ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-044

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « INFORMATIQUE et RESEAUX »

présentée et soutenue publiquement par

Jinbang CHEN

le 21 Septembre 2012

Amélioration des performances des réseaux d'entreprise

Directeur de thèse : **Prof. Guillaume URVOY-KELLER**
Co-encadrement de la thèse : **Prof. Martin HEUSSE**

Jury

M. Ernst BIERSACK, Professeur, EURECOM – France	Président
M. André-Luc BEYLOT, Professeur, IRIT/ENSEEIHT – France	Rapporteur
M. Thierry TURLETTI, Directeur de Recherche, INRIA – France	Rapporteur
M. Martin HEUSSE, Professeur, Laboratoire LIG – France	Examinateur
Mme Lucile SASSATELLI, Maître de Conférences, Laboratoire I3S – France	Examinateur

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

THÈSE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-044

EDITE - ED 130

Doctorat ParisTech

DISSERTATION

for the degree of PhD issued by

TELECOM ParisTech

Specialty « Computer Science and Networking »

presented and publicly supported by

Jinbang CHEN

21/09/2012

Enterprise Networks : Modern Techniques for Analysis, Measurement and Performance Improvement

Thesis advisor : **Prof. Guillaume URVOY-KELLER**
Thesis co-advisor : **Prof. Martin HEUSSE**

Jury

M. Ernst BIERSACK, Professor, EURECOM – France	President
M. André-Luc BEYLOT, Professor, IRIT/ENSEEIHT – France	Reporter
M. Thierry TURLETTI, Directeur de Recherche, INRIA – France	Reporter
M. Martin HEUSSE, Professor, Laboratoire LIG – France	Examiner
Mme Lucile SASSATELLI, Maître de Conférences, Laboratoire I3S – France	Examiner

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

THÈSE

To my grandparents

Deyan and Xiuqiong

Acknowledgments

First of all, I would like to sincerely express thanks to all the jury members and rapporteurs - Ernst BIERSACK, Andre-Luc BEYLOT, Thierry TURLETTI, Martin HEUSSE, Lucile SASSATELLI, and Guillaume URVOY-KELLER, as it is a great honor for me to have them evaluate my work. I greatly appreciate their continuing helps and their valuable comments for the improvement of this manuscript. Thanks to everyone, who has make this thesis possible.

Foremost I would like to thank my advisor Prof. Guillaume Urvoy-Keller for his invaluable support and brilliant ideas. He is a passionate researcher but above all a true gentleman. His never ended support has made this thesis possible. His help started from the first day of this thesis and never stopped until the end of the thesis. His lenience and patience impress me a lot, and it is truly a great experience working with him.

I gratefully acknowledge the precious opportunity I got to collaborate with Prof. Martin Heusse. Thanks for his continuing helps and valuable guidance to me. Martin, together with Guillaume, they have taught me how to conduct research and how to write a decent scientific paper. They have continued helping me a lot in the research, even though I have made so many silly mistakes from the very beginning, either in conducting the experiments or representing the results. I have learned a lot from the discussions with Marin, and I highly appreciate the collaboration with him.

I ought to mention here the special thanks to Prof. Ernst Biersack for his warmly helps (either in the research or the personal cares) and his financial support, especially for the last 7 months of my PhD study. I greatly appreciate the opportunity I had to join Eurecom's networking group headed by Ernst.

I would like to thank all my friends and colleagues at EURECOM and in French Riviera. The list is too long to be put here. But they made my stay very enjoyable during these three and a half years of PhD. In particular, I would like to thank CHEN Qing, your appearance make me believe that the life in this world is always so beautiful. I am blessed indeed by knowing you.

Last but not least, I would express my gratitude to my family for their unconditional love, trust, support and encouragement. I would never thank enough to my father Gaoshua, my mother Liyu and my brother Jincheng for their continuing love, support and understanding as we have been separated for so many years due to my study abroad.

Abstract

As the Internet evolves over the years, a large number of applications emerge, ranging from HTTP, FTP to P2P, and more recently multimedia streaming, on-line game and social networks, with varying service requirements in terms of bandwidth, delay, loss rate and so on. Still, the Internet traffic exhibits a high variability property - the majority of the flows are of small sizes while a small percentage of very long flows contribute to a large portion of the traffic volume. Several studies reveal that small flows are in general related to interactive applications - such examples consist of Web browsing, mail checking, DNS queries, and more recently tweets, posts, chats, etc. - for which one expects to obtain good user perceived performance, most often in terms of short response time. However, the classical FIFO/drop-tail scheme deployed in today's routers/switches is well known to bias against short flows over long ones.

To tackle this issue over a best-effort network, a great deal of size-based scheduling solutions have been proposed in the last decade. The key idea is to favor short flows at the expense of long ones. Although appealing by offering small response time to short flows, most of them feature one or more significant drawbacks/limits: starvation to long flows, scheduling decision based on a single dimension, namely flow size, global modification of end hosts, and the overhead of flow state maintenance. In this thesis, we have proposed a novel and simple scheduling algorithm named EFD (Early Flow Discard), which is able to overcome all the drawbacks aforementioned.

In this manuscript, we first evaluate the performance of EFD in a single-bottleneck wired network through extensive simulations. We then discuss the possible variants of EFD and EFD's adaptations to 802.11 WLANs - mainly refer to EFDACK and PEFD, which keep track of the volumes exchanged in both directions or simply count packets in a single direction, aiming at improving the flow level fairness and interactivity in WLANs. Finally, we devote ourselves to profiling enterprise traffic, and further devise two workload models - one that takes into account the enterprise topological structure and the other that incorporates the impact of the applications on top of TCP - to help to evaluate and compare the performance of scheduling policies in typical enterprise networks.

Résumé

Avec l'évolution récente d'Internet, un grand nombre d'applications sont apparues, allant du P2P au streaming multimédia, du jeu en ligne aux réseaux sociaux, avec différentes exigences de service en termes de bande passante, délai, taux de perte et ainsi de suite. Malgré ces évolution, le trafic Internet présente encore constance qui est sa propriétéde haute variabilité - la majorité des flux sont petits et un petit pourcentage des flux très longs contribuent à une grande partie du volume de trafic. Plusieurs études ont révélées que les flux courts sont en général liés à des applications interactives - navigation sur le Web, chargement email, les requêtes DNS, et plus récemment, tweets, posts, chats, etc. Pour ceux-ci, on s'attend à obtenir de bonnes performances que l'utilisateur percoit, le plus souvent, en termes de temps de réponse courts. Cependant, le schéma classique FIFO/drop-tail déployé dans routeurs/commutateurs aujourd'hui est reconnu comme néfaste pour les flux courts.

Pour résoudre ce probléme sur un réseau best-effort, de nombreuses solutions d'ordonnancement basées sur la taille des flux ont été proposées dans la dernière décennie. L'idée est de favoriser les flux courts au détriment de flux longs. Bien qu'elles soient attrayantes car offrant de petit temps de réponse aux flux courts, la plupart d'entre-elles ont un ou plusieurs des inconvénients/ limites significatifs : la famine des flux longs, une décision d'ordonnancement basée sur une seule dimension, à savoir la taille de flux, la modification nécessaire des hôtes d'extrémité et des coûts mémoire pour garder l'état de flux. Dans cette thèse, nous avons proposé un nouvel et simple algorithme d'ordonnancement appelé EFD (Early Flow Discard), qui est capable de surmonter tous les inconvénients ci-dessus.

Dans ce manuscrit, nous avons d'abord évaluer les performances d'EFD dans un réseau filaire avec un seul goulot d'étranglement au moyen de simulations. Nous discutons aussi des variantes possibles de EFD et les adaptations de EFD à 802.11 WLANs - EFDACK et PEFD, qui enregistrent les volumes échangés dans deux directions ou comptent simplement les paquets dans une direction - visant à améiorer l'équité niveau flux et l'interactivité dans les WLANs. Enfin, nous nous consacrons à profiler du trafic entreprise, en plus d'élaborer deux modéles de trafic - l'une qui considére la structure topologique de l'entreprise et l'autre qui intégre l'impact des applications au-dessus de TCP - pour aider à évaluer et à comparer les performances des politiques d'ordonnancement dans les réseaux d'entreprise classiques.

Contents

List of Figures

List of Tables

Acronyms

Here are the main acronyms used in this document. The meaning of an acronym is usually indicated once, when it first appears in the text.

ACK	Acknowledgment
AP	Access Point
DCF	Distributed Coordination Function
EFD	Early Flow Discard
FIFO	First In First Out
HTTP	Hypertext Transfer Protocol
ISP	Internet Service Provider
LAN	Local Area Network
LAS	Least Attained Service
LASACK	Least Attained Service ACK
LARS	Least Attained Recent Service
MAC	Media Access Control
MSS	Maximum Segment Size
P2P	Peer to Peer
PS	Processor Sharing
QoS	Quality of Service
RTT	Round-Trip Time
SCFQ	Self-Clocked Fair Queueing
SRPT	Short Remaining Processing Time
SYN	Synchronization
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network

Chapter 1

Introduction

1.1 Internet and the TCP/IP protocol

The history of the Internet starts from ARPANET, an experimental data network built in the early 1960s by the U.S. Department of Defense, connecting U.S. universities and the corporate research community for exchange of information. It was originally designed with the ability of individually delivering packets from source to destination through the network. The TCP/IP protocols later developed made it possible to interconnect various networks in the world, providing a universal service. A collection of interconnected networks around the world is nowadays known as the Internet.

The so called Transport Control Protocol (TCP) is based on two principles: regulation and acknowledgment, seminally established by Cerf and Khan [9] in 1974. Later, Jacobson [29] added several key features and brought TCP very close to what it looks like today. TCP (a formal description is given in [47]) is a reliable connection-oriented protocol, which allows to deliver the information from one machine to another in the network without errors. As the dominant packet processing protocol in the Internet, the TCP/IP protocol has continued to evolve over the years to meet the increasing needs of the Internet and of the small, private networks.

The Internet is able to provide a general infrastructure on which a wide range of applications can work well, including web browsing, email, file transfer, remote access, and so on. The ability to support a range of applications is critical, but the challenge becomes sterner and sterner as more and more applications with new needs are deployed in the Internet, such as multimedia streaming, peer-to-peer(P2P), etc. The set of applications dominating the Internet has changed over the last couple of years from HTTP and FTP to P2P applications, and more recently HTTP streaming.

The IP infrastructure is in principle designed to try its best to deliver packets, without providing any guarantee for the service that a packet will receive. In such a network, all users obtain "best effort" service. When a link is congested, packets are dropped as the queue overflows. In case of loss event, retransmission for packets dropped is ensured by TCP. Although such "best effort" service works well for some applications, it can not satisfy the needs of many new applications that are sensitive to packet loss and large latency, like fairly popular multimedia streaming in people's daily life. New architecture for resource allocation is therefore needed for the Internet to support resource assurance and various levels of quality of service (QoS).

1.2 Motivation of the thesis work

As the Internet evolves over the years, many of the new applications emerge with various requirements such as low response time, guaranteed loss rate and data rates. However, the Internet has limited resource management capability inside the network from the time it was originally designed and can not provide any guarantee to end users. Today, the Internet still supports only a best-effort service and the need for service differentiation still persists. To this end, researchers have been trying to re-design the Internet so that different types of application can be simultaneously supported, with minimum service requirements satisfied. As a result, various QoS mechanisms with a set of protocols to dictate the network device to serve contending applications by following a set of pre-defined policies have been proposed. In general, packet scheduling algorithms, together with buffer managements are commonly applied to manage the use of network resources in an efficient manner.

The legacy FIFO/drop-tail scheme deployed in today's routers/switches, is believed to favor long transfers at flow level, which in reverse highly restricts the transmission of short transfers - one sees the need of improvement since short flows are in general related to interactive applications like Email, Web browsing and DNS request/response. The resource sharing issue in computer networks has been studied for decades and many scheduling algorithms were first developed in the context of job scheduling in operating systems. Packet scheduling has been re-activated in the research community in the last decade due to the studies of job size distributions in a variety of computing contexts including Web file sizes, FTP file transfers, UNIX job sizes, and more. In all these cases, job size distribution has been shown to exhibit heavy tails, and be well-modeled by a Pareto distribution, or some other distributions with a power-law tail. This new finding calls for reevaluation of scheduling policies with heavy-tailed workload in the Internet, in particular for size-based scheduling policies.

Motivated by the high variability property of the Internet traffic, a number of size-based scheduling policies have been proposed. The Shortest Remaining Processing Time (SRPT) is known to be optimal [55], in the sense that it minimizes the average response time of transfers. Although appealing, SRPT is impractical as it requires knowledge of flow sizes - which is not achievable for most of the network appliances (router, access point, etc.). Therefore, more attention is given to blind size-based scheduling policies, *i.e.* scheduling policies that are not aware of the flow size. To tackle this issue, several seminal methods have been proposed, i.e. LAS [50], Run2C [5], and LARS [28]. Despite their unique feature – giving low response time to small flows – the main reasons preventing these size-based scheduling approaches from deployment are related to the following concerns: complex flow state keeping, starvation of long flows, taking into account only the accumulated amount of bytes of each flow but not rate, and so on. One of the goal of this work is to look for an alternative scheduling policy that can be used in wired networks in order to improve the overall user perceived performance by favoring short flows without the usual drawbacks associated to size-based schedulers. We also try to resolve

the TCP unfairness problem reported in 802.11 Wireless LANs with the help of scheduling disciplines at network layer, keeping the lower layer (MAC layer) protocol unchanged.
Another motivation behind this work is related to enterprise networks. Today, enterprise networks have evolved from site-centric wired networks where users' machines access application servers through a fixed infrastructure to the case where users are roaming, either from a wired to a wireless network or from inside the company to outside through a VPN access. Moreover, the ever-increasing variety of applications used in Intranets, e.g. voice and video over IP, together with consolidation of servers through virtualization and of data through SAN (Storage Area Networks) both being eventually integrated to offer highly resilient services, have significantly increased the complexity of enterprise networks. We therefore expect new emerging characteristics through the study of modern enterprise traffic. Another goal of this thesis is to explore the new features of enterprise traffic and study the impact of the applications on TCP performance, so as to help modeling enterprise workload.

1.3 Thesis Contributions and Outline

We have made several contributions in this thesis. The first contribution is the proposal of a new size-based scheduling discipline - Early Flow Discard (EFD), which simultaneously fulfills several objectives: (i) Low response time to small flows; (ii) Low bookkeeping cost, *i.e.* the number of flows tracked at any given time instant remains consistently low; (iii) Differentiating flows based on volumes but also based on rate; (iv) Avoiding starvation of long flows. EFD is not limited to a scheduling policy but also incorporates a buffer management policy, where the packet with smallest priority gets discarded when the queue gets full, as opposed to drop tail which blindly drops packets upon arrival. In Chapter 4, we evaluate the performance of EFD in wired network under flow size distribution with heavy tail property using several metrics, and compare it to other state of the art scheduling policies (LAS, Run2C and LARS - the legacy FIFO is incorporated as well for the comparison as FIFO is the current de facto standard). We consider two load regimes - underload and overload. In general, we show through extensive simulations that EFD outperforms or at least obtains a similar performance as other existing seminal policies, with the advantage of significant overhead saving on flow tracking. In addition, we further demonstrate EFD's ability of efficiently protecting low/medium rate multimedia transfers.
The second contribution is the analytic model development for EFD, which helps explaining the simulation results. In Chapter 5, we first review the commonly used models for FIFO, SCFQ, LAS and Run2C, and use them to validate our simulation results. We then present the difficulty of deriving an analytic model for EFD discipline by digging into the relationship between flows and subflows fragmented from original flows. As a starting point, we attempt to explain the flow-level results based on subflow-level analysis, but we fail mainly due to two reasons: on one hand,

the subflow size distribution in high priority queue is much less skewed than the original flow size distribution, but not exactly deterministic; on the other hand, the inter-arrival process of subflows in high priority queue is no longer Poisson process. We then switch to the problem of relating subflow level performance to flow level performance in EFD. We finally develop a model which is able to successfully link between flows and subflows, and reproduce the simulation results in an analytical way.

The third contribution of this thesis is the analysis of EFD and its variants' applicability in an 802.11 Wireless LAN environment, in which the TCP unfairness problem is addressed. EFD was originally designed in wired network and evaluated for the case of single direction flows. In contrast, data flows in both two directions and two direction transfers share the wireless medium (which is "half duplex") in 802.11 networks. In addition, the Access Point (AP) buffer size is typically small, therefore it tends to built up. In Chapter 6, two ways for the adaptation of EFD in 802.11 WLANs are proposed: keep track of the volumes exchanged in both directions or simply count packets in a single direction. We evaluate the performance of EFD and its adaptations, and compare them with state of the art scheduling disciplines. For the performance investigation, we consider several factors: (i) two different workloads - long live connections and mixed of short and long transfers; (2) small and large Access Point buffer size; (3) various symmetric level between uploads and downloads. Simulation results show that, the two variants of EFD - PEFD and EFDACK, are able to enforce a good level of fairness without paying a penalty in terms of performance degradation. Furthermore, PEFD and EFDACK can effectively improve performance in wireless networks, without the usual drawbacks associated to size-based schedulers. We raise the concern of buffer granularity in Chapter 7, which inspires from our study of size-based scheduling disciplines over 802.11 Wireless LANs in Chapter 6. We term the buffer granularity as the unit in which the buffer size of the network device interface is measured. In Chapter 7, we investigate the impact of the buffer granularity (instead of the buffer sizing) on the performance of scheduling disciplines over 802.11 WLANs. The discussion is conducted with two buffer granularities - packets and bytes, and two workload scenarios. We investigate the bottleneck link capacity sharing between uploads and downloads considering as metrics the aggregate throughput for the case of long-lived connections, and mean conditional response time in the case of more realistic workload with heavy-tailed size distribution. We conclude that measuring the buffer with the unit of bytes is highly preferred for FIFO, Run2CACK and BEFD, while LASACK, LARS and SCFQ are insensitive to the buffer granularity.

Our fourth contribution is the enterprise traffic profiling, which is conducted in Chapter 8. We develop an understanding of the basic characteristics of modern enterprise traffic at various levels based on a medium-size laboratory packet trace (Eurecom). The significant contribution is to contrast the external and internal activity in modern enterprise networks. As an additional issue, a supervised machine learning approach is proposed to find an automatic way to identify different roles (servers or clients) inside the enterprise networks.

The final contribution of this thesis, in Chapter 9, is the two new workload models proposed for enterprise network. The first model specifies how traffic flows in intranet and Internet traffic, and in two directions respectively based on the new findings of the enterprise traffic pattern through the study. The second model replays the workload extracted from the real trace, thus taking into account the impact of the applications on top.

Chapter 2

State of the Art

2.1 Size-based Scheduling

Size-based scheduling has received a lot of attention from the research community with applications to Web servers [56], Internet traffic [5, 52, 58] or 3G networks [2, 33]. The key idea is to favor short flows at the expense of long ones because short flows are in general related to interactive applications like Email, Web browsing or DNS requests/responses; unlike long flows which represent background traffic. Such a strategy pays off as long as long flows are not completely starved and this generally holds without further intervention for Internet traffic where short flows represent a small portion of the load and thus cannot monopolize the bandwidth.

Classically, size-based scheduling policies are divided into blind and non-blind scheduling policies. A blind size-based scheduling policy is not aware of the job[1] size while a non-blind is. Non blind scheduling policies are applicable to servers [56] where the job size is related to the size of the content to transfer. A typical example of non blind policy is the Shortest Remaining Processing Time (SRPT) policy, which is optimal among all scheduling policies, in the sense that it minimizes the average response time. To achieve this property, SRPT relies on a simple strategy: always service the client that is the closest to completion.

For the case of network appliances (routers, access points, etc.) the job size, i.e. the total number of bytes to transfer, is not known in advance. Several blind size-based scheduling policies have been proposed. The Least Attained Service (LAS) policy [50] bases its scheduling decision on the amount of service received so far by a flow. LAS is known to be optimal if the flow size distribution has a decreasing hazard rate (DHR) as it becomes, in this context, a special case of the optimal Gittins policy [18]. Some representatives of the family of Multi-Level Processor Sharing (MLPS) scheduling policies [30] have also been proposed to favor short flows. An MLPS policy consists of several levels corresponding to different amounts of attained service of jobs, with possibly a different scheduling policy at each level. In [5], Run2C, which is a specific case of MLPS policy, namely PS+PS, is proposed and contrasted to LAS. With Run2C, short jobs, which are defined as jobs shorter than a specific threshold, are serviced with the highest priority while long jobs are serviced in a background PS queue. Run2C features key characteristics: (i) As (medium and) long jobs share a PS queue, they are less penalized than under LAS; (ii) It is proved analytically in [5] that a M/G/1/PS+PS queue offers a smaller

[1] Job is a generic entity in queueing theory. In the context of this work, a job corresponds to a flow.

average response time than an M/G/1/PS queue, which is the classical model of a network appliance featuring a FIFO scheduling policy and shared by homogeneous TCP transfers; (iii) Run2C avoids the lock-out phenomenon observed under LAS [28], where a long flow might be blocked for a large amount of time by another long flow.

Run2C and LAS share a number of drawbacks. Flow bookkeeping is complex. LAS requires to keep one state per flow. Run2C needs to check, for each incoming packet, if it belongs to a short or to a long flow. The latter is achieved in [5] thanks to a modification of the TCP protocol so as to encode in the TCP sequence number the actual number of bytes sent by the flow so far. Such an approach, which requires a global modification of all end hosts, is questionable[2]. Moreover, both LAS and Run2C classify flows based on the accumulated number of bytes they have sent, without taking the flow rate into account.

Least Attained Recent Service (LARS) is a size-based scheduling designed to account for rates [28]. It consists in a variant of LAS, where the amount of bytes sent by each flow decays with time according to a fading factor β. LARS is able to handle differently two flows that have sent a similar amount of bytes but at different rates and it also limits the lock out duration of one long flow by another long flow to a maximum tunable value.

Despite their unique features, size-based scheduling policies have not yet been moved out of the lab. We believe the main reasons behind this lack of adoption are related to the following general concerns about size-based scheduling approaches:

- Size-based scheduling policies are in essence state-full: each flow needs to be tracked individually. Even though one can argue that those policies should be deployed at bottleneck links which are presumably at the edge of network – hence at a location where the number of concurrent flows is moderate – the common belief is that stateful mechanisms are to be avoided in the first place.

- Size-based scheduling policies are considered to overly penalize long flows. Despite all its drawbacks, the legacy scheduling/buffer management policy, FIFO/drop tail, does not discriminate against long flows while size-based scheduling solutions tend to impact both the mean response time of flows but also their variance as long flows might lock-out each others.

- As their name indicates, size-based scheduling policies consider a single dimension of a flow, namely, its accumulated size. Still, persistent low rate transfers often convey key traffic, *e.g.*, voice over IP conversations. As a result, it seems natural to account both for the rate and the accumulated amount of bytes of each flow.

A number of works, such as Run2C and LARS presented above, address partially the aforementioned shortcomings of size-based scheduling policies. Still, to the best

[2]Other works aim at favoring short flows, by marking the packets at the edge of the network so as to relieve the scheduler from flow bookkeeping [42]. However, the deployment of DiffServ is not envisaged in the near future at the Internet scale.

of our knowledge, none of them fulfill simultaneously the above objectives. In this thesis, we propose a new scheduling policy, EFD, that addresses of these objectives simultaneously. We first study its performance in a wired network in Part I and then in a wireless network in Part II.

