophia Antipolis, November 1999.

[190] Anne Millet and Zoubir Mammeri. Delay bound Guarantees with WFQ-based CBQ discipline. In *Proceedings of the 12th IEEE International Workshop on Quality of Service*, IWQoS, pages 106–113, June 2004. doi:10.1109/IWQOS.2004.1309364.

[191] Vishal Misra, Wei-Bo Gong, and Don Towsley. Stochastic Differential Equation Modeling and Analysis of TCP Windowsize Behavior. In *Proceedings of IFIP WG 7.3 Performance*, November 1999.

[192] Vishal Misra, Wei-Bo Gong, and Don Towsley. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. In *ACM SIGCOMM Computer Communication Review*, volume 30, pages 151–160. ACM, October 2000. doi:10.1145/347057.347421.

[193] Jorge Muñoz-Castañer, Rafael Asorey-Cacheda, Felipe J. Gil-Castiñeira, Francisco J. González-Castaño, and Pedro S. Rodríguez-Hernández. A Review of Aeronautical Electronics and its Parallelism with Automotive Electronics. *IEEE Transactions on Industrial Electronics*, 58(7):3090–3100, July 2011. doi:10.1109/TIE.2010.2077614.

[194] Dritan Nace and Michał Pióro. Max-Min Fairness and Its Applications to Routing and Load-Balancing in Communication Networks: A Tutorial. *IEEE Communications Surveys and Tutorials*, 10(4):5–17, 2008. ISSN 1553-877X. doi:10.1109/SURV.2008.080403.

[195] Jitendra Padhye, Victor Firoiu, Don F. Towsley, and James F. Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, April 2000. ISSN 1063-6692. doi:10.1109/90.842137.

[196] Ruoming Pang and Vern Paxson. A High-level Programming Environment for Packet Trace Anonymization and Transformation. In *ACM SIGCOMM Computer Communication Review*, pages 339–351. ACM, August 2003. doi:10.1145/863955.863994.

[197] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993. ISSN 1063-6692. doi:10.1109/90.234856.

[198] Nadim Parvez, Anirban Mahanti, and Carey Williamson. An Analytic Throughput Model for TCP NewReno. *IEEE/ACM Transactions on Networking*, 18(2):448–461, April 2010. doi:10.1109/TNET.2009.2030889.

[199] Vern Paxson and Sally Floyd. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995. ISSN 1063-6692. doi:10.1109/90.392383.

[200] Shudeer Poojary and Vinod Sharma. Analytical Model for Congestion Control and Throughput with TCP CUBIC Connections. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6, December 2011. doi:10.1109/GLOCOM.2011.6134001.

[201] Ravi S. Prasad and Constantine Dovrolis. Measuring the Congestion Responsiveness of Internet Traffic. In *Proceedings of the 8th international conference on Passive and Active Network Measurement, PAM*, pages 176–185. Springer, 2007. doi:10.1007/978-3-540-71617-4_18.

[202] Ravi S. Prasad and Constantine Dovrolis. Beyond the Model of Persistent TCP Flows: Open-Loop vs Closed-Loop Arrivals of Non-Persistent Flows. In *Proceedings of the 41st Annual Simulation Symposium, ANSS*, pages 121–130. IEEE, April 2008. doi:10.1109/ANSS-41.2008.16.

[203] Rene Queck. Analysis of Ethernet AVB for Automotive Networks using Network Calculus. In *Proceedings of the 2012 IEEE International Conference on Vehicular Electronics and Safety*, ICVES, pages 61–67. IEEE, July 2012. doi:10.1109/ICVES.2012.6294261.

[204] Krithi Ramamritham. Allocation and Scheduling of Precedence-Related Periodic Tasks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):412–420, April 1995. ISSN 1045-9219. doi:10.1109/71.372795.

[205] Fabio Ricciato, Angelo Coluccia, Alessandro D'Alconzo, Darryl Veitch, Pierre Borgnat, and Patrice Abry. On the role of flows and sessions in Internet traffic modeling: an explorative toy-model. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–8, December 2009. doi:10.1109/GLOCOM.2009.5425847.