2.2 Performance improvement in 802.11 Wireless LANS by using size-based scheduling policies

In a typical infrastructure 802.11 WLAN, mobile stations equipped with 802.11 interface communicate with an Access Point (AP) on a wireless channel, and the AP relays traffic to and from the wired network. In many cases, e.g. the enterprise, the wireless LAN is the performance bottleneck as users typically use a link with 100 Mbit/s or higher capacity to access the Internet today.
Different from wired network, wireless LAN features two key properties – on one hand, the protocol is half-duplex, meaning that uploads and downloads share the wireless medium; on the other hand, the Access Point is not granted a high enough priority to access the medium under DCF, which means that its queue, which is typically 30 to 100 packets, tends to build up.
A key performance problem, known as "TCP Unfairness" [46] occurs when TCP traffic is conveyed over an 802.11 network. This unfairness problem stems from the equal opportunity access to the wireless medium of the AP and the wireless stations in a wireless cell. Since all mobile stations exchange traffic with the wired network solely through the AP, the latter deserves to be given more chance to access the wireless channel but it is restricted by the equal access method defined by the standard 802.11 DCF (Distributed Coordination Function), leading to the fact that the AP becomes a bottleneck that limits the overall throughput by losing frames because of buffer overflow. Moreover, when TCP traffic are conveyed over wireless LAN, the competition between TCP ACKs from the uploads and TCP data packets from the downloads at the buffer of the access point even worsens the unfairness and eventually degrades the overall performance – as the buffer at the access point which is typically small, tends to build up, resulting in packet losses - recall that TCP reacts differently to the loss of data packets and ACKs.
Many authors have proposed solutions to address the TCP unfairness problem at various layers: transport, network, or MAC layer [46, 7, 37, 27, 58]. Pilosof *et al.* [46] proposed to modify the receiver window in TCP ACKs to pace sources on wireless stations and provide in this way more bandwidth for the download traffic. Several authors proposed to solve the unfairness problem by using an adequate MAC access method. Leith *et al.* [34, 35] proposed to choose suitable parameters of IEEE 802.11e to provide fairness between competing TCP uploads and downloads. AAP (Asymmetric Access Point) [38, 26, 20] sets the contention window of the AP to a constant value while wireless stations use the Idle Sense access method. Idle Sense is a alternative MAC protocol to 802.11 that varies the contention window using a AIMD approach, so as to achieve a higher fairness than the legacy DCF that

tends to punish a few stations when contention is observed. In contrast, with Idle Sense, all the stations have a similar contention window that varies according to the global amount of transmission attempts on the medium that a station might estimate by observing the wireless channel. With this way, the AP is able to obtain twice transmission capacity of the sum of all active stations independently of the number of contending stations.

Other authors considered solutions at the IP level, leaving the lower layer protocols, especially the MAC layer unchanged. Several size-based scheduling policies have been proved to be able to enforce fairness among TCP transfers, and at the same time to improve the reactivity of short connections and interactive applications. LASACK [58] as an extension of LAS, mitigates the impact of the non responsiveness of TCP ACK streams by assigning a priority to a TCP ACK packet that is a function of the number of bytes sent by the corresponding data stream. In this way, LASACK enforces fairness among upload and download TCP connections and improve the interactivity perceived by the end users. The latter is defined as the ability of the network to maintain small response time to the short flows that are generated by the interactive applications of the users, such as email and web browsing. LARS [28] that applies a temporal decay to the volume of data associated with each flow, offers similar performance to LASACK, but avoids lock-out and takes into account both the volume and the rate for scheduling.

Size-based scheduling policies are highly recommended to be used to 802.11 wireless LANs to improve flow level fairness and interactivity, as they are deployed at IP level of the access point only, leaving other layers' protocol unchanged.

2.3 Enterprise Networks

We now present key results obtained in the analysis of enterprise networks since, in the last part of this work, we present results of the analysis of a large trace captured at Eurecom and present preliminary results of the use of this trace and the information collected on the network to devise new simulation workload models. We exemplify the use of these workload models on some of the size-based scheduling policies we studied in the first two parts of the thesis.

Wide-area Internet traffic has been widely studied in many different environments from the research communities over the years [8, 22, 15, 36, 44, 6]. However, the traffic pattern and the performance issue within modern enterprise networks remains nearly unexplored. The likely reason lies in the difficulty of adequately monitoring enterprise traffic and the belief of good performance of enterprise networks in practice.

We aim to present an overview of research activities focusing on the issue of enterprise networks. In general, the vast majority of studies make use of measurements collected in wired or wireless enterprise networks, consisting of campus, research labs, *etc.* Most studies of enterprise network usually rely on packet or flow level traces, complemented with other sources such as SNMP or syslog data.

A great deal of studies relied on traces captured at an enterprise's access link, from which the network activity involving the external Internet can be easily characterized, but it does not shed any light on the activity within the enterprise networks. Recently, studies have been conducted upon the measurements made at an enterprise's core routers [43, 40]. They do not rely on any advance data mining technique, but rather report descriptive statistics to infer performance of enterprise networks. Some other studies have measured the communication on the end-hosts themselves [17]. With this method, all traffic related to each end host is incorporated for the analysis, including the traffic traversing the boundary of the enterprise in the communication between local and remote peers outside the enterprise. However, it lacks of a knowledge of what is happening in the surrounding, such as the network load. Authors in [41] presented a number of techniques for calibrating packet traces captured at different Ethernet switch ports, like leveraging TCP semantics to identify measurement loss, employing expected replication of broadcast packets to point to missing events from traces, and so on.

The authors in [43] provided a first characterization of internal enterprise traffic recorded at a large size site – LBNL (Lawrence Berkeley National Laboratory). The packet traces span more than 100 hours, over which activity from a total of several thousand internal hosts appears, although they could not capture at a given time instance all the traffic flowing inside the network, as given the large size and even more the complex structure of the LBNL network. They first looked at the basic information of internal and external traffic volume, coming up with a broad breakdown of the main components of the traffic. They also looked at the locality of traffic sources and destinations by examining the fan-in and fan-out of local peers, given that some local peers are servers accessible from the Internet. They finally examined characteristics of the applications that dominate the traffic. This article is mostly descriptive, but they pinpointed some specific phenomena like the existence of failures to establish specific connections internally. They also addressed load problem from the end hosts point of view by computing the amount of TCP retransmissions experienced by connections. They observed that TCP retransmission rate can reach up to 1%, which is much less than the observation for Internet traffic but still surprisingly large for intranet traffic.

In [40], the authors present an initial step towards understanding TCP performance in enterprise networks. In particular, they based their analysis on a dataset consisting of switch-level packet traces taken at LBNL over a few months, which is the same as the one used in [43]. They assessed the prevalence of broken TCP transactions, application used, throughput of TCP connections, and phenomena that influence performance, such as retransmissions, out-of-order delivery, and packet corruption. In general, they confirmed the common presumption that enterprise connections enjoy low loss rates.

To the best of our knowledge, no study has been conducting using large and recent traces collected in an enterprise network like the one we collected at Eurecom. From this perspective, the measurement study that we carry out in the last part of this thesis is the first of its kinds. A difficulty is however to assess its representatively.

We however encounter here a problem that is faced by most of traffic analysis studies done by the measurement community, even for Internet traces, as for example, the various habits of user lead to different observations in traffic traces collected for European, American and Asian ISPs.

Part I

Resource Allocation in Wired Networks

Chapter 3

Challenge in Flow State Keeping

3.1 Introduction

Scheduling policies significantly affect the performance of resource allocation systems. In the context of the Internet, scheduling, together with a number of mechanisms (admission control, active queue management, etc.) are widely discussed and deployed in oder to support applications with varying service requirements - delay constrains such as multimedia streaming applications and high throughput requirement such as file transfer.
Studies have shown that Internet traffic exhibits a high variability property: most of the TCP flows are short, while more than 50% of the bytes are carried by less than 5% of the largest flows. Given the heavy-tailed nature of Internet flow-size distribution [61], size-based scheduling which basically bases its decision on the amount of bytes, has received more and more attention from the research community. Among all scheduling policies, the Shortest Remaining Processing Time (SRPT) is proved to be optimal [55], in the sense that it minimizes the avervage response time. Although the performance improvement over the classical FIFO discipline is tremendous, SRPT is difficult to deploy in practice as it requries knowledge of flow sizes[1] whereas the flow size, *i.e.* the total number of bytes to transfer is not known in advance for many network appliances (routers, access points, etc.). The deployment of SRPT has been proposed for Web servers by Mor Harchol-Balter *et al.* in [25]. In addition, a common concern with SRPT as other preemptive disciplines, is the danger of starving long flows. Even more, even if the size of the flow at the time of arrival of its first packet was known, the SRPT scheduler needs to keep track of the number of remaining packets to be served for each flow. The logistics associated with counting packets for each flow is complex and grows with the number of connections over a link on the Internet which can be very large, especially in the core of network. As in practice, flow sizes are typically not known to the scheduler in most cases, hence scheduling disciplines which are not aware of the flow sizes are well worth being explored. To overcome the main disadvantage preventing SRPT from deployment, several blind size-based scheduling solutions have been proposed and analyzed to improve the performance of short transfers. As the literature in this research area is vast, we limit the references to a small but important subset. By and large, the improvement is achieved by favoring short flows. The Least Attained Service (LAS) policy [50], which bases its scheduling decision on the total amount of service received

[1] We use "flows" in the context of network systems, instead of the more general term "jobs" even though they are equivalent to each other in this thesis report.

so far by a flow, is known to be optimal if the flow size distribution has a decreasing hazard rate (DHR). As a variant of LAS, the Least Attained Recent Service (LARS) [28] additionally applies a temporal decay to the service obtained by a flow - with the ability of limiting the lock-outs and accounting for volumes and rates simultaneously. As a representative of Multi-Level Processor Sharing (MLPS) policies, Run2C [5] deploys a PS+PS model and simply uses a strict threshold to differentiate between short and long flows. Although appealing in terms of improvement on data transfer response time, most of the work dealing with giving preferential treatment based on size share an important drawback, that is the schedulers do maintain per-flow state information[2], in particular to keep track of the number of packets that have been serviced so far for each flow for LAS and LARS - which is challenging as the number of flows in progress is in the order of hundreds of thousands under a high load. Even though one can argue that those polices should be deployed at bottleneck links which are presumably at the edge of network - hence at a location where the number of concurrent flows is moderate, the common belief is that stateful mechanisms are to be avoided in the first place. Even more, note that the number of concurrently active flows on the Internet sees significantly increasing for both the core and the edge of the network as the Internet traffic expands day by day, that per-flow state keeping is impractical.

3.2 Overhead reduction of flow state keeping

To highlight the importance of the memory resources required to satisfy the requirement for flow state maintenance, we consider the case of an infinite queue. As aforementioned, full flow state keeping may dramatically increases the cost of the network appliances - in which the scheduler is implemented, including the memory consumption and the processing power, in particular for the case in which high speed RAM is used for storage. Therefore, it makes sense to relieve the scheduler from flow state keeping so as to decrease the overhead and accelerate the processing.
We term "statelessness" the property of a scheduler to not keep any state concerning the ongoing flows it is servicing. Run2C achieves this property albeit at the cost of a modification of TCP. The DiffServ [42][3] paradigm can also be seen as stateless mechanisms as flows are marked at the edges and the elements in the core need only to read DSCP to take their scheduling decision. Recall that a small amount of long flows contribute to the majority of the Internet traffic load. As such, if we are able to properly identify long flows in a certain manner so as to maintain limited flow states for these long flows only instead of all flows, a significant overhead-saving for flow state keeping will be naturally obtained, retaining the desirable property of providing low response time to short flows at the same time as short flows are

[2]In Run2C [5], they propose to modify the TCP sequence number to indicate the amount of bytes so far for each flow, which can achieve statelessness. If so, all size-based scheduling will benefit from it. However, this proposal is not fully acceptable in practice, since it requires a global modification of all end hosts.

[3]Note that, the deployment of DiffServ is not envisaged in the near future at the Internet scale.

preferentially served over long flows.
One possible solution to tackle the flow state keeping issue is exposed in [31]. The idea is to apply a probabilistic method to detect long flows[4]. A similar idea called SIFT is also proposed in [48]. Basically, two PS queues (packets in PS queue are drained in FIFO order at packet level) are maintained, and the packets of long flows are enqueued in a low priority queue, while the packets of short ones are preferentially served from a high priority queue. The main idea is to perform a probabilistic test on every arriving packet independently, and store the flow id of each sampled flow along with the total number of sampled packets of each sampled flow. Once this number exceeds some threshold given, all future packets from the flow are forwarded to the low priority queue. Such a flow is called long in this scenario. We called a packet is sampled if it succeeds in the probabilistic test, and furthermore a flow is said to be sampled if the number of sampled packets of this flow exceeds the sampling threshold. Note that "sampled" means "selected" here. Algorithm 2 shows the scheme in pseudo-code. As long flows normally consist of many packets, this approach is expected to be able to identify long flows with a very high probability.

Algorithm 2 : Probabilistic sampling scheme in pseudo-code, from [49]

```
1: if packet_arrival then
2:    flow_id = get_flowid_from_packet;
3:    if sampled_packets(flow_id) > threshold then
4:       forward_to_low_priority_queue;
5:    else
6:       forward_to_high_priority_queue;
7:       if test_success == true then
8:          sampled_packets(flow_id) ++;
9:       end if
10:   end if
11: end if
```

We next discuss the choice of the probability for sampling and the effect of different sampling threshold values on the performance. During the discussion, we point out that this approach is problematic as noticeable amount of short flows[5] are misclassified as long flows and vice versa in case the sampling threshold equals one (i.e. a flow is sampled once one of its packets is sampled), resulting in a limited cost saving (further discussion in detail is given in Chapter 4 Section 4.6.1). If we increase the sampling threshold - meaning that a flow is tagged as long flow in case more than one packet from it are sampled, the false positives (a short flow is misclassified as a

[4]Note that this mechanism is proposed in [31] to do admission control function and not a scheduling.

[5]Note that there are several ways to define short flows. Here in this section, we call a short flow is a flow with size less than a given size threshold, and vice versa for long flows.

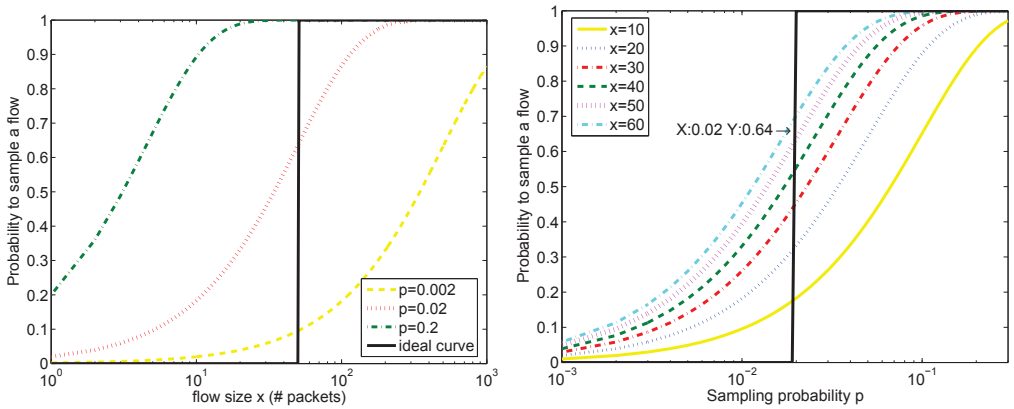
long one) are reduced but the implementation of the algorithm becomes complex. Let Ts denote the sampling threshold, and N denote the packets arrived of a flow until the flow is sampled. For a flow to be sampled, one must have $N \geq Ts$. Let p be the probability to sample a packet in a test. Then, the probability $P_s(x)$ that a flow of size x is sampled equals

$$P_s(x) = 1 - \sum_{i=0}^{T_s-1} \frac{x!}{i!(x-i)!} p^i (1-p)^{x-i} \tag{3.1}$$

Let T denotes the threshold used by a deterministic scheme to partition flows into short and long flows - an experimental value is usually assigned to this threshold in the literatures. Like in Run2C [5], the authors in [48] use a threshold of 20 packets to differentiate between short and long flows. Since SIFT-like probabilistic sampling scheme classifies flows in a randomized fashion, it is reasonable to choose a probability p and a sampling threshold Ts so that on average it takes T packets until a flow is sampled, i.e. the expected number of packets until a flow is sampled equals T.
Thus, for $Ts = 1$, meaning that a flow is sampled once one of its packets is sampled, Equation 3.1 reduces to

$$P_s(x) = 1 - (1-p)^x \tag{3.2}$$

It is easy to see that N follows a geometric distribution with an average of $1/p$ in case $Ts = 1$. Hence, the probability p should be equal to $1/T$. Figure 3.1 (a) and (b) plot $P_s(x)$, the probability that a flow of size x is sampled, respectively as a function of x for various value of p and as a function of p for various value of x. T is set to 50, so that the ideal p value is 0.02. We observe from Figure 3.1 (a) that, a noticeable amount of small flows are sampled even if p is low as 0.02. Enlarging or reducing p will make the situation even worse as large p such as $p = 0.2$ leads to more short flows to be sampled with high probability, while small p such as $p = 0.002$ prevent long flows from being sampled. Figure 3.1 (b) verifies that $p = 0.02$ is the best choice in case $T = 50$ but is still far away from the expectation as short and long flows can not be strictly partitioned by $T = 50$. The ideal case is indicated in solid line in black. In summary, although the scheme is simple and easy to implement when the sampling threshold Ts equals to one, there is no way to avoid false positive. In addition, there are always false small flows sampled that will be put into the low priority queue, leading to a degradation of their performance.
To compensate the error, or equivalently to reduce the misclassification in terms of either false positive or false negative, a straightforward solution is to increase the sampling threshold Ts. For $Ts > 1$, a flow is sampled if at least Ts of its packets are sampled. In this case $E(N)=Ts/p$ and hence $p=Ts/T$. Note that this procedure converges to the deterministic scheme as Ts increases. When Ts is equal to T, i.e. $p = 1$ - meaning that a flow is classified as long flow once it has generated more than T packets, this scheme degenerates to Run2C in which a strict threshold is used for flow differentiation.



(a) Flow sampling probability as a function of flow size

(b) Flow sampling probability as a function of packet sampling probability

Figure 3.1: Flow sampling probability Ps, in which Ts=1

Figure 3.2 plots $P_s(x)$, the probability that a flow of size x is sampled, as a function of x for various values of Ts. T is set to 50. It is evident that the higher the value of Ts, the better the classification. However, increasing the value of sampling threshold Ts complicates the implementation and does not improve the overhead of flow state keeping, although it reduces the misclassification, in particular for false positives.

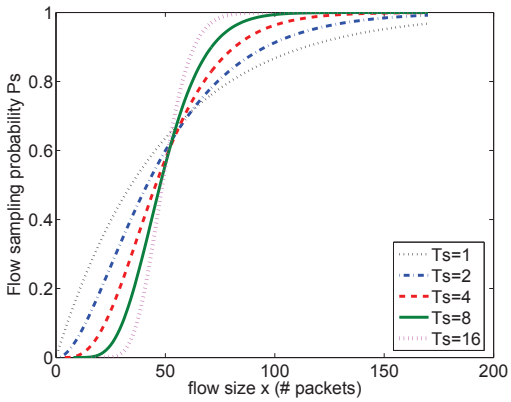


Figure 3.2: Flow sampling probability as a function of flow size

From an implementation point of view, it is best to use $Ts = 1$. However, small values of Ts increase the probability of misclassification. False positives need to be avoided in any case, although the authors in [49] argue that the fraction of misclassified flows is quite low for heavy-tailed flow size distribution - they take an example of bounded Pareto distribution with parameters: the smallest flow size $m = 1$, the largest flow size $M = 10^6$, the shape parameter $\alpha = 1.1$, together with $Ts = 1$, $T = 100$, $p = Ts/T = 0.01$, end up with the value of 2.6% for the fraction

of misclassified flows among all flows.

3.3 Conclusion

As non-blind size-based scheduling policies are not applicable to devices within the network where the flow size can not be guessed, attentions has shifted to blind size-based scheduling disciplines. However, many blind size-based scheduling policies such as LAS and LARS, do need to maintain per-flow state information - which is problematic as the overhead of full flow state keeping becomes dramatic for Internet traffic. Other scheduling disciplines like Run2C, propose to modify the TCP sequence numbers to indicate the amount of bytes sent so far for each flow, consequently avoiding flow state keeping - this not only makes the scheme TCP dependent, but also reduces the randomness of initial sequence numbers. To tackle this issue, schemes like SIFT have been proposed. Single sampling (i.e. $Ts = 1$) is not sufficient as false positives exist - which is harmful to the performance of short flows, whereas multiple sampling is able to reduce false positives as the sampling threshold increases, but at the same time increases the complexity of the implementation and maintains a large amount of flow states since many short flows are sampled in the first sampling.
In conclusion, the approach to maintain the flow state in the size-based scheduling policies proposed so far can be categorized as follows:

- Full flow state approach as in LAS. An argument in favor of keeping one state per active flow is that the number of flows to handle remains moderate as it is expected that such a scheduling policy be implemented at the edge of the Internet.

- No flow state approach: an external support is provided to the scheduler, either at the end-hosts by modifications of the transport layer [5] or by some intermediate boxes, as in the case of a DiffServ scheme [42], that marks the packets. The extent of the changes required to the Internet architecture prevents the deployment of such approaches in a near future.

- Probabilistic approaches: a test is performed at each packet arrival for flows that have not already be incorporated in the flow table [11, 31, 48]. The test is calibrated in such a way that only long flows should end up in the flow table. Still, false positives are possible. Several options have been envisioned to combat this phenomenon especially, a re-testing approach [48] or an approach where the flows in the flow table are actually considered as long flows once they have generated more than a certain amount of packets/bytes after their initial insertion [11].

In this thesis work, we are seeking a scheduling discipline, which is simple and easy to implement, with the ability of significantly reducing the overhead of flow state

keeping with orders of magnitude, and retaining the good property of size-based scheduling disciplines - improving data transfer response times.

Chapter 4

EFD: An Efficient Low-Overhead Scheduler

4.1 Introduction

Despite their unique features, size-based scheduling policies have not yet been moved out of the lab. We believe the main reasons behind this lack of adoption are related to the following general concerns about size-based scheduling approaches:

- As discussed in the previous chapter, size-based scheduling policies are in essence state-full: each flow needs to be tracked individually. Even though one can argue that those policies should be deployed at bottleneck links which are presumably at the edge of network – hence at a location where the number of concurrent flows is moderate – the common belief is that stateful mechanisms are to be avoided in the first place.

- Size-based scheduling policies are considered to overly penalize long flows. Despite all its drawbacks, the legacy scheduling/buffer management policy, FIFO/drop tail, does not discriminate against long flows while size-based scheduling solutions tend to impact both the mean response time of flows but also their variance as long flows might lock-out each others.

- As their name indicates, size-based scheduling policies consider a single dimension of a flow, namely, its accumulated size. Still, persistent low rate transfers often convey key traffic, *e.g.*, voice over IP conversations. As a result, it seems natural to account both for the rate and the accumulated amount of bytes of each flow.

A number of works address partially the aforementioned shortcomings of size-based scheduling policies. Although, to the best of our knowledge, none of them fulfill simultaneously the above objectives. This chapter presents a new scheduling policy, EFD (Early Flow Discard) that aims at fulfilling the following objectives: (i) Low response time to small flows; (ii) Low bookkeeping cost, *i.e.*, the number of flows tracked at any given time instant remains consistently low; (iii) Differentiating flows based on volumes but also based on rate; (iv) Avoiding lock-outs.
EFD manages the physical queue of an interface (at the IP level) as a set of two virtual queues corresponding to two levels of priority: the high priority queue first and the low priority queue at the tail of the buffer. Formally, EFD belongs to the family of Multi-Level Processor Sharing policies and is effectively a PS+PS

scheduling policy. *The key feature of EFD is the way flow bookkeeping is performed. In EFD, we keep an active record only for flows that have at least one packet in the queue. This simple approach allows to fulfill the entire list of objectives listed above. Specifically, in EFD the active flow table size is bounded to a low value.* Also, although EFD has a limited memory footprint, it can discriminate against bursty and high rate flows. EFD is not limited to a scheduling policy but also incorporates a buffer management policy, where the packet with smallest priority gets discarded when the queue is full, as opposed to drop tail which blindly discards packets upon arrival. This mechanism is similar to the one used in previous works [50, 11].
Section 4.2 gives an overview of the related works mentioned above. Section 4.3 presents the proposed scheduling scheme. The simulation environment, including network setup, network topology and workload appear in Section 4.4. Then we use simulations to evaluate its performance and compare with other schedulers in Section 4.6. Finally we conclude this chapter in Section 4.7.

4.2 Related Work

With respect to the criteria listed previously (low memory footprint, deadlock avoidance, ability to take into account rate and not only the size), we now review the pros and cons of LAS, LARS and Run2C. Run2C and LAS share a number of drawbacks. Flow bookkeeping is complex. LAS requires to keep one state per flow. Run2C needs to check, for each incoming packet, if it belongs to a short or to a long flow. The latter is achieved in [5] thanks to a modification of the TCP protocol so as to encode in the TCP sequence number the actual number of bytes sent by the flow so far. Such an approach, which requires a global modification of all end hosts, is questionable[1]. Moreover, both LAS and Run2C classify flows based on the accumulated number of bytes they have sent, without taking the flow rate into account.
As discussed in Chapter 3, some approaches propose to detect long flows by inserting the flow in the table probabilistically [11, 48, 31]. The key idea here is to perform a simple random test (with a low probability of success) upon packet arrival to decide if the corresponding flow should be inserted in the table. As long flows generate many packets, it is unlikely to miss them, while many short flow simply go unnoticed. These approaches differ in the way they trade false positive rate against the speed of detection of a long flow.
So far, a single work addresses the problem of accounting for rates in size-based scheduling [28]. With the Least Attained Recent Service policy (LARS), the amount of bytes sent by each flow decays with time according to a fading factor β. LARS is able to handle differently two flows that have sent a similar amount of bytes but at different rates and it also limits the lock out duration of one long flow by another

[1] Other works aim at favoring short flows, by marking the packets at the edge of the network so as to relieve the scheduler from flow bookkeeping [42]. However, the deployment of DiffServ is not envisaged in the near future at the Internet scale.

long flow to a maximum tunable value.

4.3 Early Flow Discard

In this section, we describe how EFD manages space and time priority. EFD belongs to the family of Multi-Level Processor Sharing scheduling policy. EFD features two queues. The low priority queue is served only if the high priority queue is empty. Both queues are drained in a FIFO manner at the packet level (which is in general modeled as a PS queue at flow level). In terms of implementation, a single physical queue for packet storage is divided into two virtual queues. The first part of the physical queue is dedicated to the virtual high priority queue while the second part is the low priority queue. A pointer is used to indicate the position of the last packet of the virtual high priority queue. This idea is similar to the one proposed in the Cross-Protect mechanism [31]. We now turn our attention to the flow management in EFD and the enqueuing and dequeuing operations. We also discuss the spatial policy used when the physical queue gets full.

4.3.1 Flow management

EFD maintains a table of active flows, where flows are defined here as the sets of packets that share a common identity, consisting of a 5-tuple: source and destination addresses, source and destination ports and protocol number. Flows remain in the table as long as there is one corresponding packet in the buffer and discarded when the last packet leaves. Consequently, a TCP connection (or UDP transfers) may be split over time into several fragments handled independently of each other by the scheduler. Note that unlike most scheduling mechanisms that keep per flow states, EFD does not need to use any garbage collection mechanism to clean its flow table. This happens automatically upon departure of the last packet of the flow. A flow entry keeps track of several attributes, including flow identity, flow size counter, number of packets in the queue.

4.3.1.1 Packet enqueuing

For each incoming packet, a lookup is performed in the flow table of EFD. A flow entry is created if the lookup fails and the packet is put at the end of the high priority queue. Otherwise, the flow size counter of the corresponding flow entry is compared to a preset threshold *th*. If the flow size counter exceeds *th*, then the packet is put at the end of the low priority queue; otherwise the packet is inserted at the end of the high priority queue. The purpose of *th* is to favor the start of each flow. In our simulations, we use a *th* value of 20 packets (up to 30 KB for packets of 1500 bytes each). Obviously, if a connection is broken into several fragments, from the scheduler's perspective, then each time it will handle each fragment as a unique one and assign the start (within threshold *th*) of each fragment a high priority, by directing all packets making up the start of each fragment into the high priority

queue. We believe that this makes sense as this happens only if the connection has not been active for a significant time –it has not been backlogged for a while– and thus can be considered as fresh.

4.3.1.2 Packet dequeuing

When a packet leaves the queue or gets dropped, it decreases the number of queued packets of the corresponding flow entry. The flow entry stays in the table as long as at least one of its packets is in the queue. So **the flow table size is bounded by the physical queue size** in packets[2]. Indeed, in the worst case, there are as many entries as distinct flows in the physical queue, each with one packet.

This policy ensures that the flow table remains of small size. Also if a flow sends at high rate for a short period of time, its packets will be directed to the low priority queue only for the limited period of time during which the flow is backlogged: EFD is sensitive to flow burstiness.

4.3.2 Buffer management

When a packet arrives to a queue that is full, EFD first inserts the arriving packet to its appropriate position in the queue, and then drops the packet that is at the end of the (physical) queue. This buffer policy implicitly gives space priority to short[3] flows, which differs from the traditional drop-tail buffer management policy. This approach is similar to the Knock-Out mechanism of [11] and the buffer management proposed to LAS in [50]. As large flows in the Internet are mostly TCP flows, we can expect that they will recover from a loss event with a fast retransmit; unlike short flows that might time out.

Algorithm 3 represents the algorithm in pseudo-code, which assists in the description of the EFD scheduling. Note that the flow states are efficiently managed in EFD by dropping flow entries from the flow table as soon as the last packet of a flow in the flow table leaves the queue. Therefore, the existence of a flow entry in the flow table, implies that there is at least one of its packets currently present in the queue.

4.4 Performance Evaluation Set Up

In this section, we present the network set up – network topology and workload – used to evaluate the performance of EFD and to compare it to other scheduling policies. All simulations are done using QualNet [1].

[2] In most if not all active equipments – routers, access points – queues are counted in packets and not in bytes.

[3] Due to the discussion in the above paragraph, a short flow is a part of a connection whose rate is moderate.

Algorithm 3 : Early Flow Discard algorithm

```
1: function packet_arrival(p)
2: # A new packet p of flow F arrives
3: if no packets of F are present in the queue then
4:    create a flow entry R(F) for F;
5:    # p is a high priority packet
6:    if the queue is full then
7:       if only high priority packets in the queue then
8:          p is dropped;
9:          return;
10:      else
11:         the last packet of low priority queue is dropped;
12:         p is inserted at the end of high priority queue;
13:      end if
14:   else
15:      p is inserted at the end of high priority queue;
16:   end if
17: else
18:   # at least one packet of F reside in the queue, so that a flow entry for F exists in the table
19:   if number of bytes already served of flow F < threshold th  then
20:      # p is a high priority packet
21:      if the queue is full then
22:         if only high priority packets in the queue then
23:            p is dropped;
24:            return;
25:         else
26:            the last packet of low priority queue is dropped;
27:            p is inserted at the end of high priority queue;
28:            update the flow entry R(F) in the table;
29:         end if
30:      else
31:         p is inserted at the end of high priority queue;
32:         update the flow entry R(F) in the table;
33:      end if
34:   else
35:      # p is a low priority packet
36:      if the queue is full then
37:         p is dropped;
38:         return;
39:      else
40:         p is put at the end of low priority queue;
41:         update the flow entry R(F) in the table;
42:      end if
43:   end if
44: end if
45:
46: function packet_departure(p)
47: # A packet p of flow F leaves due to the end of service or dropping
48: if no more packets of flow F are in the queue after p leaves then
49:    the flow entry R(F) is deleted from the table;
50: else
51:    update the flow entry R(F) in the table;
52: end if
```

4.4.1 Network Topology

We evaluate the performance of EFD and compare it to other scheduling policies for the case of a single bottleneck network, using the classical dumbbell topology depicted in Figure 4.1.

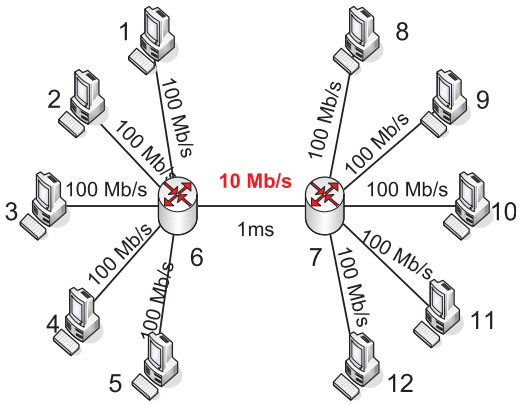


Figure 4.1: Network topology

A group of senders (nodes 1 to 5) are connected to a router (node 6) by 100Mbps bandwidth links and a group of receivers (nodes 8 to 12) are connected to another router (node 7) with a 100Mbps bandwidth link. The two aggregation routers are connected to each other with a link at 10Mbps. All links have 1 ms propagation delay.
All nodes use FIFO queues, except the bottleneck node which uses one of the four scheduling policies that we compare in this work: FIFO, LAS, RuN2C or EFD. The bottleneck buffer has a finite size of 300 packets. This value might be considered as quite high. Note that we will discuss the use of EFD in a WLAN context in Chapter 6.

4.4.2 Workload generation

Data transfer requests arrive according to a Poisson process, the server and the client are picked at random and the content requested is distributed according to a bounded Zipf distribution. A bounded Zipf distribution is a discrete analog of a continuous bounded Pareto distribution.
Transfers are performed over TCP or UDP depending on the simulation. In all cases, the global load is controlled by tuning the arrival rate of requests. For each simulation set-up, we consider an underload and an overload regime, which correspond respectively to workloads of 8 and 15 Mb/s (80% and 150% of the bottleneck capacity). For TCP simulations, we use the GENERIC-FTP model of Qualnet, which corresponds to an unidirectional transfer of data. For UDP transfers, we use a CBR application model where one controls the inter-packet arrival time. The latter enables to control the exact rate at which packets are sent to the bottleneck.

In both TCP and UDP cases, IP packets have a size of 1500 bytes.

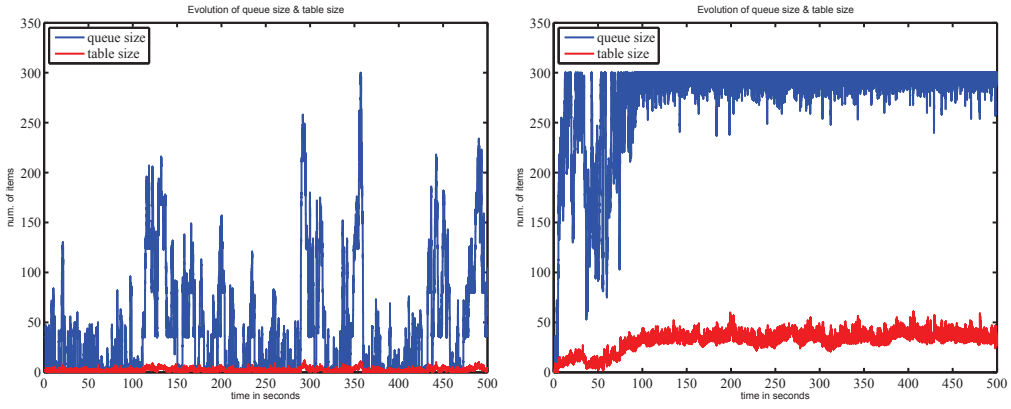
4.5 EFD Internal Dynamics

In this section, we present a detailed analysis of the way EFD operates. We focus on the following aspects:

- The evolution of the flow table size;
- How traffic is split between the low and high priority queue.
- How connections are fragmented by the scheduler due to the insertion/removal process within the flow table;

4.5.1 Flow table

Due to the discarding mechanism of flow entries within the flow table in EFD, a flow entry exists in the flow table only if at least one packet of the flow resides in the finite queue. An important consequence is that the flow table size is bounded by the physical queue size in packets[4]. Indeed, in the worst case, there is as many entries as distinct flows in the physical queue, each with one packet.
For the TCP workload, we plot in Figure 4.2 the instantaneous queue size and the instantaneous flow table size in both two regimes: underload and overload. Remind that the buffer size is 300 packets in our experiments. Figure 4.2 reveals that both flow table and packet queue grow up as the traffic intensity increases, but the table size is consistently below the queue size. Even in the overload case, the flow table size remains fairly small. We further investigate this issue in Section 4.6.1.



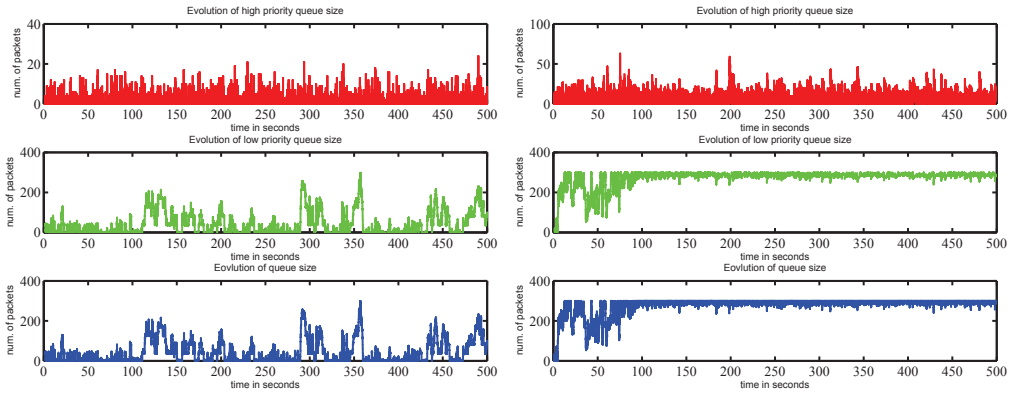
(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 4.2: Queue size & Table size

[4]In most if not all active equipments – routers, access points – queues are counted in packets and not in bytes.

4.5.2 Virtual queue sizes

As EFD features two queues with high and low priority respectively from the implementation point of view, Figure 4.3 depicts the evolution of the two virtual queue sizes, together with the overall queue (*i.e.* physical queue) size in underload and overload. One clearly sees that the low priority queue carries the bulk of the traffic. It is in line with our expectation as we want the high priority queue to be lightly loaded so that packets can be served as fast as possible, in order to grant short flows with low mean response times.



(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 4.3: Evolution of high and low priority queue size in both underload and overload.

4.5.3 Flow fragmentation

With EFD, a connection can be fragmented into many flows - each one is treated as fresh by the scheduler. In addition, the packets of one of these flows might be partly serviced in high priority or low priority queue: the first *th* packets are serviced by the high priority queue and the rest by the low priority queue. We call this phenomenon "flow fragmentation". It is in clear contrast to FIFO, LAS and RuN2C.
In practice, several phenomena can lead to break a connection into many fragments. For instance, during connection establishment, the TCP slow start algorithm limits the number of packets in flight so that it does not continuously occupy the buffer. This is however not a problem, as those flows are smaller than *th* and thus the start of the TCP transfer will receive a high priority. If the flow lasts longer and it is effectively able to use its share of the capacity, then the connection will eventually occupy the buffer without interruption and therefore stay in the flow table. Figure 4.4 illustrates such a scenario. It is apparent that, as the connection size increases, the number of fragments tends to reach a limit so that, for the longest connections, a small number of fragments correspond to many packets.
To get a better understanding of the way EFD partitions traffic among the low and high priority queues, we present in Figure 4.5(a) the distributions of transfer

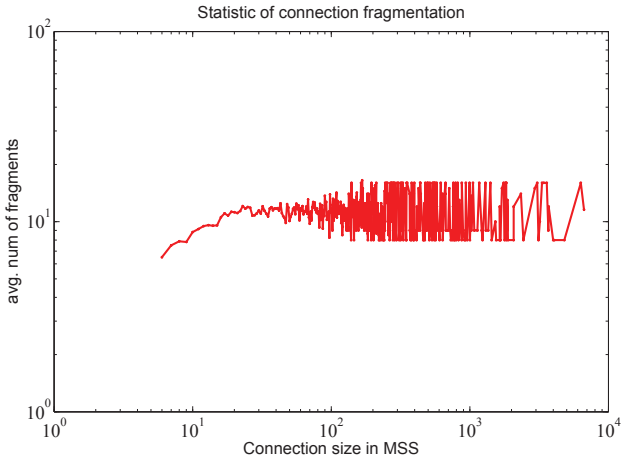
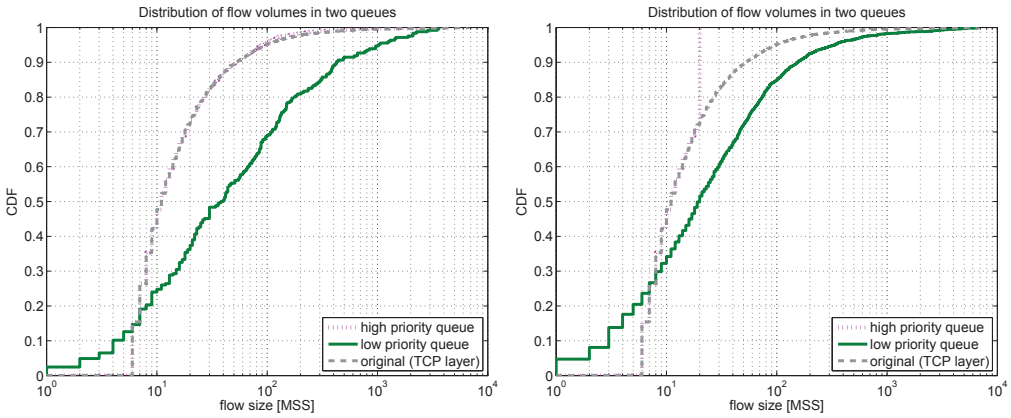


Figure 4.4: Number of segments per connection - workload of 8Mbit/s

sizes in both queues. Due to the way EFD operates, a given transfer is broken in possibly many flows or fragments from the scheduler's viewpoint. In addition, the *th* first packets of each flow are serviced by the high priority queue and the rest, if any by the low priority queue. For each TCP transfer, we sum the total number of packets serviced at the high priority queue and the low priority queue respectively over all the segments of this transfer. We further add the original distribution of transfer sizes (at the TCP layer). We observe from Figure 4.5(a) that the distribution of flow sizes in the low priority queue consists of larger flows than in the high priority queue, even though long transfers can be partially or fully serviced in the high priority queue. This behavior of EFD is in contrast to Run2C, which is another Multi-Level Processor Policy, with the same number of levels and policy at each, but that adopts a fixed threshold per transfer: the first *th* packet goes to the high priority queue while the rest goes to the low priority queue – see Figure 4.5(b). Clearly, EFD imposes a higher load on the high priority queue as compared to Run2C. This should not be interpreted as a drawback of EFD as compared to Run2C since it allows EFD to account for rates and not only for volumes, as we further illustrate with the UDP experiments that we describe below.

4.6 Performance Evaluation

In this section, we compare the performance of EFD to other scheduling policies. Our objective is to illustrate the ability of EFD to fulfill the 4 objectives listed in the introduction, namely (i) low bookkeeping cost, (ii) low response time to small flows, (iii) avoiding lock-outs, (iv) protecting long lasting delay sensitive flows.
To illustrate the first 3 items, we consider a TCP workload with homogeneous transfers, i.e., transfers that take place on paths having similar characteristics. For the last item - protecting long lived delay sensitive flows - we add a UDP workload



(a) EFD (b) RuN2C

Figure 4.5: Distribution of flow volumes in two queues - workload of 8Mbit/s

to the TCP workload in the form of a CBR traffic, in order to highlight the behavior of each scheduler in presence of long lasting delay sensitive flows.

4.6.1 Overhead of flow state keeping

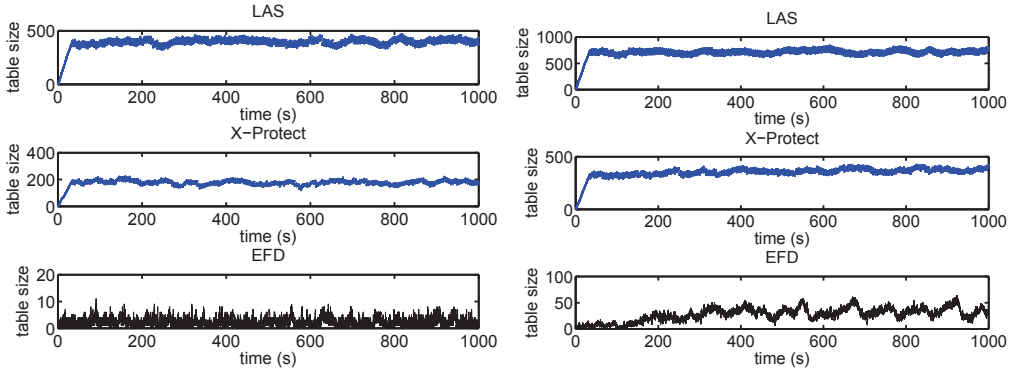
The approaches to maintain the flow table in the size-based scheduling policies proposed so far can be categorized as follows:

- Full flow table approach as in LAS [50]. An argument in favor of keeping one state per active flow is that the number of flows to handle remains moderate as it is expected that such a scheduling policy be implemented at the edge of the Internet.
- No flow table approach: an external mechanism marks the packets or the information is implicit (coded in the TCP SEQ number in Run2C) [5, 42]
- Probabilistic approaches: a test is performed at each packet arrival for flows that have not already be incorporated in the flow table [11, 31, 48]. The test is calibrated in such a way that only long flows should end up in the flow table. Still, false positives are possible. Several options have been envisaged to combat this phenomenon especially, a re-testing approach [48] or an approach where the flows in the flow table are actually considered as long flows once they have generated more than a certain amount of packets/bytes after their initial insertion [11].
- EFD deterministic approach: the EFD approach is fully deterministic as flow entries are removed from the flow table once they have no more packet in the queue.

In this section, we compare all the approaches presented except the "No flow table approach" for our TCP workload scenario. We consider one representative of

each family: LAS, X-Protect and EFD. We term X-Protect a Multi-Level Processor Scheduling policy that maintains two queues, similarly to Run2C, but uses the probabilistic mechanism proposed in [31] to track long flows[5]. As for the actual scheduling of packets, X-Protect mimics Run2C based on the information it possesses. If the packet does not belong to a flow in the flow table nor passes the test, it is put in the high priority queue. If it belongs to a flow in the flow table, it is put either in the high priority queue or in the low priority queue, depending on the amount of bytes sent by the flow. We use a threshold of 30KB, similar to the one used for EFD.

The evolution of flow table size over time for load of 8Mbit/s (underload) and 15Mbit/s (overload) are shown in Figure 4.6. For LAS and X-Protect, the flow table is visited every 5 seconds and the flows that have been inactive for 30 seconds are removed.

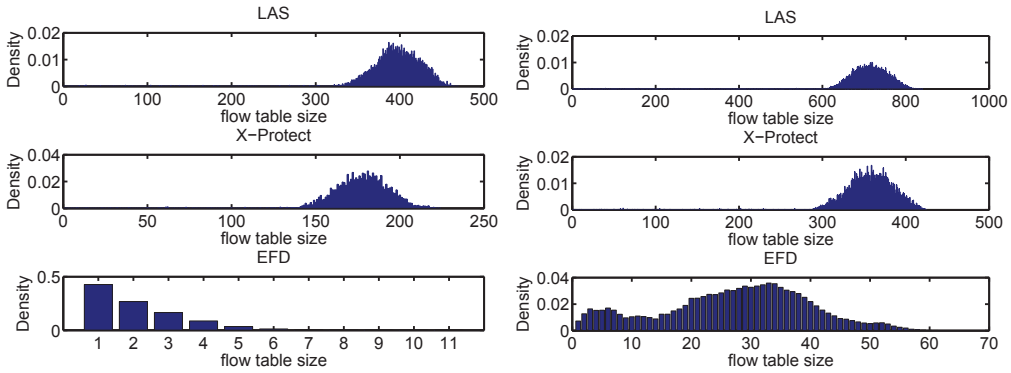


(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 4.6: Evolution of flow table size over time

We observe how X-Protect roughly halves the number of tracked flows, compared to LAS. By contrast, EFD reduces it by one order of magnitude. The reason why X-Protect offers deceptive performance is the race condition that exists between the flow size distribution and the probabilistic detection mechanism. Indeed, even though a low probability, say 1%, is used to test if a flow is long, there exists so many short flows that the number of false positives becomes quite large, which prevents the flow table from being significantly smaller than in LAS. The histograms in Figure 4.7 confirm the good performance of EFD in underload and also overload, as EFD keeps the flow table size to a few 10s of entries at most. Note that this is clearly smaller than the actual queue size (300 packets) that constitutes an upper bound on the flow table size in EFD as explained before. We finally report the mean value and the 95% level confidence interval of the flow talbe size over 1000 seconds simulation for both load conditions in Table 4.1.

[5]Note that this mechanism is proposed in [31] to do admission control function and not a scheduling.