[206] George F. Riley. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, MoMeTools '03, pages 5–12. ACM, August 2003. ISBN 1-58113-748-6. doi:10.1145/944773.944775.

[207] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27(1):31–41, January 1997. doi:10.1145/251007.251012.

[208] Jim Roberts. A survey on statistical bandwidth sharing. *Computer Networks*, 45(3): 319–332, June 2004. doi:10.1016/j.comnet.2004.03.010.

[209] Chloé Rolland, Julien Ridoux, and Bruno Baynat. LiTGen, a Lightweight Traffic Generator: Application to P2P and Mail Wireless Traffic. In Steve Uhlig, Konstantina Papagiannaki, and Olivier Bonaventure, editors, *Passive and Active Network Measurement*, volume 4427 of *Lecture Notes in Computer Science*, pages 52–62. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-71616-7. doi:10.1007/978-3-540-71617-4_6.

[210] SAE International. ARP 4754A: Guidelines For Development Of Civil Aircraft and Systems, December 2010.

[211] Charalampos Samios and Mary K. Vernon. Modeling the Throughput of TCP Vegas. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):71–81, June 2003. doi:10.1145/781027.781037.

[212] Franklin E. Satterthwaite. An Approximate Distribution of Estimates of Variance Components. *Biometrics Bulletin*, 2(6):110–114, December 1946. doi:10.2307/3002019.

[213] Alexander Sayenko, Timo Hämäläinen, Jyrki Joutsensalo, and Jarmo Siltanen. On providing bandwidth and delay guarantees using the revenue criterion based adaptive WFQ. In *Proceedings of the 9th Asia-Pacific Conference on Communications. APCC 2003.*, volume 2, pages 745–750, September 2003. doi:10.1109/APCC.2003.1274457.

[214] Jean-Luc Scharbarg and Christian Fraboul. Simulation for end-to-end delays distribution on a switched Ethernet. In *Proceedings of the 2007 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1092–1099, September 2007. doi:10.1109/EFTA.2007.4416904.

[215] Jean-Luc Scharbarg, Frédéric Ridouard, and Christian Fraboul. A Probabilistic Analysis of End-To-End Delays on an AFDX Avionic Network. *IEEE Transactions on Industrial Informatics*, 5(1):38–49, February 2009. doi:10.1109/TII.2009.2016085.

[216] Henrik Schioler, Hans P. Schwefel, and Martin B. Hansen. CyNC – A MATLAB/SimuLink Toolbox for Network Calculus. In *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '07, pages 60:1–60:10. ICST, October 2007. doi:10.4108/valuetools.2007.1804.

[217] Jens B. Schmitt and Frank A. Zdarsky. The DISCO Network Calculator - A Toolbox for Worst Case Analysis. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools (Valuetools'06)*. ICST, ACM, October 2006. doi:10.1145/1190095.1190105.

[218] Jens B. Schmitt, Frank A. Zdarsky, and Markus Fidler. Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch.... In *Proceedings of the 27th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM 2008, pages 1669–1677, April 2008. doi:10.1109/INFOCOM.2008.228.

[219] Bianca Schroeder, Adam Wierman, and Mor Harchol-balter. Closed versus open system models and their impact on performance and scheduling. In *Proceedings of the Symposium on Networked Systems Design and Implementation*, NSDI 2006, 2006.

[220] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (INTERNET STANDARD), July 2003. URL `http://www.ietf.org/rfc/rfc3550.txt`. Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164.

[221] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low Extra Delay Background Transport (LEDBAT). RFC 6817 (Experimental), December 2012. URL `http://www.ietf.org/rfc/rfc6817.txt`.

[222] M. Shreedhar and George Varghese. Efficient Fair Queuing Using Deficit Round-Robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, June 1996. doi:10.1109/90.502236.

[223] Biplab Sikdar, Shivkumar Kalyanaraman, and Kenneth S. Vastola. Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno, and SACK. *IEEE/ACM Transactions on Networking*, 11(6):959–971, December 2003. doi:10.1109/TNET.2003.820427.