(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 4.7: Histogram of the flow table size

Table 4.1: Statistics - flow table size

		LAS	X-Protect	EFD
8Mbit/s (underload)	mean	392.4961	176.8407	1.9997
	95%-CI	[392.3759, 392.6163]	[176.7861, 176.8952]	[1.9962, 2.0032]
15Mbit/s (overload)	mean	704.0326	352.9599	27.9053
	95%-CI	[703.8622, 704.2030]	[352.8678, 353.0519]	[27.8824, 27.9282]

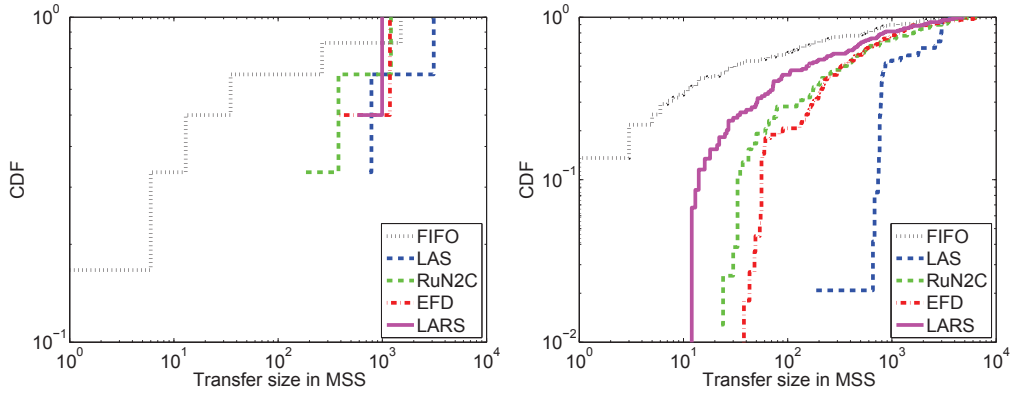
4.6.2 Mean response time

Response time is a key metric for a lot of applications, especially interactive ones. An objective of EFD and size-based scheduling policies in general is to favor interactive applications, hence the emphasis put on response time. We consider four scheduling policies: FIFO, LAS, Run2C and EFD. FIFO is the current de facto standard and it is thus important to compare the performance of EFD to this policy. LAS can be considered as a reference in terms of (blind) size-based scheduling policies as a lot of other disciplines have positioned themselves with respect to LAS. Run2C, for instance, aims at avoiding the lock out of long flows observed more often with LAS than for *e.g.* FIFO. We do not consider the X-protect policy discussed in Section 4.6.1, as Run2C can be considered as a perfect version of X-protect since Run2C distinguishes packets of flows below and above the threshold *th* (we use the same threshold *th* for both EFD and Run2C).

Response times are computed only for flows that complete their transfer before the end of the simulation. When comparing response times, one must thus also consider the amount of traffic due to flows that terminated their transfer and to flows that did not complete. The lack of completion of a flow can be due to a premature end of simulation. However, in overload and for long enough simulations as in our case, the main reason is that they were set aside by the scheduler.

We first turn our attention to the aggregate volumes of traffic per policy for the underload and overload cases. We observe no significant difference among the schedul-

ing policies in terms both of number of complete and incomplete connections. The various scheduling policies lead to a similar level of medium[6] utilization.
In contrast, when looking at the distribution of incomplete transfers, it appears that the flows killed by the different scheduling policies are not the same. We present in Figure 4.8 the distribution of incomplete transfers where the size of a transfer is the total amount of MSS packets transferred at the end of the simulation. A transfer is deemed incomplete if we do not observe a proper TCP tear down with two FIN flags. As expected, we observe that FIFO tends to kill a lot of small flows while the other policies discriminate long flows.

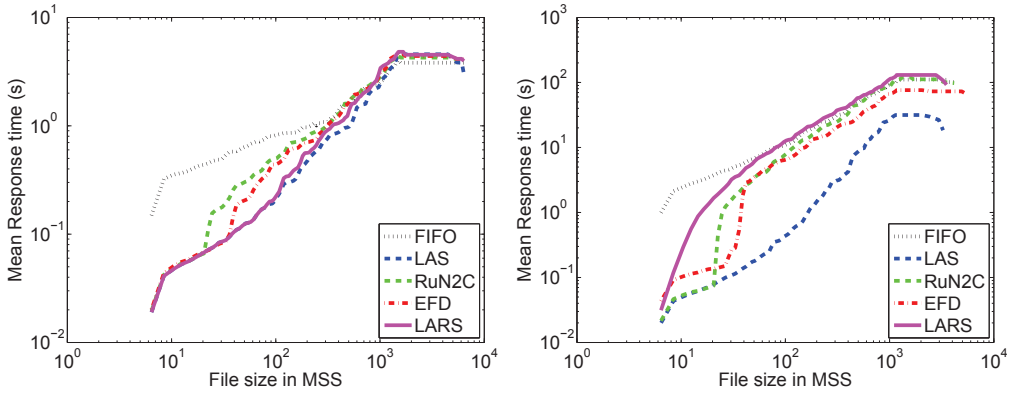


(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 4.8: Distributions of incomplete transfers size

Distributions of the response times for the (complete) short and long transfers in underload and overload conditions are presented in Figure 4.9. Under all load conditions, LAS, EFD and Run2C manage to significantly improve the response time of the short flows as compared to FIFO. EFD and Run2C offer similar performance. They both have a transition of behavior at about th value ($th = 20$ MSS). Still, the transition of EFD is smoother than the one of Run2C. This was expected as Run2C applies a strict rule: below or above th for a given transfer, whereas EFD can further cut a long transfer into fragments which individually go first to the high priority queue. Overall, EFD provides similar or slightly better performance than Run2C with a minimal price in terms of flow bookkeeping. LAS offers the best response time of size-based scheduling policies in our experiment for small and intermediate size flows. For large flows its performance are equivalent to the other policies in underload and significantly better for the overload case. However, one has to keep in mind that in overload conditions, LAS deliberately killed a large set of long flows (see Figure 4.8), hence its apparent better performance. LARS behaves similarly to LAS in underload and degrades to fair queueing –which brings it close to FIFO in this case– when the networks is overloaded.
As a complement to Figure 4.9, we plot the mean value together with the 95%

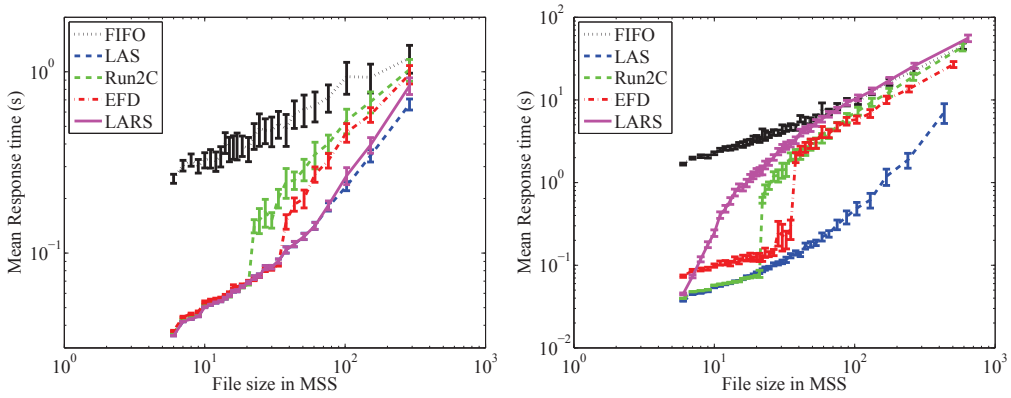
[6]The medium is the IP path as those policies operate at the IP level.



(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 4.9: Conditional mean response time

confidence interval of the response time over the flow size in Figure 4.10. Remember that the distribution of flow sizes generated exhibits high variability - meaning that small number of longest flows carry the majority of traffic load. Thus, it is problematic when calculating the confidence interval of flow response time, especially for long flows as the number of long flows collected from the workload is limited. To handle this issue approximately, we accumulate the samples by starting from a certain flow size and spanning adjacent flow sizes if exist in an ascending order until the number of samples reaches a threshold value given (threshold value equals to 200 for example), during which the mean value of all flow sizes traversed is taken as the flow size to pair with the confidence interval. Although flow size spans a limited range of values (up to 300 MSS) compared to Figure 4.9 (up to 9000 MSS) by taking the processing method presented above, Figure 4.10 reinforces the credibility of the simulation results.



(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 4.10: Confidence interval of response time over flow size

Given the flows which have completed their transfers before the end of the simulation - meaning that two FINs have been observed, we next partition them into short and long ones with the definition that long flows contribute to 50% of the traffic load. This classification method coming from experimental study has the advantage that the meaning of short and long flows is consistently similar for both load regimes given in Table 4.2. We further summarize the mean value, along with the 95% level confidence interval of data transfer response time for short and long flows respectively in Table 4.2. It makes sense as we are able to intuitively observe the improvement the new size-based scheduling brings from these statistic data in a synthetic way. Table 4.2 confirms the ability of giving small response time to short flows with negligible penalty on long flows of size-based schedulings (LAS, LARS, Run2C and EFD) as compared to the legacy FIFO - in particular for the case of underload, in line with the results illustrated in Figure 4.9.

Table 4.2: Performance Statistics - 300MSS buffer - 10Mbit/s bottleneck link

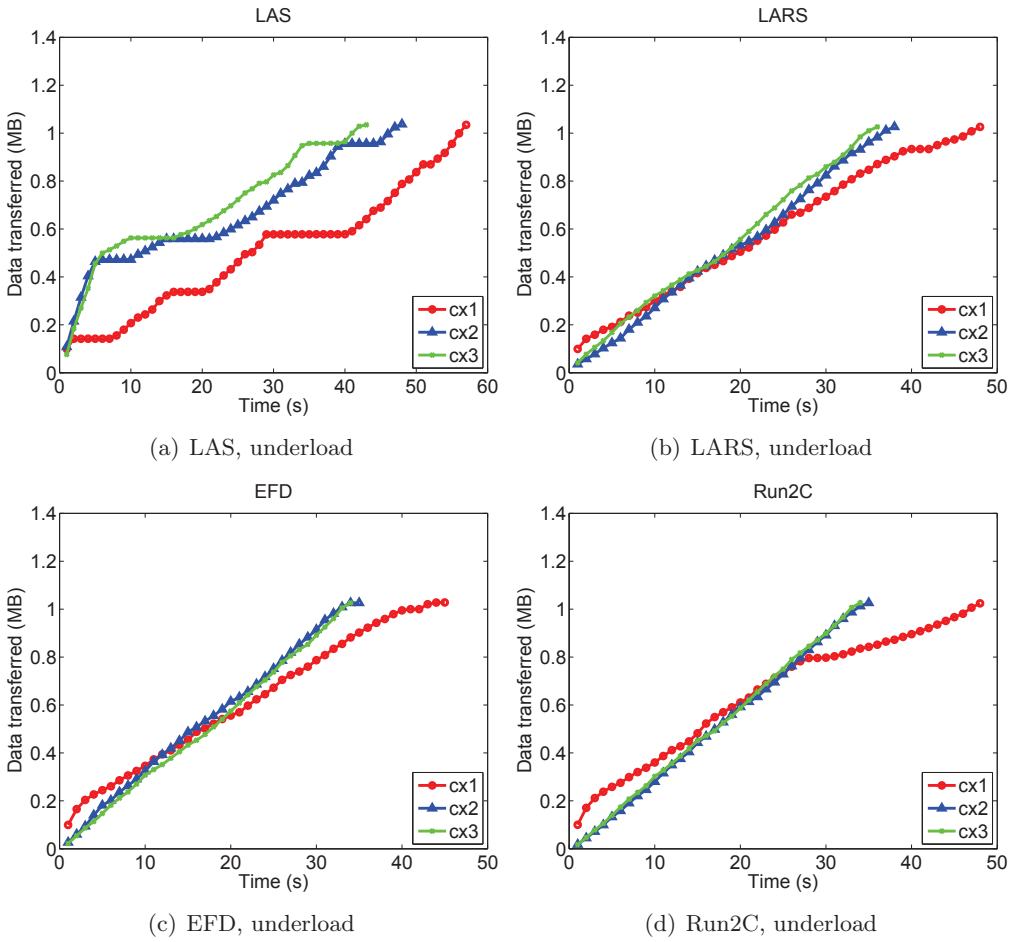
			8Mbit/s - underload		15Mbit/s - overload	
			short flows	long flows	short flows	long flows
Mean size (MSS)			21	1020	21	1086
Response time (seconds)	mean	FIFO	0.390	3.519	3.264	55.788
		LAS	0.070	3.944	0.104	34.517
		Run2C	0.108	3.861	0.848	58.406
		EFD	0.090	3.758	0.705	40.014
		LARS	0.073	3.842	1.521	64.882
	95%-CI	FIFO	[0.379, 0.400]	[2.709, 4.329]	[3.211, 3.317]	[48.951, 62.625]
		LAS	[0.069, 0.071]	[2.727, 5.161]	[0.099, 0.109]	[24.970, 44.064]
		Run2C	[0.104, 0.112]	[2.942, 4.779]	[0.808, 0.888]	[49.902, 66.911]
		EFD	[0.087, 0.093]	[2.887, 4.628]	[0.673, 0.738]	[34.397, 45.630]
		LARS	[0.071, 0.075]	[2.886, 4.798]	[1.471, 1.572]	[56.686, 73.078]

4.6.3 Lock-outs

The low priority queue of EFD is managed as a FIFO queue. As such, we expect EFD, similarly to Run2C, to avoid lock-outs observed under LAS whereby an ongoing long transfer is blocked for a significant amount of time by a newer transfer of significant size. This behavior of LAS is clearly observable in Figure 4.11(a) where the progress (accumulated amount of bytes sent) over time of the 3 largest transfers of one of the above simulations[7]. We indeed observe large periods of times where the transfers experience no progress, which leads to several plateaus. This is clearly in contrast to the cases of LARS, EFD and to a lesser extent of Run2C, for the same connections, shown in Figures 4.11(b), 4.11(c) and 4.11(d) respectively. The

[7]Those 3 connections did not start at the same time, the time axis is relative to their starting time. *Therefore, it is different from the case of long-live connections in which a long flow might be blocked by a newly arriving long flow and the bandwidth is then fairly shared by both after the same amout of bytes have been obtained.*

progress of the connections in the latter cases is indeed clearly smoother with no noticeable plateau.



(a) LAS, underload

(b) LARS, underload

(c) EFD, underload

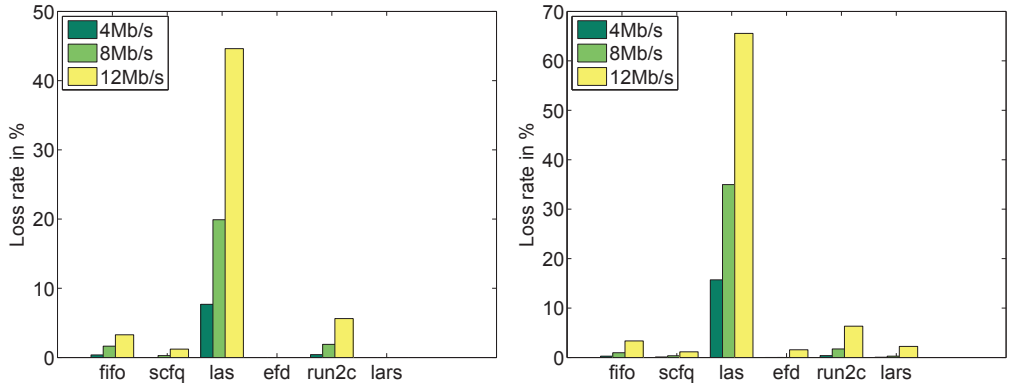
(d) Run2C, underload

Figure 4.11: Time diagrams of the 3 largest TCP transfers under LAS, LARS, EFD and Run2C (underload), relative to the start of each transfer

4.6.4 The Case of Multimedia Traffic

In the TCP scenario considered above, FTP servers were homogeneous in the sense that they had the same access link capacity and the same latency to each client. The transfer rate was controlled by TCP. In such conditions, it is difficult to illustrate how EFD takes into accounts the actual transmission rate of data sources. In this section, we have added a single CBR flow to the TCP workload used previously.
We consider two rates 64Kb/s and 500Kb/s for the CBR flow, representing typical audio (e.g., VoIP) and video stream (e.g., YouTube video - even though the YouTube uses HTTP streaming) respectively. The background load also varies - 4, 8 and 12Mbps- which correspond to underload/moderate/overload regimes as the

bottleneck capacity is 10 Mbps. To avoid the warm-up period of the background workload, the CBR flow is started at time t=10s and keeps on sending packets continuously until the end of the simulation. The simulation lasts for 1000 seconds. Since small buffers are prone to packet loss, we assign to the bottleneck a buffer of 50 packets, instead of 300 packets previously. The loss rates experienced by the CBR flow are given in Figure 4.12, in which a well-known fair scheduling scheme called SCFQ [19] is added for the comparison, in addition to the disciplines mentioned before.



(a) a CBR flow with rate of 64Kb/s

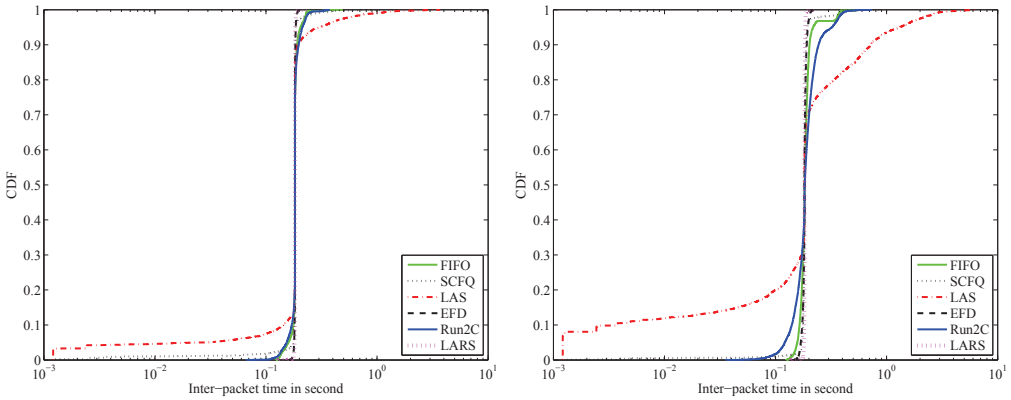
(b) a CBR flow with rate of 500Kb/s

Figure 4.12: Loss rate experienced by a CBR flow in various background loads

As we can see from the figure, for the case of a CBR flow with rate of 64Kbps, LAS discards a large fraction of packets even at low load. This was expected as LAS only considers the accumulated volume of traffic of the flow and even at 64 Kbps, the CBR flow has sent more than 8 MB of data in 1000 s (without taking the Ethernet/IP layers overhead into account). In contrast, FIFO, SCFQ and Run2C offer low loss rates in the order of a few percents at most. As for EFD and LARS, they effectively protect the CBR flow under all load conditions.

To further analyze this behavior, we next examine the inter-departure time distribution of the CBR flow as in [28]. We do not report results for all three background loads but simply pick two of them - 4Mbps and 12Mbps as they are representative for the illustration. We present them in Figure 4.13 for a CBR flow with rate of 64Kbps. We observe from Figure 4.13 that, LAS serves packets in batch (many packets have short delay between them) while EFD and LARS forward packets in a much more regular way as most packets have almost the same delay between them in both background load regimes. In addition, the jitter apparently ramps up under LAS and Run2C as the background traffic grows from 4Mbps (underload) to 12Mbps (overload).

As the rate of the CBR flow increases from 64Kbps to 500Kbps, no packet loss is observed for EFD in underload/moderate load conditions, similarly to SCFQ, whereas the other scheduling disciplines (FIFO, LAS, Run2C and LARS) are hit at various degrees. In overload, EFD and LARS blow up similarly to LAS (which still



(a) CDF of inter-departure times for a CBR flow with 4Mb/s background traffic

(b) CDF of inter-departure times for a CBR flow with 12Mb/s background traffic

Figure 4.13: Jitter of a CBR flow with rate of 64Kb/s

represents an upper bound on the loss rate as the CBR flow is continuously granted the lowest priority). EFD behaves slightly better than LARS as the load in the high priority queue is by definition lower under EFD than under Run2C.

When looking at the above results from a high level perspective, one can think at first sight that FIFO and SCFQ do a decent job as they provide low loss rates to the CBR flow in most scenarios (under or overload). *However, those apparently appealing results are a side effect of a well-known and non desirable behavior of FIFO. Indeed, under FIFO, the non responsive CBR flow adversely impacts the TCP workload, leading to high loss rates. This is especially true for the CBR flow working at 500 kbps. SCFQ tends to behave similarly if not paired with an appropriate buffer management policy [19].* In contrast, LARS and EFD offer a nice trade-off as they manage to simultaneously grant low loss rates to the CBR flow with a low penalty to the TCP background workload. Run2C avoids the infinite memory of LAS but still features quite high loss rates since the CBR flow remains continuously stuck in the low priority queue.

Overall, EFD manages to keep the desirable properties of size-based scheduling policies and in addition manages, with a low bookkeeping cost, to protect multimedia flows as it implicitly accounts for the rate of this flow and not only its accumulated volume.

4.7 Conclusion

In this chapter, we have proposed a simple but efficient packet scheduling scheme called *Early Flow Discard* (EFD) that uses a fixed threshold for flow discrimination while taking flow rates into account at the same time. EFD possesses the key feature of keeping an active record only for flows that have one packet at least in the queue. With this strategy, EFD caps the amount of active flow that it tracks to the queue

size in packets.
Extensive network simulations revealed that EFD, as a blind scheduler, retains the good properties of LAS like small response times to short flows. In addition, a significant decrease of bookkeeping overhead, of at least one order of magnitude is obtained as compared to LAS, which is convincing from a practical point of view. Lock-outs which form the Achilles' heel of LAS are avoided in EFD, similarly to Run2C. In contrast to LAS and Run2C, EFD inherently takes both volume and rate into account in its scheduling decision due to the way flow bookkeeping is performed. We further demonstrated that EFD can efficiently protect low/medium multimedia flows in most situations.
Naturally, we discuss in next chapter the analytic model for EFD scheduling policy. We expect kind of M/G/1 like model for EFD. However, things become more complex for EFD as a flow can be randomly broken into several new flows under EFD, and theoretically describing this behavior is challenging. In addition, as another extension of EFD, we explore in the second part of this thesis the applicability of EFD to WLAN infrastructure networks, where the half-duplex nature of the MAC protocol needs to be taken into account [58].

Chapter 5

Analytic Model for EFD Discipline

5.1 Motivation

Taking advantage of the heavy tail property of Internet traffic distribution, size-based scheduling methods - basing their scheduling decision on the amount of data transfered, have been extensively studied and proved to be able to greatly enhance the responsiveness by favoring small transfers, which in general represent interactive applications in real life scenarios. Although appealing compared to the legacy FIFO, in terms of improving end user interactivity, shortcomings along with representative state of the art size-based schedulers have been continuously addressed and discussed - LAS/LARS and Run2C need to keep track of the volumes conveyed by each and every ongoing connections explicitly or implicitly, while EFD manages to significantly limit the overhead of flow bookkeeping; LAS may in particular starve long transfers, whereas LARS, Run2C and EFD are able to diminish or even eliminate it in various manners; And finally, taking rate into account when scheduling seems to be necessary since more and more rate-hungry applications are launched to the Internet nowadays, for which LARS/EFD are able to cope with but not LAS/Run2C.

Given the good performance demonstrated through extensive simulations, our objective in this chapter is to develop analytic models to estimate the average flow transfer time as a function of flow size for representative size-based scheduling policies (LAS, Run2C, EFD, and incorporate FIFO as the baseline for comparison). We match the accuracy against QualNet simulation for a single bottleneck link dumbbell topology with a given flow arrival rate, flow size distribution, bottleneck link capacity and bottleneck link scheduling policy.

In prior work [30, 5], analytic models for FIFO, LAS and Run2C have been widely discussed and developed based on a particular queueing model - M/G/1, in which assumptions concerning the arrival process, service requirement distribution and the number of servers are made. In most cases, these models are used to compare the theoretical properties of size-based scheduling policies, consisting of the unfairness investigation, optimality issue, etc. Unlike prior policies, it is not straightforward to derive an analytic model for EFD discipline due to the key feature it possesses. Recall that in EFD, a flow entry remains in the table as long as there is at least one corresponding packet present in the queue, and is dropped immediately when the last packet leaves. Let us consider the example of a TCP connection in its early

infancy. Assuming that the delayed ACK is turned off, and neglecting the possible interaction with other flows and the connection set-up, the scheduler will create an entry for the first data packet, delete it upon the packet's departure from the queue, create a new entry for the second flight of 2 packets, delete it upon departure, etc. Consequently, a TCP connection (or UDP transfers) may be split over time into several fragments handled independently by the scheduler. We denote "subflows" in the remaining of this chapter those fragments which are split from a complete flow resulting from EFD's flow bookkeeping mechanism. Figure 5.1 illustrates a sequence of subflows split from the same flow. Each subflow is truncated by a preset threshold *th*, aiming at giving high priority to the beginning of each subflow so as to eventually favor short and persistent low rate flows, and avoid lock-outs without paying a too high price in terms of flow bookkeeping.

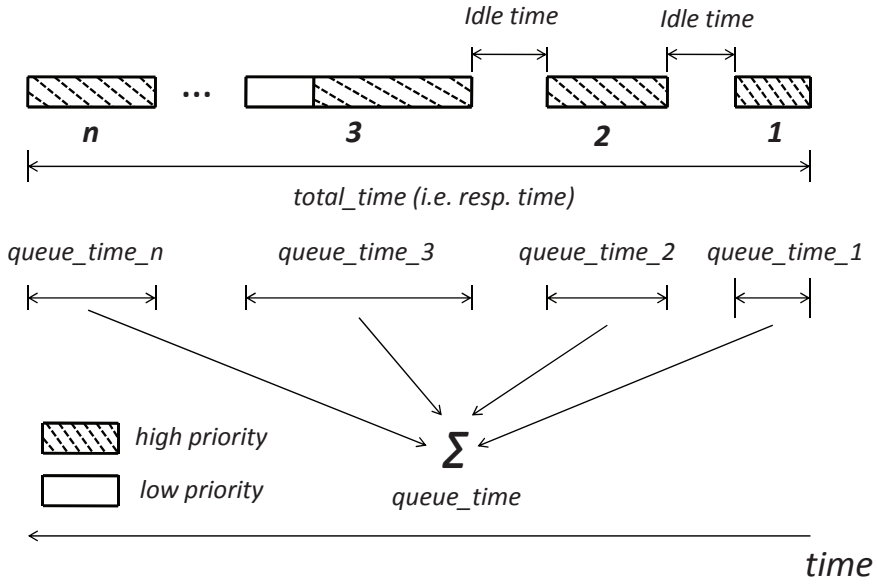
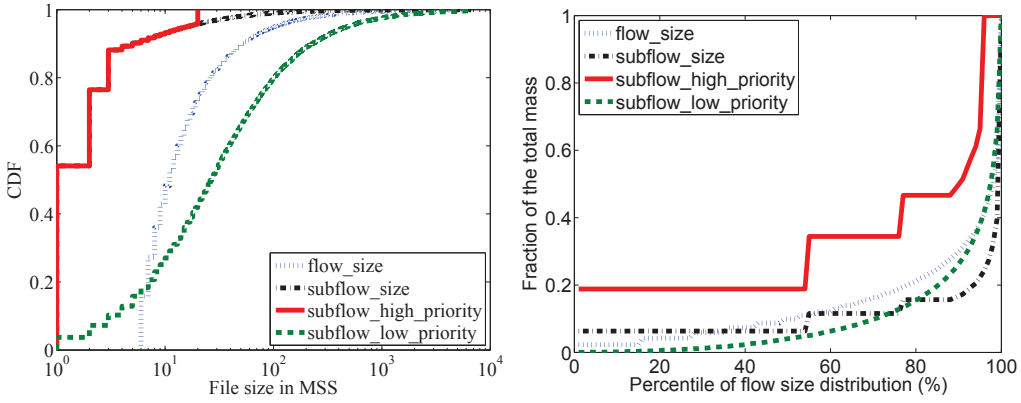


Figure 5.1: How a flow is split into subflows?

In order to investigate how a flow is split into subflows under EFD scheduling policy, we report the flow size and subflow size distribution in Figure 5.2 (Section 5.2.2 details the experimental setup). Ideally, EFD was designed to favor short and persistent low rate flows. In reality, the TCP congestion control policy restricts the packets' arrival behavior, beginning with slow start in general by sending packets in flights and recovering in the manner of timeout (RTO) or fast retransmit/recovery (FR/R) in case of loss event. Furthermore, even the packets sent in the same flight are not always observed to arrive to the scheduler at the same time since they may cross traffic over the same or different paths. We observed from Figure 5.2(a) that, flows are prone to be split into many extremely small subflows, in which subflows

with size less than or equal to 3 packets make up around 90% of subflows, given that flow size is bounded pareto distributed. In addition, subflow size distribution (black dotted line) retains the heavy tail property as flow size distribution shown in mass-weighted distribution in Figure 5.2(b), but exhibits smaller variability in contrast to the original flow size distribution (blue dotted line).



(a) CDF (b) Mass-Weighted Distribution

Figure 5.2: Flow and subflow size distribution, $\rho = 0.5$

Given the subflows obtained from flows, when applying the threshold method, each subflow is cut into two parts - one directed to the high priority queue while the other one goes to the low priority queue, if the subflow size is equal to or larger than the threshold value *th*. For simplicity, we term them as subflows without changing the name, but accompany with high or low priority queue for distinction from original subflows. We report the subflow size distribution in high and low priority queue respectively (red and green curves) in Figure 5.2, in which subflow size distribution in high priority queue is in essence the same as original subflow size distribution but truncated at threshold *th* - which is set to 20 MSS in our experiments. The subflows in low priority coming from the remaining of the original subflows by subtracting the first *th* packets, reveal a less pronounced heavy tail property.

5.2 Analytic Models for FIFO, LAS and Run2C

In this section we estimate the average response time as a function of flow size using validated analytic models with the assumption of an M/G/1 queueing model for FIFO, LAS and Run2C, letting the discussion of deriving the analytic model for EFD to latter sections. The goal is to assess the applicability of those models to a single bottleneck wired network by checking their results against QualNet simulation. We also want to use the analytic models to help understanding some of our simulation results.

5.2.1 The M/G/1 Model

We consider the classical M/G/1 model, where arrivals follow a Poisson process and service requirements are independent identically distributed following a general distribution, with a single server providing service and infinite waiting room. The overall cumulative distribution function of service requirements[1] is given by $F(x)$, and the complement of the distribution is given by $F^c(x)$. Assume that flows arrive with rate λ, and the density of service demands is denoted as $f(x)$. In addition, key measures for modeling policies include (1) the ith moments of the service requirement distribution m_i, (2) the link load $\rho = \lambda m_1$, (3) the ith moments of the service requirements distribution in which the service requirements are truncated at a given value x, which are given by

$$m_i(x) = \int_0^x u^i f(u)\mathrm{d}u + x^i F^c(x),$$

and (4) the load due to the truncated flows,

$$\rho(x) = \lambda m_1(x).$$

Below we simply list the formulas for calculating the average transfer time for flows that (a) share a common bottleneck link and (b) experience negligible contention at other points in their respective transmission paths, assuming zero propagation delay between the flow endpoints and the bottleneck. The latter(zero propagation delay) allows us to compare directly the service time in the queue and the network response time extracted from the QualNet. Note that the analytic models are developed with the assumption of infinite waiting room in M/G/1 model, meaning that the queue will never build up, in contrast to simulations where the buffer size at the AP is finite. In the results presented below, we took it equal to 300 packets.

5.2.1.1 FIFO Model

When the packets are scheduled over the link in FIFO order, the transmission of packets from different flows will be interleaved since packets of each flow arrive at variable times. Typically, a variable number of packets in a particular flow will arrive between two consecutive packets in another active flow - we term it as the "quantum" for interleaving. The quantum is usually not constant and not uniformly distributed, resulting in varying flow rate for flows with different sizes. FIFO discipline in practice is therefore quite complex to model. We consider two queueing models for modeling a real FIFO queue. With the assumption that one flow consisting of certain number of packets is fully processed before the other (quantum of zero), the formula based on Little's Law and PASTA property to calculate the average transfer time for a

[1] Note that the service requirement and the flow size can be easily converted into each other through the bottleneck link capacity. Once the bottleneck link bandwidth is given, the service requirements and the flow sizes are proportional to each other.

flow with the service requirement of x in M/G/1-FIFO model - so called Pollaczek-Kinchin formula, is given by:

$$E[T(x)]_{FIFO} = \frac{\lambda m_2}{2(1-\rho)} + x \tag{5.1}$$

The FIFO model is expected to overestimate the response time of short flows as they are heavily penalized if a large flow is processed (since all the packets of the large flow will be processed together). Conversely, one can expect that the model will underestimate the response time of the large flows.
Processor Sharing, in which flows are fairly serviced in round-robin manner, is generally believed to be a good model at flow level for FIFO discipline with TCP flows having the same RTT which is the case in our simulations. Given that the propagation delay is assumed to be zero, the delays for connection establishment and retransmitting lost packets are negligible. The average transfer time for a flow with the service requirement of x in M/G/1-PS model is given by:

$$E[T(x)]_{PS} = \frac{x}{1-\rho} \tag{5.2}$$

5.2.1.2 LAS Model

For simplicity, the analytic model for LAS is derived by assuming that each flow has at least one packet in the queue of the bottleneck link as in [53]. With this assumption, the average transfer time of a flow with size x - expressed in units of time, under the LAS policy can be modeled by the mean response time to serve a job of size x at a server in an M/G/1 queue, given by:

$$E[T(x)]_{LAS} = \frac{W_o(x) + x}{1-\rho(x)} \tag{5.3}$$

where $W_o(x)$ is the average backlog of packets but with truncated service time, given simply by the P-K mean value formula as

$$W_o(x) = \frac{\lambda m_2(x)}{2(1-\rho(x))} \tag{5.4}$$

Note that each flow does not actually have one packet in the queue at every time instant. Due to the fact that a packet will always be served before the packets with lower priority although this packet may arrive a bit latter than the time at which it should arrive in the model, this model is expected to be not perfect but good enough to match the simulation.

5.2.1.3 Run2C Model

As a threshold based scheduling policy, Run2C is essentially a two level PS model (M/G/1-PS+PS) at the flow level. To derive the model, we consider these two PS queues separately. Let *th* be the threshold value. For those flows whose size is less

than or equal to *th*, they are served in the system in a pure PS manner where the service time distribution is truncated at *th*. Therefore the mean conditional response time is given by

$$E[T(x)]_{Run2C} = \frac{x}{1-\rho(th)}, \qquad for\ x \in [0, th] \tag{5.5}$$

For flows with size $x \in (th, \infty)$, the mean response time conditional on the flow size consisting of the delay due to the time spent in the first high priority queue where the flow is serviced up to the threshold *th*, and the time spent in the low priority queue, is given by

$$E[T(x)]_{Run2C} = \frac{W_o(th) + th}{1-\rho(th)} + \frac{\alpha(x-th)}{1-\rho(th)}, \qquad for\ x \in (th, \infty) \tag{5.6}$$

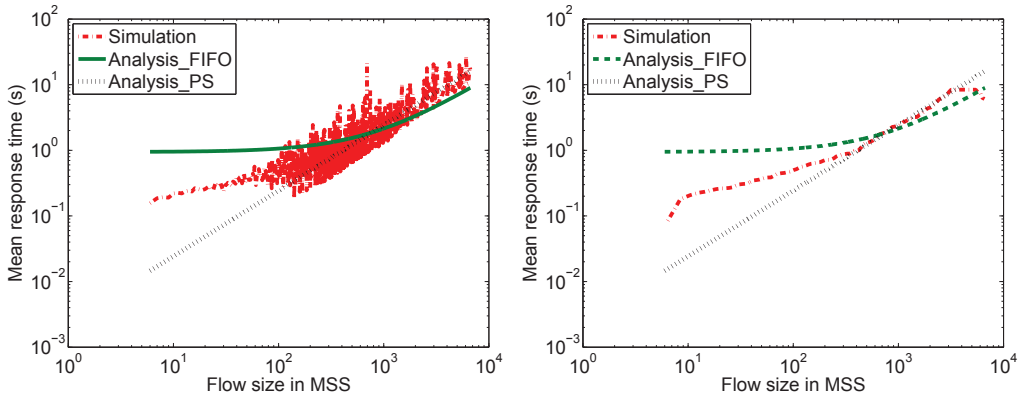
where $W_o(th)$ is given by Equation(5.4), and $\alpha(x)$ is the virtual time spent in the low priority queue. We do not discuss the term $\alpha(x)$ in detail since it is integrated in an integral equation without explicit expression[5], requiring either fixed point iterations or the finite approximation of the Riemann sum to solve the equation. For simplicity, we focus on the flows with the size less than or equal to threshold value *th* in the remaining of this chapter, and model the high priority queue only, given by Equation(5.5).
Note that given the model for Run2C, a similar model for EFD can be easily derived by replacing the flow size distribution with the subflow size distribution, obtaining the mean conditional response time of subflows, whose size is restricted to the range of values smaller than or equal to the threshold *th*. However, note that this approach will not lead directly to an analytical model for EFD at flow level since it requires knowledges of the process that maps flows to subflows. We investigate such a model in Section 5.3.2.

5.2.2 Mutual Validations

To match the simulations with the analytical results so as to help to understand the simulations, we implement the above scheduling policies in the simulator QualNet. In our experiments, a wired dumb-bell topology with single bottleneck link is deployed, using a representative buffer of 300 packets on the bottleneck link. A traffic profile with a specified flow size distribution - bounded Pareto distribution with high variability - is generated. The total load over the bottleneck link, ρ, is equal to 0.5, which is moderate and will not build up the queue, resulting in a negligible loss rate.
Simulations are run for 5000 seconds, and the data is collected after a warm up period of 100 seconds. We obtain performance indicators in terms of the mean transfer time as a function of flow size, by taking the workload generated out of a specified distribution during the simulations, as the input for the analytic model. All link propagation delays are set to be zero so as to remove the influence from the simulation results, since the analytic models do not incorporate the latency over

the paths. We compare the results obtained from simulation with the analytical results for mean response time conditionally on the flow size. If the results are in agreement, we consider them as mutually validated.

We firstly focus on the case of moderate load (*i.e.* $\rho = 0.5$), and leave the discussion of the possible difference observed under high load (*i.e.* $\rho = 0.9$) at the end of this section. Figure 5.3 presents the mean flow transfer time against flow size for the QualNet simulation with FIFO scheduling discipline deployed over the bottleneck link and the corresponding analytical models - we test both FIFO and PS flow level models to see if they help, even though the good model for FIFO at flow level normally should be PS. We depict the results in two manners - *raw* and *medfilt*. We term "*raw*" as the raw data we collected directly from the simulation and simply take the mean value among the flows with the same size, while "*medfilt*" denotes the data collected from the simulation and smoothed by one-dimensional median filtering technique with a specified bin size.



(a) FIFO - raw

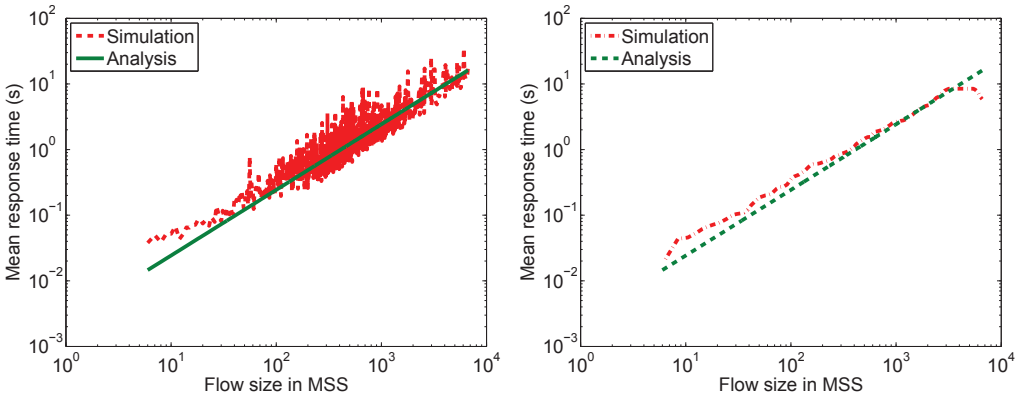
(b) FIFO - medfilt

Figure 5.3: Analysis and Simulation, $\rho = 0.5$, FIFO

The results in Figure 5.3 demonstrate that the mean flow transfer time from the simulation is in between the analytical results estimated by the FIFO model and the PS model for short and moderate flows, in which the PS model underestimates the simulation whereas the FIFO model overestimates the simulation. In addition, the PS model fits the simulation estimates quite well for flows larger than 400 packets but the FIFO model fails. One possible explanation behind the phenomenon is that, due to practical bound on simulation running time, the simulated flow size distribution does not precisely match the (bounded) Pareto distribution. To partially mitigate the discrepancy between the simulation estimate and the PS model, another possibility is to add up the connection set up time and the connection tear down time which are significant and not negligible for short flows, to the analytical results.

SCFQ is known to be an approximate implementation of Processor Sharing for packet networks. Thus it makes sense to see in practice how well SCFQ mimics the PS model. We report the simulation result for SCFQ, together with the analytic

estimate from PS model in Figure 5.4. In general, a good agreement between the two items can be clearly observed from the figure - the PS model only slightly underestimates the response time of small and medium size flows.



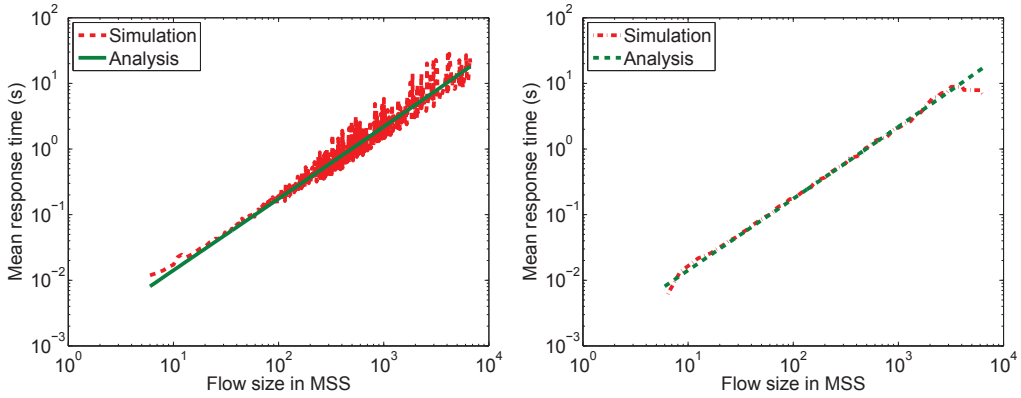
(a) SCFQ - raw (b) SCFQ - medfilt

Figure 5.4: Analysis and Simulation, $\rho = 0.5$, SCFQ

We plot the mean transfer time against flow size obtained from the analytic model as well as from the simulation for the LAS scheduling policy in Figure 5.5. We observe a nearly perfect agreement between the analytic model and the simulation under the link load $\rho = 0.5$ and low loss rate, as anticipated in Section 5.2.1.2. Note that we may see the discrepancy, especially for long flows if we increase the link load to a high value, for example $\rho = 0.9$. Most of the loss is expected to be experienced by long flows in high load under LAS since short flows are highly protected by LAS, leading that long flows having actual (simulated) mean transfer time higher than predicted by the analytic model. The reason is that, the throughput obtained with the commonly used TCP protocol is known to be inversely proportional to the square root of the loss rate. Therefore, increasing the loss rate will reduce the throughput and increase the mean transfer time. Still, the analytic model is observed to have slightly lower mean transfer time than the simulation for short flows in Figure 5.5 due to the reason that the analytic model does not incorporate the connection set up time and the tear down time, as mentioned for FIFO.

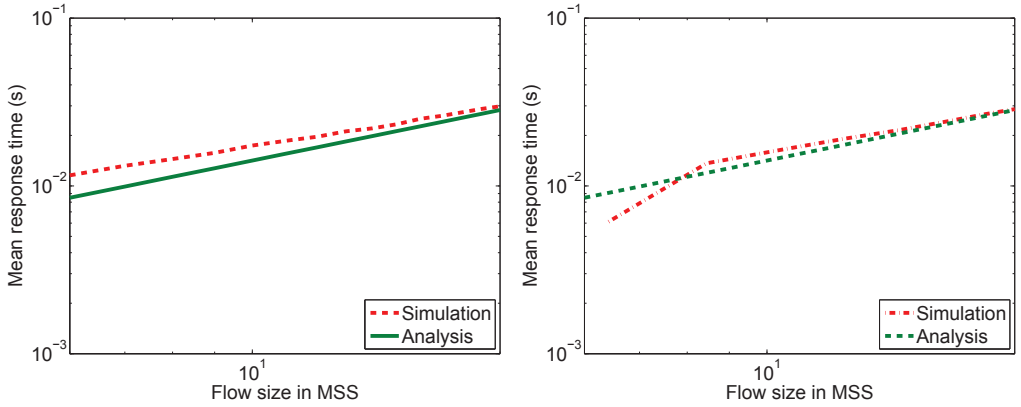
To avoid solving the integral equation, we partially take use of the analytic model for Run2C given in Section 5.2.1.3 and report in Figure 5.6 the results obtained from the analytic model as well as from the simulation for flows with size less than or equal to the threshold value *th*. We present the "*raw*" and the "*medfilt*" data in Figure 5.6.

The results in Figure 5.6 demonstrate that the analytic model is in good agreement with the simulation estimates. In addition, the results from the analytic model as well as from the simulation are observed to converge as the flow size increases. One possible explanation behind it is that, as the flow size increases, the connection set up time and the connection tear down time become less and less significant, and



(a) LAS - raw (b) LAS - medfilt

Figure 5.5: Analysis and Simulation, $\rho = 0.5$, LAS

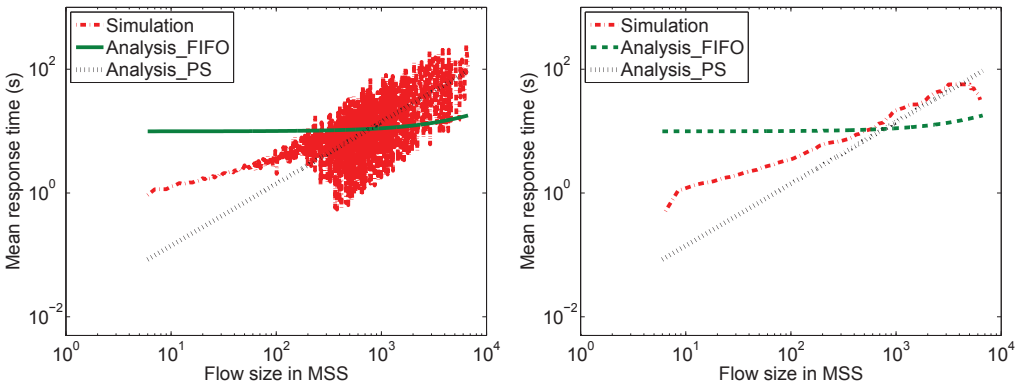


(a) Run2C - raw (b) Run2C - medfilt

Figure 5.6: Analysis and Simulation, $\rho = 0.5$, Run2C

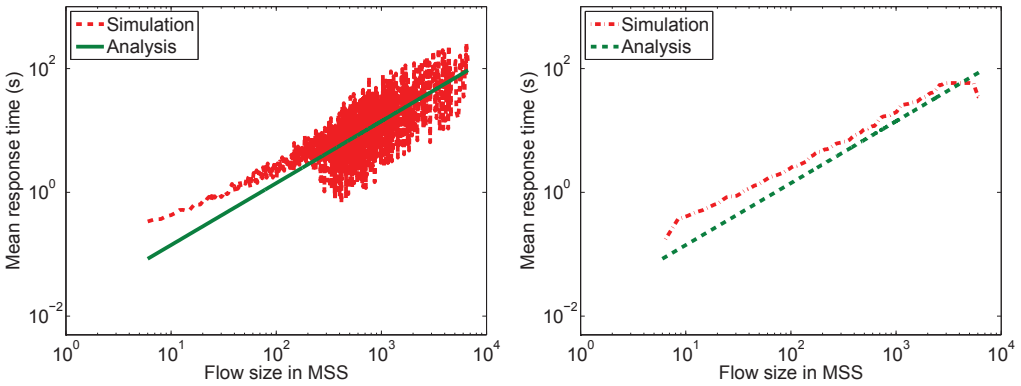
will be finally negligible as compared to the mean flow transfer time when the flow size is large enough.