[224] Jörg Sommer, Sebastian Gunreben, Frank Feller, Martin Köhn, Ahlem Mifdaoui, Detlef Saß, and Joachim Scharf. Ethernet – A Survey on its Fields of Application. *IEEE Communications Surveys and Tutorials*, 12(2):263–284, April 2010. ISSN 1553-877X. doi:10.1109/SURV.2010.021110.00086.

[225] Joel Sommers and Paul Barford. Self-Configuring Network Traffic Generation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 68–81. ACM, October 2004. ISBN 1-58113-821-0. doi:10.1145/1028788.1028798.

[226] Joel Sommers, Rhys Bowden, Brian Eriksson, Paul Barford, Matthew Roughan, and Nick Duffield. Efficient Network-wide Flow Record Generation. In *Proceedings of the 30th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM 2011, pages 2363–2371, April 2011. doi:10.1109/INFCOM.2011.5935055.

[227] David Starobinski and Moshe Sidi. Stochastically Bounded Burstiness for Communication Networks. *IEEE Transactions on Information Theory*, 46(1):206–212, January 2000. ISSN 0018-9448. doi:10.1109/18.817518.

[228] Dimitrios Stiliadis and Anujan Varma. Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms. *IEEE/ACM Transactions on Networking*, 6 (5):611–624, October 1998. doi:10.1109/90.731196.

[229] Ion Stoica, Hui Zhang, and T. S. Eugene Ng. A Hierarchical Fair Service Curve Algorithm for Link-sharing, Real-time, and Priority Services. *IEEE/ACM Transactions on Networking*, 8(2):185–199, April 2000. ISSN 1063-6692. doi:10.1109/90.842141.

[230] Jack F. Suen, Russel B. Kegley, and Jonathan D. Preston. Affordable Avionic Networks with Gigabit Ethernet: Assessing the Suitability of Commercial Components for Airborne Use. In *Proceedings of IEEE Southeastcon 2013*, pages 1–6, April 2013. doi:10.1109/SECON.2013.6567491.

[231] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM 2006, pages 1–12, April 2006. doi:10.1109/INFOCOM.2006.188.

[232] The International Electrotechnical Commission. IEC 61508: Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems, May 2010.

[233] Pedro Velho and Arnaud Legrand. Accuracy Study and Improvement of Network Simulation in the SimGrid Framework. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques, SIMUTOOLS*. ICST, March 2009. doi:10.4108/ICST.SIMUTOOLS2009.5592.

[234] Pedro Velho, Lucas M. Schnorr, Henri Casanova, and Arnaud Legrand. Flow-level network models: have we reached the limits? Technical Report 7821, INRIA, November 2011.

[235] Kashi Venkatesh Vishwanath and Amin Vahdat. Realistic and Responsive Network Traffic Generation. In *ACM SIGCOMM Computer Communication Review*, SIGCOMM '06, pages 111–122. ACM, September 2006. ISBN 1-59593-308-5. doi:10.1145/1159913.1159928.

[236] Milan Vojnović and Jean-Yves Le Boudec. Stochastic Analysis of Some Expedited Forwarding Networks. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2 of *INFOCOM 2002*, pages 1004–1013, June 2002. doi:10.1109/INFCOM.2002.1019348.

[237] Milan Vojnović and Jean-Yves Le Boudec. Bounds for Independent Regulated Inputs Multiplexed in a Service Curve Network Element. *IEEE Transactions on Communications*, 51(5):735–740, May 2003. doi:10.1109/TCOMM.2003.811383.

[238] WAND Network Research Group. libflowmanager, Accessed 2015/07/30. URL `http://research.wand.net.nz/software/libflowmanager.php`.

[239] WAND Network Research Group. libprotoident, Accessed 2015/07/30. URL `http://research.wand.net.nz/software/libprotoident.php`.

[240] David X. Wei and Pei Cao. *NS-2 TCP-Linux*: An NS-2 TCP Implementation with Congestion Control Algorithms from Linux. In *Proceeding of the 2006 workshop on ns-2: the IP network simulator*, number 9. ACM, 2006. doi:10.1145/1190455.1190463.