We next turn our attention to the case of high load ($\rho = 0.9$), in which noticeable loss rate is expected to experience. We finally obtain qualitatively similar results as the case of moderate load ($\rho = 0.5$), except that higher variability of flow transfer time for long flows are pronouncedly observed for the scheduler of FIFO, SCFQ and LAS. Small flows normally offer low load and are highly protected by LAS, therefore loss events are mainly experienced by long flows, resulting in high variability of response time for long flows. We report all these results in the case of high load ($\rho = 0.9$) for FIFO, SCFQ, LAS and Run2C, illustrated in Figure 5.7, Figure 5.8, Figure 5.9 and Figure 5.10 respectively. In summary, the commonly used models for FIFO, SCFQ, LAS and for the high priority queue seem valid with our experimental set-up and validate our QualNet simulations.



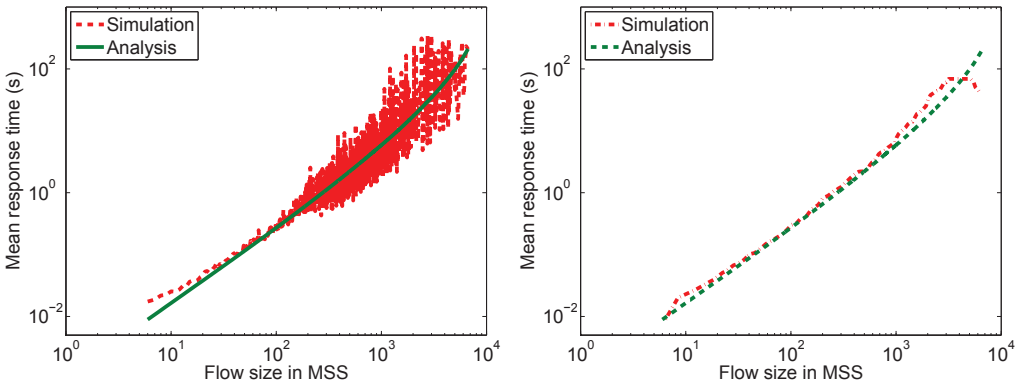
(a) FIFO - raw (b) FIFO - medfilt

Figure 5.7: Analysis and Simulation, $\rho = 0.9$, FIFO



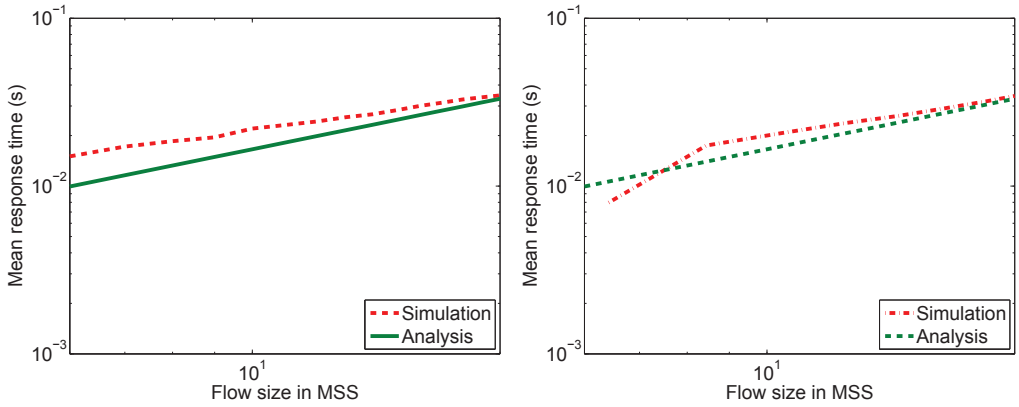
(a) SCFQ - raw (b) SCFQ - medfilt

Figure 5.8: Analysis and Simulation, $\rho = 0.9$, SCFQ



(a) LAS - raw (b) LAS - medfilt

Figure 5.9: Analysis and Simulation, $\rho = 0.9$, LAS



(a) Run2C - raw (b) Run2C - medfilt

Figure 5.10: Analysis and Simulation, $\rho = 0.9$, Run2C

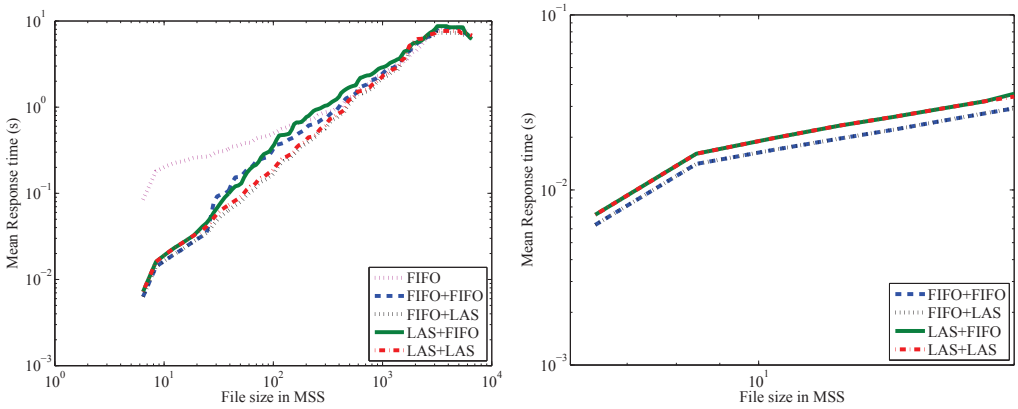
5.3 Performance of EFD and its variants: an analytical explanation

5.3.1 The Numerical Analysis: first attempt at subflow level

The EFD policy was initially designed for scheduling at flow level and investigated for single direction transfers in wired network (see Chapter 4), accounting for volumes in bytes. An alternative is to count volume in terms of number of packets, which may bring difference when deployed for co-existing bidirectional transfers, as data packets are generally MSS packets at transport level while ACKs are 40 bytes packets (see Chapter 6). For simplicity, we discuss only the byte-based scenario for EFD and its variants in this chapter, meaning that the volumes are always measured in bytes. Initially, the EFD scheduling is FIFO+FIFO scheme since packets within each (virtual) queue are drained using the FIFO discipline at packet level. We extend EFD by discussing the impact of alternative scheduling disciplines in the EFD scheme. In particular, we consider two candidates, FIFO and LAS, which leads to four combinations: FIFO+FIFO, LAS+FIFO, FIFO+LAS, LAS+LAS.
We investigate the performance of the aforementioned EFD's variants - using the same experimental setup detailed in Section 5.2.2 - in terms of the average conditional response time as a function of flow size. To simplify the discussion and the analysis, in the remaining of this chapter, we consider a moderate load level (*i.e.* $\rho = 0.5$). Figure 5.11(a) reports the result collected from the simulation for all flow sizes, while Figure 5.11(b) highlights the result for flow sizes less than or equal to the threshold *th*.
We observe from Figure 5.11(a) that FIFO+XX[2] performs slightly better than LAS+XX for short flows - with size less than 30 packets[3], whose packets are gener-

[2]We use FIFO+XX to denote the two possibilities: FIFO+FIFO and FIFO+LAS, and similar meanings are given for LAS+XX, XX+LAS and XX+FIFO.

[3]Note that the value "30" is a bit larger than the threshold *th* - which is set to 20 packets in our



(a) all flow sizes (b) flow sizes $\leq th$

Figure 5.11: Mean response time in simulation - EFD's Variants

ally directed to the high priority queue, implying that replacing FIFO with LAS in the high priority queue is detrimental. We believe the reason behind this observation is that, LAS features bad performance when the distribution has a low variability - see [30]. This is the case in the high priority queue perspective here, since the subflow sizes in this queue range between 1 and 20 MSS only, and the distribution is much less skewed (CoV close to 1) than the overall flow size distribution (CoV of 6), shown in Figure 5.2(a) and Figure 5.12(a) to be presented later. Let us consider the extreme case: a distribution with a CoV of 0, *i.e.* a deterministic distribution. For the scheduling policy LAS, the conditional response time are all the same, given by:

$$E[T(x)]_{LAS} = \frac{\lambda x^2}{2(1-\rho)^2} + \frac{x}{1-\rho} \tag{5.7}$$

whereas $E[T(x)]_{FIFO}$ and $E[T(x)]_{PS}$ for a deterministic distribution are given by:

$$E[T(x)]_{FIFO} = \frac{\lambda x^2}{2(1-\rho)} + x \tag{5.8}$$

and

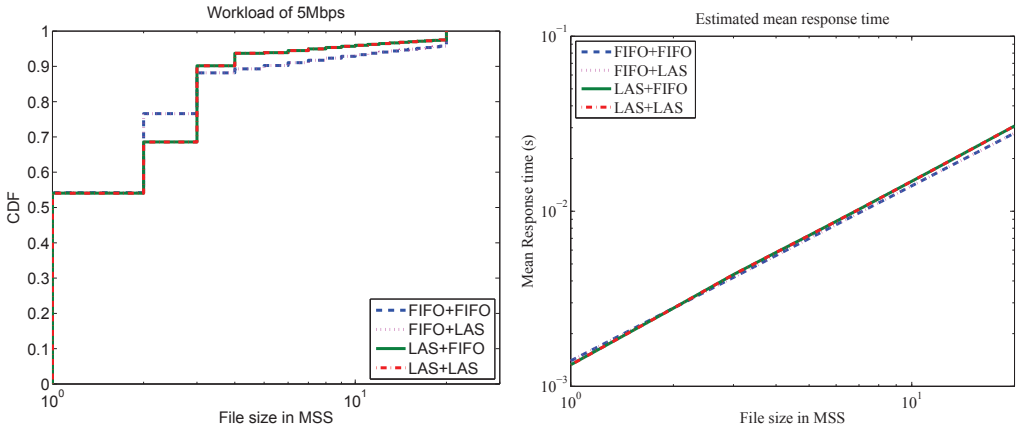
$$E[T(x)]_{PS} = \frac{x}{1-\rho} \tag{5.9}$$

It turns out that $E[T(x)]_{LAS} = \frac{E[T(x)]_{FIFO}}{1-\rho} \geq E[T(x)]_{FIFO}$ and $E[T(x)]_{PS} \leq E[T(x)]_{LAS}$. Hence, LAS offers larger response time, as compared to FIFO and PS for a deterministic service distribution.

experiments, meaning that even flows with size slightly larger than *th* benefit when scheduled, due to the fact that the EFD-like scheduler is prone to split a flow into groups of subflows and handles each subflow separately.

Furthermore, XX+LAS is observed to outperform XX+FIFO for medium and intermediate size flows in Figure 5.11(a). It is understandable since the majority of packets of each medium and long flow are scheduled in low priority queue by EFD-like scheduler and the subflow size distribution in low priority queue still exhibits relatively high variability as shown in Figure 5.2. Thus LAS deployed in low priority queue obtains lower conditional response time for medium and intermediate size flows as compared to FIFO.

In order to explain the performance discrepancy among EFD's variants illustrated in Figure 5.11 from the analytical point of view, a straightforward method is to estimate the performance metric from the analytic model developed for each discipline used in high priority queue - M/G/1/PS given by equation (5.2) for FIFO+XX and M/G/1/LAS by equation (5.4) for LAS+XX respectively[4] - by taking the subflow size distribution in high priority queue tracked during the simulation as input for each analytical model. Since the subflow size in high priority is limited to the maximum value *th*, we simply report the subflow size distribution and the mean response time over the subflow sizes in high priority queue in Figure 5.12, in which the subflow sizes are counted by including data packets only and not acknowledgments. We expect to observe some kind of similarity concerning the performance metric obtained from the M/G/1 model for subflows in high priority queue and the one obtained from the simulation for flows, although flows and subflows are not exactly equivalent in our context. To put it differently, we believe that the performance of subflows can in some sense reflect one of the original flow, and help to understand the different behavior of scheduling disciplines.



(a) subflow sizes in high priority queue

(b) the analytical results ($\leq th$)

Figure 5.12: Analytic model - EFD's Variants

This is unfortunately not the case as shown in Figure 5.12(b). Since the analyt-

[4]Note that we use the M/G/1 model here, simply assuming that the arrivals of the subflows in high priority queue still follows Poisson process after split from the original flows. We defer the discussion of the arrival process to later part.

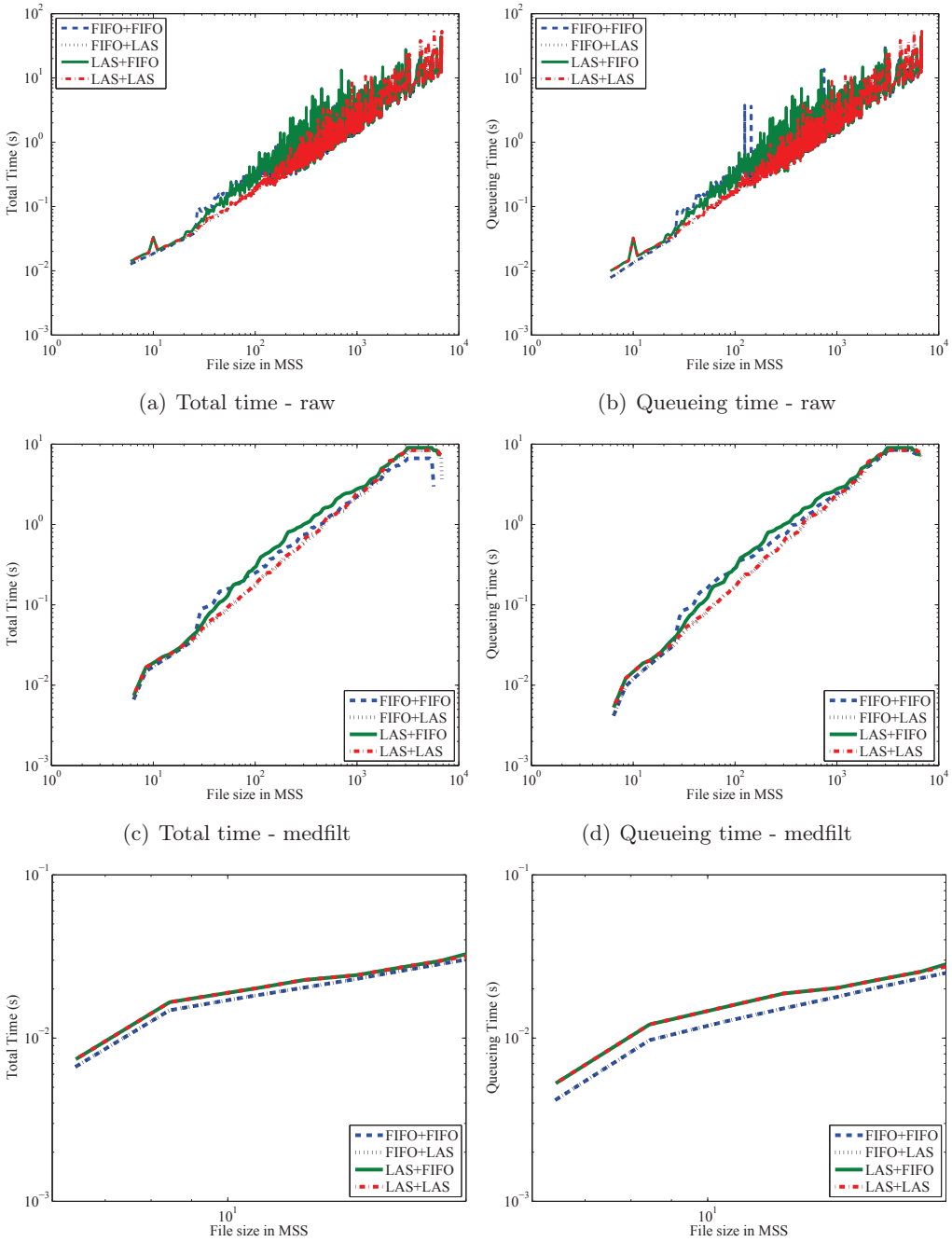
ical model is capturing the queueing time, and the results in Figure 5.11(b) relate to the flow response time, we decided to dig into the simulation results to extract metrics closer to the actual queueing time experienced by flows. We introduce two new terms here for the following presentation. Based on the splitting phenomenon illustrated in Section 5.1 and the diagram depicted in Figure 5.1, we define the total time of a flow as the time between the first packet of the first subflow entering the queue until the last packet of the last subflow leaving the queue, including the possible idle times between every two adjacent subflows. The queueing time of a subflow is defined as the time a subflow resides in the queue, denoted as $queue\_time\_1, queue\_time\_2, ..., queue\_time\_n$ respectively for subflow1, subflow2, ..., subflow$n$, shown in Figure 5.1. We then obtain the queueing time of a complete flow by summing up the queueing time of all its subflows, which differs from the total time by excluding the idle times, demonstrated in Figure 5.1. We use the queueing time to approximate the response time by considering the fact that, the queueing time makes up a significant fraction of the response time[51]. We report the "*raw*" and "*medfilt*" results for total time and queueing time in Figure 5.13.

We observe that qualitatively the queueing time of the flows are upper bounded by the total time, which is in line with our expectation. Moreover, the gap between total time and queueing time is significant for small flows, but negligible for medium size and long flows. This is confirmed by the scatter plot and the corresponding time ratio over flow sizes, respectively given in Figure 5.14(a) and (b). One possible explanation behind it is that, the small flows are more sensitive to the idle times since their response time are generally small because of countable amount of packets to be transfered.

As a partial conclusion, it seems that the discrepancy between the variants of EFD in the high priority queue is a phenomenon that relates to what is happening in the queue, even through the analytical models do not exhibit them clearly (see Figure 5.12(b)).

A last point concerning the discrepancy between simulation that operate at flow level and queueing models that operate at subflow level is the inter-arrival process of subflows in high priority queue. We justify that it is not Poisson process anymore, although the original flows arrive in a Poisson process manner. We test the hypothesis by plotting the inter-arrival time of subflows against the corresponding exponential distribution with the same mean value, and verifying through their QQ-plot, shown in Figure 5.15(a) and (b). Interestingly, LogNormal and Weibull fit the distribution of the inter-arrival time of subflows quite well, given that the same mean value is guaranteed under the test. We report the results for LogNormal only in Figure 5.15(c) and (d).

In summary, in this part, the analytical models for FIFO and LAS did not allow us to confirm what we observed through simulations. We confirmed that the discrepancy observed between LAS+XX and FIFO+XX in our simulations is due to the queueing phenomenon by removing the idle time between the subflows from the original response time. The analytical result with a CoV of zero was maybe



(a) Total time - raw

(b) Queueing time - raw

(c) Total time - medfilt

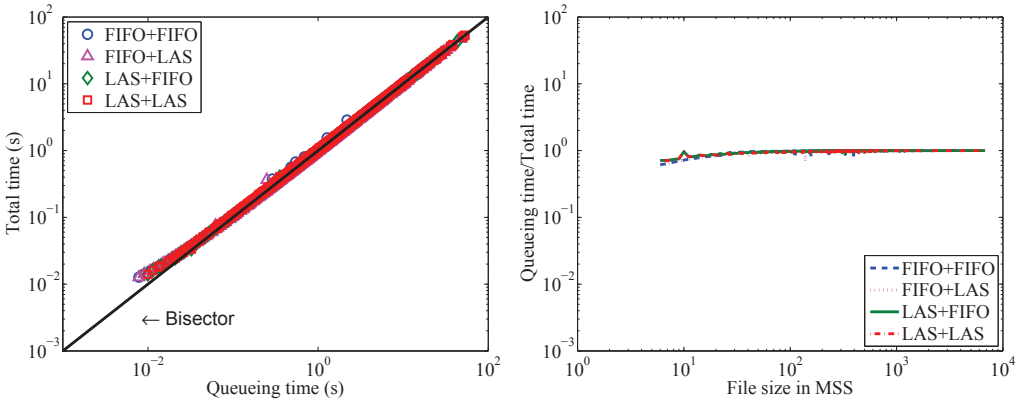
(d) Queueing time - medfilt

(e) Total time($\leq th$)

(f) Queueing time($\leq th$)

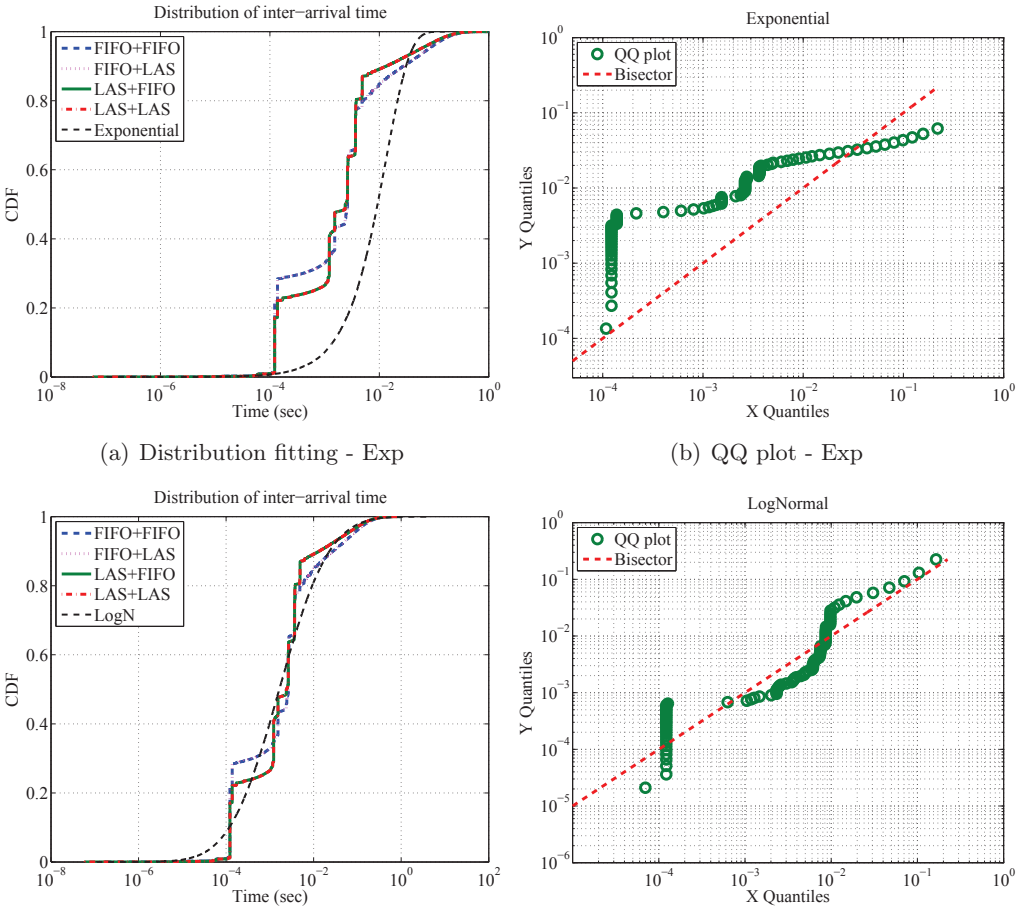
Figure 5.13: Total time and Queueing time - EFD's Variants

misleading as the CoV in the high priority queue is in fact close to 1 in our case. We leave this problem open for the moment and switch to the problem of relating subflow level performance to flow level performance in EFD.



(a) Scatter plot for queueing and total time (b) Queueing time/Total time

Figure 5.14: Time comparison - EFD's Variants



(a) Distribution fitting - Exp (b) QQ plot - Exp

(c) Distribution fitting - LogN (d) QQ plot - LogN

Figure 5.15: Hypothesis testing - EFD's Variants

5.3.2 A Model for Shifting from Subflow to Flow

To explain the simulation results which are obtained at flow level, in previous section we attempted at subflow level as a starting point but fail. There are likely two reasons for that: on one hand, our analysis on subflows is based on the assumption that the CoV of subflow sizes in high priority queue is zero, but it is close to 1 in fact; on the other hand, the inter-arrival process of subflows in high priority queue is no longer Poisson process while the existing analytic models of the scheduling discipline used in our analysis is M/G/1 model-based. In this section, we endeavor to build a model to move from subflow to flow.

5.3.2.1 A TCP-rule based Model

In order to roughly emulate how flows are split into subflows by EFD scheduler, we propose a straightforward but simple model. With this model, we only focus on TCP connections, in which NewReno is deployed with delayed ACK enabled. B. Sikdar *et al.* [57] reported the congestion window (cwnd) increase pattern when delayed ACK is enabled as:

$$cwnd_n = \lfloor 2^{(n-1)/2} + 2^{(n-2)/2} \rfloor \tag{5.10}$$

which produces the sequence of window size:

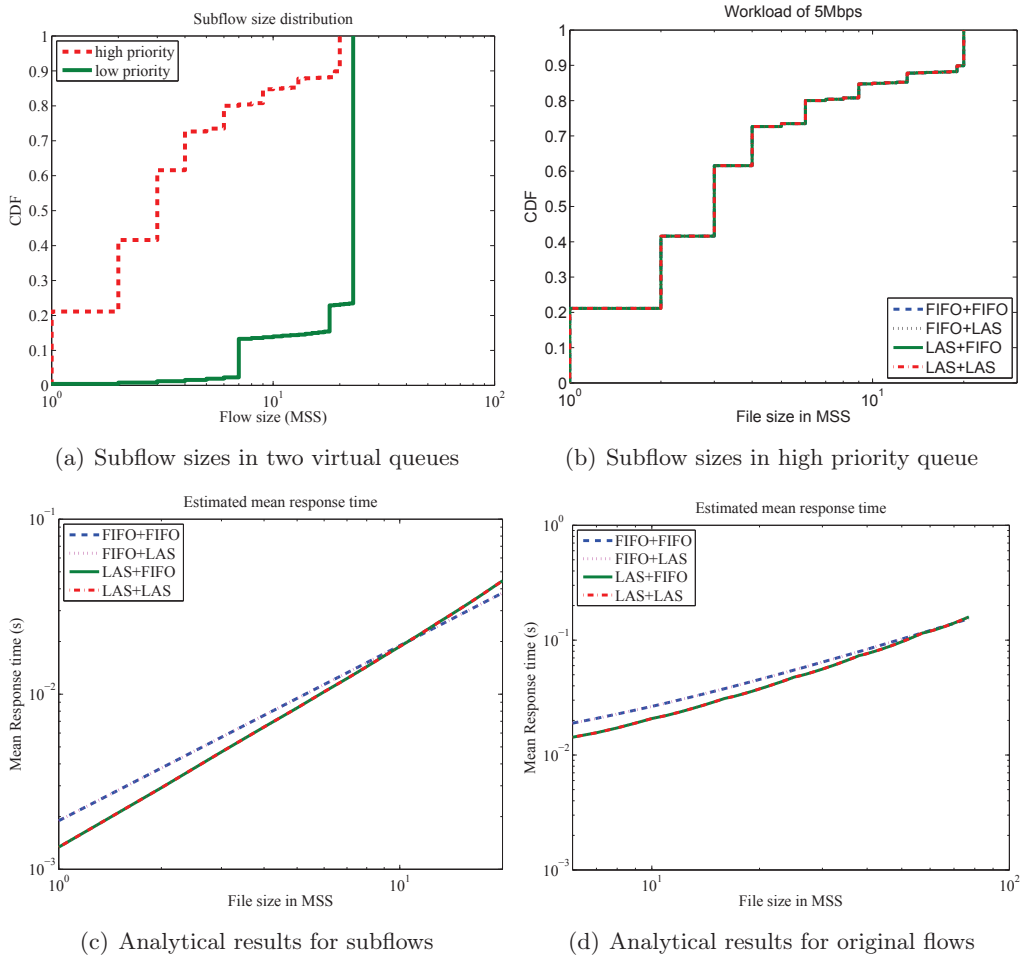
$$1, 2, 3, 4, 6, 9, 13, 19, 27, 38, 54, ...$$

Suppose that the maximum congestion window is set to 64.5KB, which is equivalent to 43MSS. Then the sequence will be capped to

$$1, 2, 3, 4, 6, 9, 13, 19, 27, 38, 43, 43, ...$$

Under this logic, a TCP connection is transfered as a sequence of flights, whose sizes in packets[5] are limited by the TCP delay ACK mechanism and the maximum congestion window. By taking into account the effect of the threshold in EFD-like scheduling policies, each flight consisting of a certain number of packets will be either fully given high priority if the volume is below the threshold, or be cut in case its volume exceeds the threshold. The subflow size distributions in the two virtual queues (high and low priority queues) under this model for FIFO+FIFO (the same as other variants) are given in Figure 5.16(a).
We estimate the conditional mean response time for subflows by the analytic models presented in Section 5.2.1 - M/G/1/PS given by Equation (5.2) for FIFO+XX and M/G/1/LAS by Equation (5.4) for LAS+XX respectively, assuming that the arrival process of the subflows in the high priority queue is still a Poisson process. However, given the performance estimate of subflows from analytic model in Figure 5.16(c),

[5]Note that we count the volumes in bytes, assuming that each data packet has a fixed size of MSS.



(a) Subflow sizes in two virtual queues

(b) Subflow sizes in high priority queue

(c) Analytical results for subflows

(d) Analytical results for original flows

Figure 5.16: A Performance Model for EFD

we are not able to compare it to the results at flow level obtained from simulation illustrated in Figure 5.11(b).

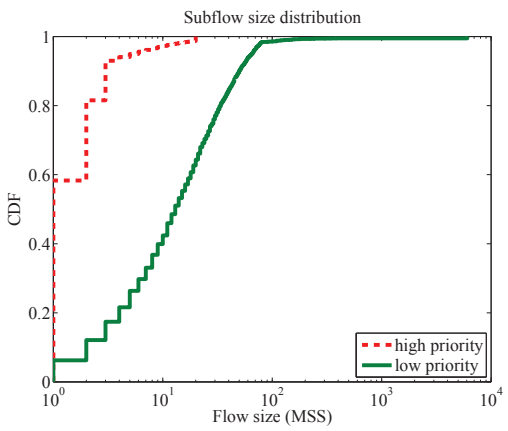
There exists a simple approach to shift from subflow to flow based on the congestion window growth presented before. To explain this method, let us take an example : a flow with 3 data packets. The break down gives 1+2. Hence, the following packets will go in the queue: one SYN, one ACK, a single data packet and then a group of two data packets, one FIN, and one ACK. Let Tsub(1) and Tsub(2) be the response time of the subflows of size 1 and 2 respectively in the queue. The response time of the flow in the queue is: T(3)=5Tsub(1)+1Tsub(2). One can add the latencies on each part of the path to assess the total response time. It is easy with (full-duplex) Ethernet (wired) links as the return path is non congested in our scenarios. Note that there is no need to add these latencies in our case as the propagation delay of each link is set to be zero in our simulations. At the end, we obtain an estimation of the response time of the flows originally generated whose packets only reside and

are served in high priority queue, given in Figure 5.16(d).
By comparing the results of flows in Figure 5.11(b) and Figure 5.16(d), we found that, this model does not fit the simulation results quite well. The root of the problem seems to be the distribution of subflows produced by the model. Indeed, as shown in Figure 5.2(a) and Figure 5.12(a), while this model produces small subflows in high priority, those small subflows do not account for an as large portion as the one observed from the simulation. More efforts need thus, in our opinion, to be devoted to the design of a flow-to-subflow matching model. We propose an alternative approach in the next section.

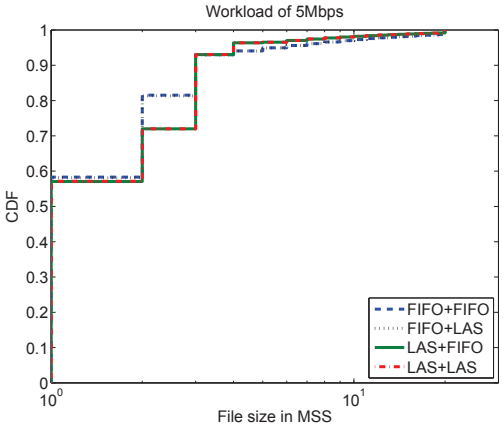
5.3.2.2 An Alternative Model

We restrict our objective to analytical verification of the performance discrepancy among EFD's variants for the flow sizes less than or equal to the threshold *th*. As an alternative, we produce the subflows in a probabilistic way by directly taking the subflow size distribution obtained from the simulation - similar to the one for EFD shown in Figure 5.2. Note that for EFD's variants, the subflow size distribution might be slightly different. Given the subflow size distribution, we are able to extract the probability for each subflow with specific size. So that for each incoming flow, we split it into subflows whose size is determined each time by taking a probabilistic test - the test is conducted by generating an independent random number in between 0 and 1, and comparing it to the accumulated probability to finally fix the subflow size. With this approach, we expect to produce a fairly large fraction of subflows with only few packets.
Since the distribution of subflow size is quite similar for EFD's variants, we report only the one in two priority queues for FIFO+FIFO in Figure 5.17(a). In addition, we illustrate the subflow size distribution in high priority queue for all four variants in Figure 5.17(b). As expected, we observe from the two figures that subflows produced with this probabilistic approach has a distribution which is very close to the one from the simulation (see Figure 5.2 and Figure 5.12(a)). Applying the same method we use in Section 5.3.2 to calculate the conditional mean response time for subflows by the analytic models and the same method to shift from subflows to flows, we demonstrate the performance of EFD's variants against the subflow and flow size in Figure 5.17(c) and (d) respectively. Note that small flows with size less than or equal to *th* are no doubt split into subflows with size less than or equal to *th*. It is therefore reasonable to construct these flows from subflows whose size is limited to *th* - the response time of these subflows are calculated from the analytic models, and finally used for the estimation of the flows.
Qualitatively in line with the simulation results shown in Figure 5.11(b), we do observe the discrepancy from Figure 5.17(d) among EFD's variants. Precisely, LAS deployed in high priority queue offers larger response time, as compared to FIFO. Therefore, LAS is believed to be detrimental when the service requirement has fairly low variability. The extreme case in which the service requirement is deterministic, has been theoretically analyzed and the similar conclusion has been made in Section

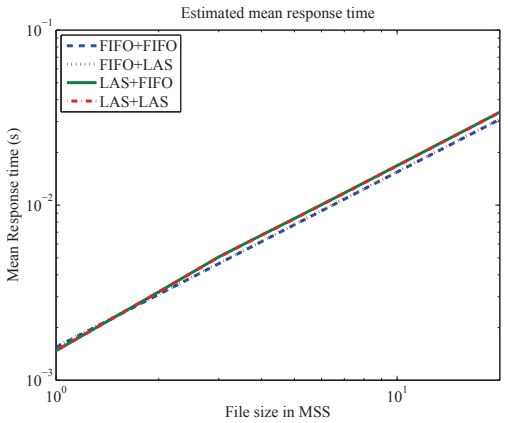
5.3.1.
In particular, the result in Figure 5.17(e) show that, if we don't incorporate the response time of the control packets (one SYN, one ACK, one FIN, and one ACK) to the calculation of response time for the flows when shifting from subflow to flow, we observe more pronounced discrepancy among EFD's variants, which is highly close to the one obtained from the simulation shown in Figure 5.11(b). The accurate result should be in between Figure 5.17(d) and Figure 5.17(e) with the explanation that Figure 5.17(e) is believed to underestimate the conditional response time against flows by not counting the latency introduced by the acknowledgments, whereas Figure 5.17(d) is likely to overestimate the results since the time to serve an acknowledgment and a regular data packets are quite different.



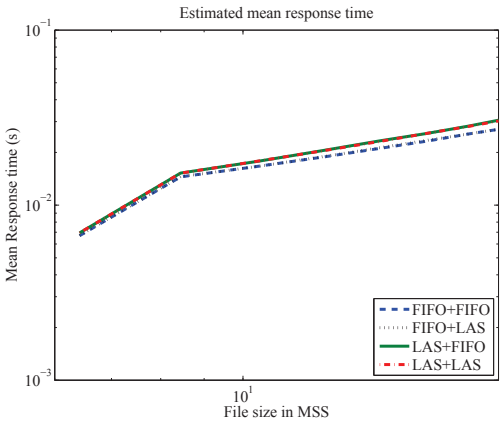
(a) EFD (FIFO+FIFO)



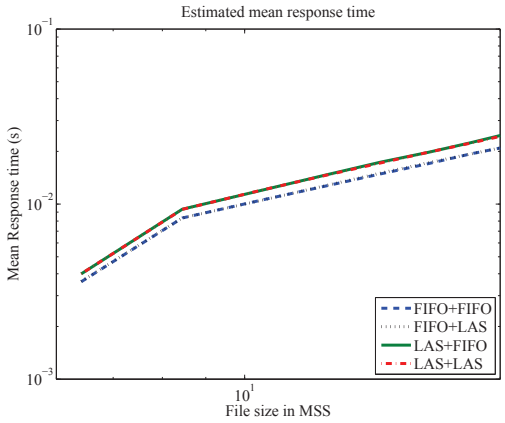
(b) Subflow sizes in high priority queue



(c) Analytical results for subflows



(d) Analytical results for original flows



(e) Analytical results for original flows-ACKs excluded

Figure 5.17: An Alternative for Modeling EFD's variants

Part II

Resource Allocation in Wireless LAN

Chapter 6

Analysis of Early Flow Discard (EFD) discipline in 802.11 WLAN

6.1 Introduction

We consider the typical infrastructure-based WLAN where mobile stations equipped with 802.11 interface communicate with an Access Point (AP) on a wireless channel and the AP relays traffic to and from the wired network. In many cases, the wireless LAN is the performance bottleneck, *e.g.* companies or labs frequently use access links to the Internet with 100 Mbit/s or higher capacity.
The TCP transport protocol is used for controlling the vast majority of data transfers in volume (bytes sent) and the majority of flows. When TCP traffic is relayed over an 802.11 network, a key performance problem, known as "TCP Unfairness", occurs. It happens when the downloads data packets, from the wired network, and TCP level acknowledgments from the uploads compete to access the access point downlink buffer. The buffer at the access point tends to fill up because the Distributed Coordination Function (DCF) at the MAC layer does not grant enough priority to the Access Point as compared to the other stations in the cell [46]. Several solutions have been investigated at various levels of the protocol stack (MAC, IP, Transport) to address the TCP unfairness problem [7, 37, 27, 58] .
In Chapter 4, EFD is investigated in wired network and using some pretty large buffer of 300 packets. In this chapter, we investigate the performance of EFD (Early Flow Discard) policy in 802.11 networks, where buffer sizes tend to be smaller as they typically range between 30 and 100 packets.
Our contributions are as follows:

- We propose two adaptations of EFD in WLAN networks, EFDACK and PEFD, that aim at mitigating the TCP unfairness problem. EFDACK keeps track of the amount of bytes sent by each flow in both the upload and download directions, which requires reading TCP segments (the acknowledgment number field) within IP packets. This is the same idea as the one of LASACK [58]. In contrast, PEFD keeps track of the number of packets and does not distinguish between uploads and downloads.

- We compare EFDACK and PEFD to state-of-the-art size scheduling policies, Run2C, LASACK, LARS and also FIFO and SCFQ.

- We demonstrate that the two modifications of EFD either outperform other

scheduling policies or perform similarly but with a lower overhead in terms of flow bookkeeping[1].

- We demonstrate that PEFD, which requires no inspection of TCP packets achieves similarly to EFDACK, except when the buffer size becomes too small.
- We extend the original design of EFD by considering alternative scheduling policies for the low and high priority queues and discuss their impact.

The remainder of this chapter is organized as follows. We introduce new variants of EFD to be analyzed in an 802.11 context in Section 6.2. In Section 6.3, we detail our evaluation methodology. Sections 6.4 and 6.5 present the evaluation results of the various scheduling disciplines. Section 6.6 concludes the chapter.

6.2 Scheduling disciplines

The original work on EFD (see Chapter 4) considered the applicability of EFD in wired networks. In the present chapter, our focus is on 802.11 networks, which feature two key properties that lead to the TCP performance problem: (i) the protocol is half-duplex, meaning that uploads and downloads share the wireless medium and (ii) the Access Point is not granted a high enough priority to access the medium under DCF, which means that its queue, which is typically 30 to 100 packets, tends to build up.
EFD was designed with quite large buffers of typically 300 packets in mind, which is not unusual for routers. In a wireless context, 300 packets seems like a big buffer, although high speed access points (802.11n) typically store hundreds of packets when a station temporarily leaves the network to scan for other access points. When this temporary buffer is cleared (once the station comes back) the AP reverts to its normal operational mode where it typically uses a buffer (shared by all stations) that is always smaller. Hence, we explore how reducing the buffer size impacts EFD's behavior.

6.2.1 Adapting EFD to half-duplex links

The original EFD policy accounts for volumes in bytes. An alternative is to count volumes in terms of number of packets. In the remainder of the paper, we refer to these two EFD flavors as BEFD (Byte-based EFD) and PEFD (Packet-based EFD) respectively. To illustrate the difference between these two options, consider the case of a WLAN with a single upload and a single download. At the buffer of the AP, one observes, in the downstream direction, the data packet stream from the download and the ACK packet stream from the upload. As data packets are generally MSS packets while ACKs are 40 bytes packets, one clearly sees that counting

[1]The benefit of EFD concerning the overhead has been clearly justified in Chapter 4. To avoid redundancy, we don't discuss the memory consumption in this paper as the two modifications of EFD naturally inherit this good property from EFD.

volumes in bytes or packets will significantly impact the priority granted to the ACK stream: when counting in bytes, its priority will consistently be maximum whereas the competition between the upload and download will be more fair when counting in packets.
In addition to BEFD and PEFD, we introduce a variant of EFD that accounts for the half-duplex nature of MAC layer protocol. It attributes a virtual service size to TCP ACK packet by accounting for the total amount of data traffic that has been transferred by the flow so far, obtained through the TCP acknowledgment number in the TCP header. We call EFDACK this scheduling policy. Considering the same example as above of a WLAN cell with a single upload and a single download, and assuming that the flows are continuously tracked by the scheduler, the priority of an ACK packet is related to the total amount of bytes sent by the upload. We compare EFDACK, BEFD and PEFD extensively in Sections 6.4 and 6.5. Although EFDACK uses TCP level information, it can also handle UDP streams. The advantage of TCP here is that it allows the scheduler to infer what was sent in the other direction unlike UDP. This means that EFDACK treats UDP flows that would be full duplex (e.g., VoIP transfers) as simplex flows, *i.e.* it accounts only for a single direction of transfer.
Essentially, the original EFD and its adaptation for 802.11 network - EFDACK, are FIFO+FIFO schemes since packets within each (virtual) queue are drained using the FIFO discipline at packet level. We also investigate in this chapter the impact of alternative scheduling disciplines deployed to high and low priority queues. In particular, we consider two candidates, FIFO and LAS, which leads to four combinations: FIFO+FIFO, LAS+FIFO, FIFO+LAS, LAS+LAS. We explore the relative merits of these flavors of EFD in Section 6.5.1.
A last point to mention is that each of the scheduling policies that we consider is paired with a buffer management scheme. For FIFO or SCFQ (an implementation of Processor Sharing for packet networks [19]), this is drop tail. In contrast, for the size-based scheduling policies, when the queue is full, the newly arriving packet is assigned a priority according the scheduling policy and this is the packet with the smallest priority that is discarded.

6.3 Evaluation Methodology

In this section, we provide a high level overview of the evaluation methodology we apply to compare the variants of EFD that we introduced in the previous section to state-of-the-art scheduling policies.

6.3.1 Network Configuration

In this chapter, we consider a simple network configuration with 10 wired hosts and 10 wireless stations associated to a single access point, as depicted in Figure 6.1. We use the 802.11a protocol with nominal bit rate of 54Mbit/s, with RTS/CTS disabled. Good and fair radio transmission conditions are guaranteed as the 10

wireless stations are at the same physical distance from the access point and in line of sight of each other. The 10 wired hosts are connected to a router with an output rate 10 times larger than its input rate, so that its output queue never builds up. With such a configuration, the bottleneck is the access point. We use QualNet 4.5 to obtain all simulation results. TCP NewReno is used with delayed ACK enabled in the simulations.

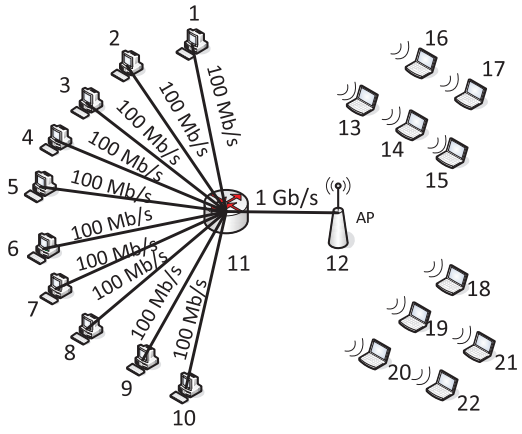


Figure 6.1: Network Set-up, with one way delay of 2ms in wired part

6.3.2 Workload

We consider essentially two workloads. First, we use only long-lived flows: while unrealistic, results obtained under such a workload enable to pinpoint easily some fundamental characteristics of a scheduling policy, due to the relative simplicity of the scenario.

Second, we consider a more realistic case of a mix of short and long flows. We generate the workload with the assumption that TCP connections arrive according to a Poisson process with rate λ and adjust λ so as to obtain two regimes: a medium load of 10 Mbit/s and a high load of 20 Mbit/s. These loads have to be considered relatively to the maximum throughput of a single TCP transfer over 802.11a at 54 Mbit/s, which is 27.3 Mbit/s [16]. The workload consists of bulk TCP transfers of varying size, generated from a bounded Zipf distribution with an average size of about 60 Kbytes (40 packets with size of 1500 bytes each), which is in line with flow sizes observed on typical campus WLANs [39]. The minimum transfer size is 6 MSS, and the maximum transfer volume corresponds to 10 MB with a coefficient of variation[2] of 6, which controls how the mass of the distribution is split between short and long transfers. Note that bounded Zipf is a discrete equivalent of a continuous (bounded) Pareto distribution, and Pareto is a long tailed distribution

[2]The CoV is defined as the ratio of the standard deviation to the mean of a distribution. The larger it is, the more skewed the distribution.

usually adopted for modeling flows in the Internet. Each packet has a fixed size of 1500 bytes in our simulations.

A last important parameter of the workload, in a 802.11 scenario where the medium is managed in a half-duplex manner, is the ratio of download to upload traffic. We denote by λ_d and λ_u the arrival rate of TCP downloads and uploads respectively. We considered initially three scenarios: $\frac{\lambda d}{\lambda u}$=1 for symmetric load, $\frac{\lambda d}{\lambda u}$=10 and $\frac{\lambda d}{\lambda u}$=100 for two asymmetric loads respectively. Those three scenarios are related to real use cases. The case $\frac{\lambda d}{\lambda u}$=10 corresponds to a typical residential user browsing the Web with no heavy P2P nor HTTP streaming (YouTube, DailyMotion, *etc.*) activity [45]. Clients that rely heavily on P2P tend to produce more symmetric ratios, corresponding to $\frac{\lambda d}{\lambda u}$=1. On the other side of the spectrum, a trend in residential network is to see more and more heavy hitters characterized by a heavy HTTP streaming activity [45]. In such a scenario, almost all bytes flow from the server to the client, leading to ratios close to 100.

To gain insights about the typical traffic within an enterprise network, we captured one full day of traffic within the Eurecom network, which comprises about 600 machines and 60 servers. We analyzed the ratio of download to upload traffic for intranet traffic and Internet traffic of each host and found that Internet traffic corresponds to an average ratio of 10, as users mostly browse the Internet, without heavy HTTP streaming activity. In contrast, intranet traffic (SMB, LDAP, etc.) is larger in volume and highly symmetric, *i.e.* characterized by ratio close to 1. A reason why the ratio of the latter is symmetric is that p2p traffic is banned from the network, as from most enterprise networks in general.

In Section 6.5, we consider the cases $\frac{\lambda d}{\lambda u}$=1 for symmetric load, and $\frac{\lambda d}{\lambda u}$=10 for asymmetric load as the case $\frac{\lambda d}{\lambda u}$=100 is less frequent in enterprise networks and degenerates to the pure download case, where the TCP unfairness problem typically vanishes. We sum up the simulation parameters in Table 6.1.