[241] Michele C. Weigle, Prashanth Adurthi, Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith. Tmix: A Tool for Generating Realistic TCP Application Workloads in ns-2. *ACM SIGCOMM Computer Communication Review*, 36(3):65–76, July 2006. ISSN 0146-4833. doi:10.1145/1140086.1140094.

[242] Bernard L. Welch. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, January 1947. doi:10.2307/2332510.

[243] Adam Wierman, Takayuki Osogami, and Jörgen Olsén. A Unified Framework for Modeling TCP-Vegas, TCP-SACK, and TCP-Reno. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS)*, pages 269–278. IEEE, October 2003. doi:10.1109/MASCOT.2003.1240671.

[244] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The Worst-Case Execution-Time Problem – Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, May 2008. ISSN 1539-9087. doi:10.1145/1347375.1347389.

[245] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4 of *INFOCOM 2004*, pages 2514–2524. IEEE, March 2004. doi:10.1109/INFCOM.2004.1354672.

[246] Peng Yang, Juan Shao, Wen Luo, Lisong Xu, Jitender Deogun, and Ying Lu. TCP Congestion Avoidance Algorithm Identification. *IEEE/ACM Transactions on Networking*, 22(4):1311–1324, August 2014. ISSN 1063-6692. doi:10.1109/TNET.2013.2278271.

[247] Opher Yaron and Moshe Sidi. Performance and Stability of Communication Networks via Robust Exponential Bounds. *IEEE/ACM Transactions on Networking*, 1(3):372–385, June 1993. doi:10.1109/90.234858.

[248] Qinghe Yin, Yuming Jiang, Shengming Jiang, and Peng Yong Kong. Analysis on Generalized Stochastically Bounded Bursty Traffic for Communication Networks. In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN 2002)*, pages 141–149. IEEE, November 2002. doi:10.1109/LCN.2002.1181778.

[249] Xiang Yu, Ian Li-Jin Thng, Yuming Jiang, and Chunming Qiao. Queueing Processes in GPS and PGPS with LRD Traffic Inputs. *IEEE/ACM Transactions on Networking*, 13(3):676–689, June 2005. doi:10.1109/TNET.2005.850213.

[250] Hui Zhang. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10):1374–1396, October 1995. ISSN 0018-9219. doi:10.1109/5.469298.

[251] Jiandong Zhang, Shasha Qiao, Dujuan Li, and Guoqing Shi. Modeling and simulation of EDF scheduling algorithm on AFDX switch. In *Proceedings of the 2011 IEEE International Conference on Signal Processing, Communications and Computing*, ICSPCC, pages 1–4, September 2011. doi:10.1109/ICSPCC.2011.6061640.

[252] Lixia Zhang, Scott Shenker, and David D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. *ACM SIGCOMM Computer Communication Review*, 21(4):133–147, September 1991. doi:10.1145/115994.116006.

[253] Xuesong Zhang, Xin Chen, Lei Zhang, Guofeng Xin, and Tong Xu. End-to-End delay analysis of avionics Full Duplex switched Ethernet with different flow scheduling scheme. In *Proceedings of the 2011 International Conference on Computer Science and Network Technology*, volume 4 of *ICCSNT 2011*, pages 2252–2258, December 2011. doi:10.1109/ICCSNT.2011.6182423.

[254] Zhi-Li Zhang, Don Towsley, and Jim Kurose. Statistical Analysis of Generalized Processor Sharing Scheduling Discipline. *ACM SIGCOMM Computer Communication Review*, 24(4):68–77, October 1994. ISSN 0146-4833. doi:10.1145/190809.190321.

[255] Zhi-Li Zhang, Don Towsley, and Jim Kurose. Statistical Analysis of the Generalized Processor Sharing Scheduling Discipline. *IEEE Journal on Selected Areas in Communications*, 13(6):1071–1080, August 1995. doi:10.1109/49.400662.

ISBN 978-3-937201-49-8
DOI 10.2313/NET-2015-07-1

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)

ISBN 978-3-937201-49-8
9 783937 201498