Table 6.1: Simulation Parameters

<table>
<tr><td colspan="2">Simulator</td><td colspan="3">QualNet 4.5</td></tr>
<tr><td colspan="2">MAC protocol</td><td colspan="3">802.11a@54Mbit/s</td></tr>
<tr><td rowspan="7">Workload</td><td rowspan="2">long-lived cnxs</td><td>buffer size</td><td colspan="2">10-70 MSS</td></tr>
<tr><td>composition</td><td colspan="2">5 uploads vs. 5 downloads</td></tr>
<tr><td rowspan="5">mixed workload</td><td>buffer size</td><td colspan="2">30MSS / 300 MSS</td></tr>
<tr><td>transfer size distr.</td><td colspan="2">bounded Zipf</td></tr>
<tr><td rowspan="2">load regimes</td><td>medium</td><td>10 Mbit/s</td></tr>
<tr><td>high</td><td>20 Mbit/s</td></tr>
<tr><td rowspan="2">traffic ratio</td><td>sym.</td><td>$\lambda_d/\lambda_u = 1$</td></tr>
<tr><td>asym.</td><td>$\lambda_d/\lambda_u = 10$</td></tr>
</table>

6.3.3 Performance Metrics

We focus on two performance metrics in our study. First, the global volumes uploaded and downloaded. It is important to keep an eye on this metric to assess the ability of a scheduling policy to effectively use the available network capacity. Secondly, the conditional response times in each flow direction as they allow to observe how the scheduling discipline treats each flow size and also if unfairness exists between uploads and downloads or between flows of various sizes.

6.4 The Case of Long-lived Connections

In this section, we evaluate the fairness of the following disciplines: FIFO, BEFD, PEFD, EFDACK, LASCAK, LARS, Run2C and SCFQ for the case of long lived TCP transfers, in order to highlight the impact of half-duplex nature of 802.11 wireless links. In the case of Run2C, we use a variant that takes into account the volume transferred in both directions (by tracking ACK number progress), as otherwise it would only worsen the unfairness. We refer to it as Run2CACK.
Each Qualnet simulation lasts 100 seconds. We consider a scenario with 5 uploads and 5 downloads. The TCP unfairness problem gets more pronounced with decreasing buffer size [46]. This is because the root of the problem lies in the competition to access the buffer of the AP. Conversely, unfairness eventually vanishes for all scheduling disciplines when buffer size increases, although at the cost of extreme queueing delays for *e.g.* FIFO. In our simulations, we considered buffer sizes from 10 to 500 packets. We observed that losses are not observed any more when the buffer reaches around 300 packets. Indeed, since the receiver's advertised window is set to 65 KB, which is equivalent to 43 MSS, at most 5×43 outstanding data packets for the 5 downstream flows and $5 \times (43/2)$ outstanding ACK packets for the 5 upstream flows can be in the buffer at any time (with delayed ACK). For values larger than 300 packets, all policies are fair, although response time explodes for FIFO.
We report below on results for small buffer sizes from 10 to 70 packets. Figure 6.2 depicts the aggregate long term throughput of the uploading and downloading flows, by taking the average of 30 independent simulations.
The pronounced unfairness between uploads and downloads experienced by legacy FIFO is clearly illustrated by Figure 6.2 when the buffer size is small. Moreover, we observe from the ratio of upload to download aggregate throughputs that, the original EFD (*i.e.* BEFD) is even less fair than FIFO, as uploads highly restrain downloads and achieve throughput 2 to 3 orders of magnitude larger than that of downloads when the buffer size is small. This is due to the high priority granted to ACKs as mentioned in Section 6.2.1. With small buffer, this low priority translates into high loss rates for downloads under BEFD and Run2C. In contrast, the loss rates experienced under LASACK, PEFD, EFDACK and LARS are negligible (with a buffer larger than 20 packets). Although Run2CACK keeps track of bidirectional traffic, long lived connections quickly end up in the low priority queue, so that this

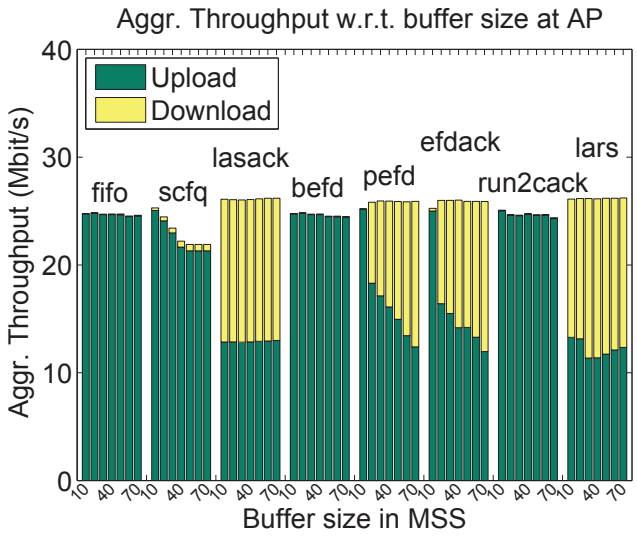


Figure 6.2: Long-lived connections: 5 uploads against 5 downloads

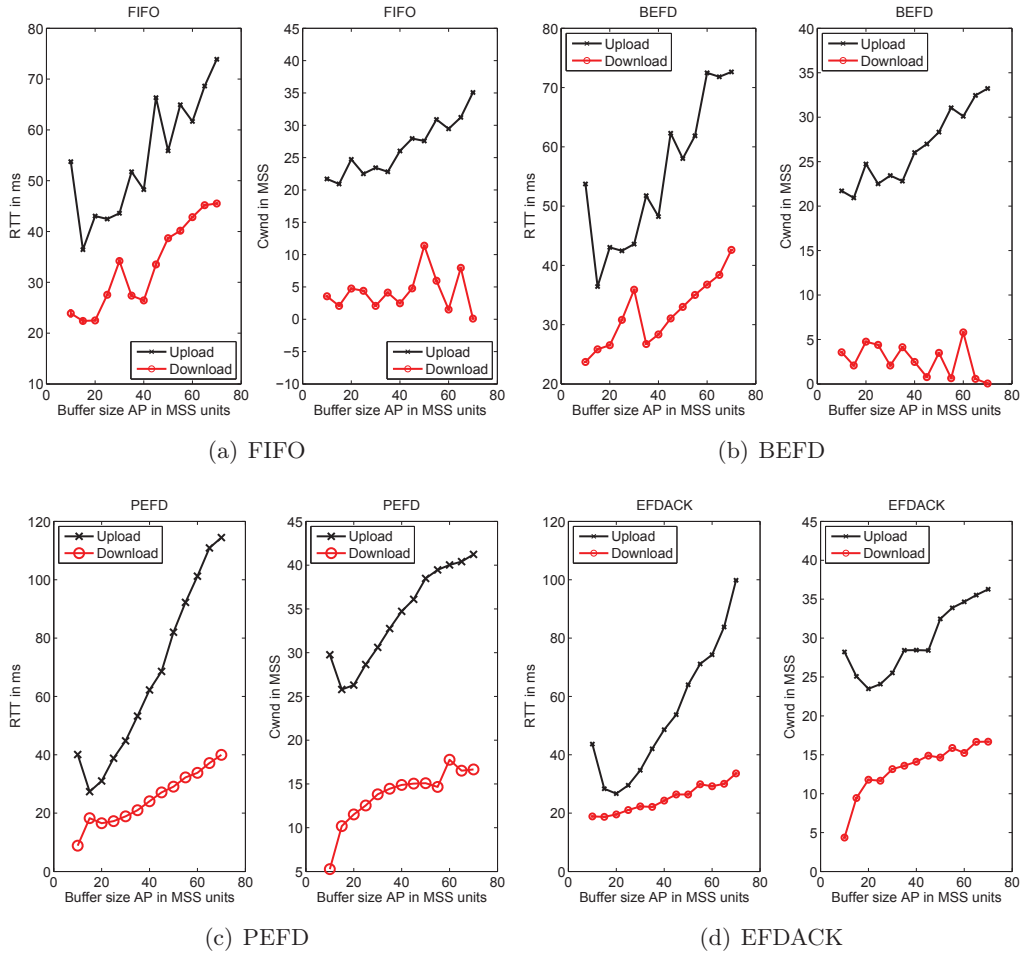
policy degenerates to FIFO in this setup.

Figure 6.2 further demonstrates that the network capacity is fairly shared between uploads and downloads under LASACK [27] and under LARS [28]. Meanwhile, PEFD and EFDACK are able to enforce a good level of fairness – far better than FIFO, SCFQ, and BEFD but not as perfect as LASACK or LARS – when the buffer size is larger than 20 packets. An interesting point is that fairness is not obtained at the expense of performance degradation as the aggregate throughputs under PEFD and EFDACK are larger than the ones of FIFO and SCFQ.

In an attempt to better understand the modus operandi of BEFD, PEFD and EFDACK, we have computed the mean value of the two metrics: RTT and congestion window, both for the uploads and the downloads, as a function of the buffer size at the access point, which are represented in Figure 6.3, by collecting the samples in 30 independent simulations.

A scheduling policy might impact both the congestion window of a flow and its RTT. It can impact the congestion window by creating losses. Controlling the RTT is simply obtained by varying the priority of the packet of the flow at the scheduler. In a sense, losses can be seen as an extreme case of the delay (an infinite delay), hence the RTT is the primary variable through which a scheduler controls a TCP connection. Furthermore, if the scheduler considers only the direction in which ACKs travel, then delaying the ACKs is the only control variable as dropping them has only a limited impact on *cwnd* growth.

We observe first that RTTs are similar between uploads and downloads when the queuing policy does not differentiate between up and down directions. This is the case for FIFO and BEFD. This confirms the fact that there is a single bottleneck (the buffer of the AP) that governs all RTTs. When its size grows, the RTT grows. Second, it is clear that for FIFO, the download congestion windows do not signifi-



(a) FIFO (b) BEFD

(c) PEFD (d) EFDACK

Figure 6.3: How does the scheduler control the connection throughputs? RTT and Cwnd w.r.t. buffer size at AP

cantly grow, so that these connections throughput remains low. With BEFD things are even worse. With EFDACK, uploads and downloads are effectively decoupled by the scheduler that inflates the RTT to compensate congestion window increase. The result with EFDACK is that throughputs of uploads and downloads are eventually similar, *i.e.* the TCP unfairness problem vanishes. We observed a similar effect with LARS, and to a lesser extent with PEFD.

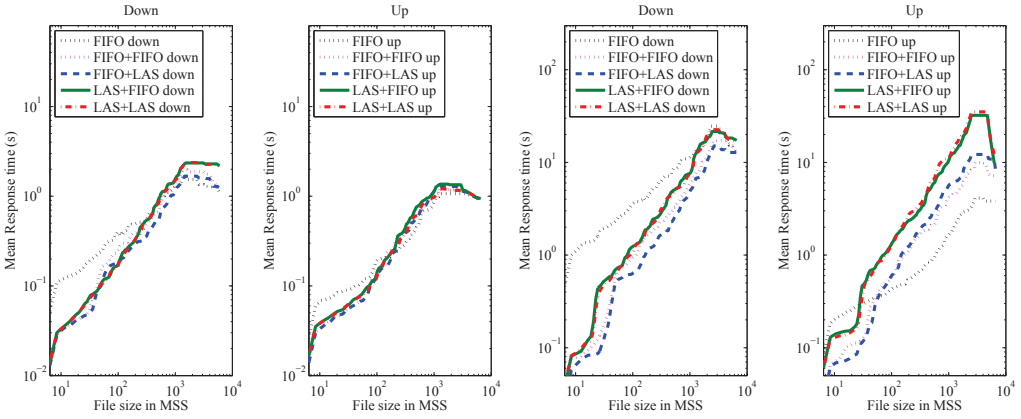
One of the lessons of the above evaluation is that SCFQ and BEFD are clearly ineffective when the traffic consists of both uploads and downloads. This is why we rule them out from further investigation bellow. One can argue that this is also the case for FIFO. However, as FIFO is the legacy scheduling discipline, we keep it as a reference point hereafter.

6.5 Performance Evaluation using Realistic Workloads

In this section, we first investigate the impact of varying the scheduling discipline for EFD like schemes. We consider 4 combinations of disciplines: FIFO+FIFO, LAS+FIFO, FIFO+LAS, LAS+LAS in two different flavors corresponding to a threshold either in byte like in EFDACK or in packets like PEFD. We conclude that the original FIFO+FIFO is a good candidate and thus focus only on the original PEFD and EFDACK in subsequent analyses.
We next compare PEFD and EFDACK to FIFO, LARS, LASACK and Run2CACK. We examine the conditional response time of uploads and downloads, assuming a highly skewed (as the coefficient of variation is 6) flow size distribution. Finally, we discuss the impact of the buffer size at the AP on the performance of scheduling policies in 802.11 networks.
The simulation parameters are given in Table 6.1, and each simulation lasts 5000s. Some connections are unfinished at the end of a simulation due to the premature end of simulation; however, under high load and for long enough simulations as in our case, the main reason is that they were set aside by the scheduler. We report performance results only for the connections that have completed a transfer. In this section, we do not represent on the figures the confidence intervals (for each flow size) as, given the number of curves per figure, they tend to obscure the graphs. Still, they enabled us to check that the simulations were long enough to draw conclusions based on the conditional mean response times. We put these figures and tables related to the confidence interval in the Appendix A.

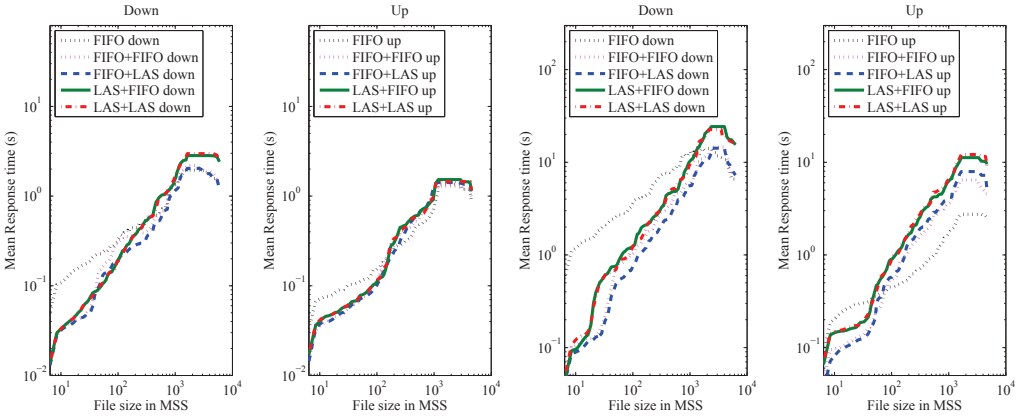
6.5.1 Comparison of EFD Variants

In this part, we consider four variants of EFD: LAS+FIFO, FIFO+LAS, LAS+LAS as well as FIFO+FIFO itself. For each variant, we have two flavors, depending on the bookkeeping option which is either in bytes like EFDACK or packets as PEFD. Before going into the details, we need to explicit the way LAS is used here. This is the global EFD scheduler that assigns the volumes, either in packets or bytes depending on the strategy. Each packet is thus marked with an associated volume and, when LAS is used, it manages the queue where it is applied in such a way that packets are always sorted in ascending order of their associated volume.
We conducted simulations for a symmetric load and 10 Mbit/s (moderate load) and 20 Mbit/s (high load) respectively. The buffer size is set to 30 packets. Average conditional response times of byte-based schemes are depicted in Figure 6.4 while the case for the packet-based schemes are illustrated in Figure 6.5. Results with an asymmetric load are qualitatively similar and we do not present them here.
We observe from Figure 6.4(a) that the 4 schemes perform similarly. They all offer lower response time to short flows as compared to FIFO, but at the cost of a slight increase of completion time for long flows when the offered load is moderate at 10 Mbit/s. A similar effect for the case of packet-based scenario is visible in Figure 6.5(a). When the load is high, the behavior of the 4 different schemes differ especially



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

Figure 6.4: Comparison between various queueing policies in EFD queues – Average response time, symmetric load, byte-based



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

Figure 6.5: Comparison between various queueing policies in EFD queues – Average response time, symmetric load, packet-based

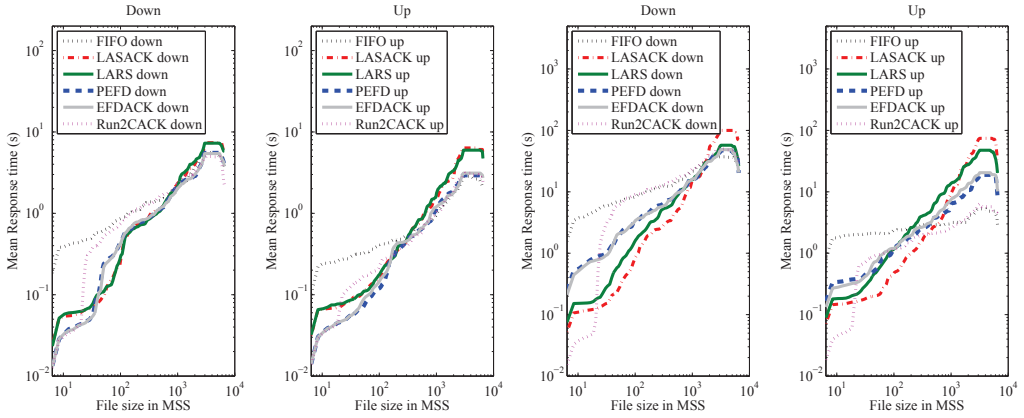
for the byte-based scenario. FIFO+LAS basically offers the best response time for both scenarios, as illustrated in Figure 6.4(b) and Figure 6.5(b). FIFO+FIFO performs quite close to FIFO+LAS for the byte-based scenario. Using LAS in the high priority queue seems detrimental. Though the use of LAS is different from the original LAS policy that has a full knowledge of the history of each flow, we believe that the bad performance obtained when LAS is used in the high priority queue is a consequence of the bad performance of LAS when the distribution has a low variability - as investigated in Chapter 5 for the case of unidirectional traffic and wired networks.

In conclusion, modifying the queuing discipline of each individual queue in an EFD scheduler (reasoning on packet or bytes) appear beneficial only for the low priority

queue and can have a detrimental effect in the high priority. Overall, the benefit of LAS in the low priority queue seems limited in comparison to the increased complexity. We thus consider only the original FIFO+FIFO flavors, namely PEFD and EFDACK in the rest of this chapter.

6.5.2 Impact of Load and Symmetry Ratio

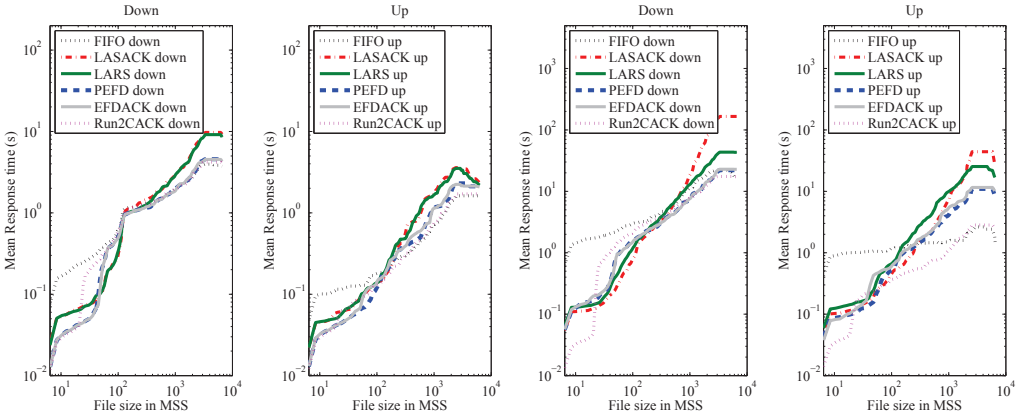
We present simulation results for 10 and 20 Mbit/s and for symmetric ($\frac{\lambda d}{\lambda u}$=1) and asymmetric ($\frac{\lambda d}{\lambda u}$=10) scenarios. The buffer size is set to 30 packets. Conditional response times of uploads and downloads are depicted in Figures 6.6 and 6.7 respectively. The response time is defined as the time required for a TCP connection of a given size to complete its transfer (set-up, data transfer and tear-down).



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

Figure 6.6: Comparison of EFD variants for a symmetric workload: average response time – AP buffer of 30MSS

We first observe that under FIFO, for all the scenarios and all load condition - even a moderate load - the TCP unfairness problem is visible. It is thus a performance problem for any operational 802.11 network.
In contrast, we observe that all size-based scheduling policies mitigate the TCP unfairness problem, while granting a high priority to short flows, whose performance significantly improve as compared to FIFO. These are obtained at the cost of a negligible increase of the response time of long flows.
An important remark is that we present conditional response times as a function of flow size so as to see the impact of the scheduling disciplines on each flow size. However, with a point of view that would perhaps better account for user experience, one could have considered the percentiles of flow size on the x-axis. This would have magnified the left side of each plot because short flows represent the majority of flows, *e.g.*, the 90-th quantile is less than approximately 50 packets, meaning that 90% of the flows experience a significant improvement with the size-based scheduling policies we consider.



(a) Workload of 10Mbit/s

(b) Workload of 20Mbit/s

Figure 6.7: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 30MSS

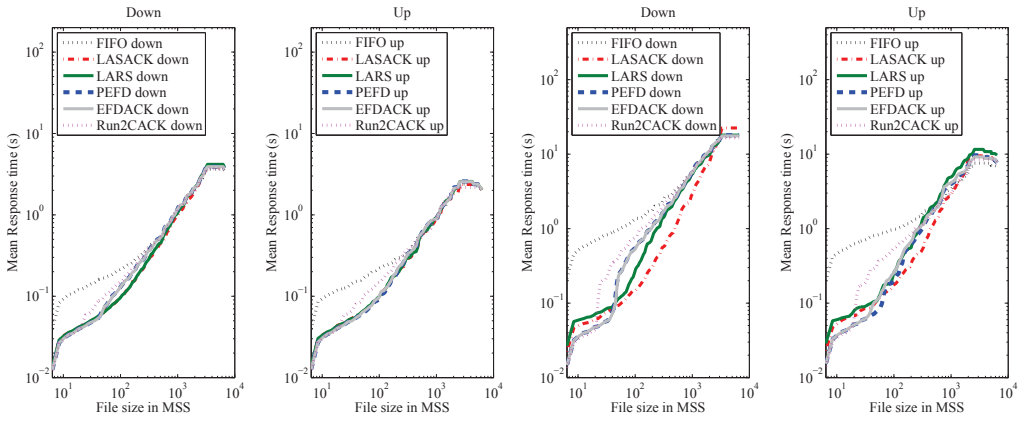
The figures show that LASACK performs slightly better than PEFFD and EFDACK, especially for mid-size-flows. This is a side-effect of the threshold used in PEFD and EFDACK. Overall, the take-away message is that PEFD and EFDACK are able to behave almost as well as state-of-the-art size-based scheduling policies that keep track of all flows (in contrast to EFD like policies that have a memory "limited to the buffer"). Here, Run2CACK uses the same threshold as EFD to decide in which queue a packet should go. But due to its infinite memory, flows go earlier in the low priority queue, following the expected behavior described in Section 6.2. In fact, Run2CACK gives a more marked transition than EFD, with a pronounced protection of short flows detrimental to mid-size ones, so that it is in fact more sensitive to the transition threshold setting.

6.5.3 The Impact of Buffer size at AP

We considered buffer sizes ranging from 10 to 500 packets. We picked two representative values: 30 and 300 packets. Simulations are conducted in an asymmetric load scenario. Results are presented respectively in Figures 6.7 and 6.8.

When the buffer size is large - 300 MSS for instance, there is no more unfairness between uploads and downloads even with FIFO regardless of the load, as the queue rarely overflows. Nevertheless, this is obtained at the cost of very long times spent in the AP downlink queue.

Comparing with figure 6.7, PEFD, EFDACK and LASACK do not suffer nor benefit from larger buffer space. This is in line with our previous results and the results obtained in the original EFD - see Chapter 4, although the buffer size is directly linked to the scheduler "memory". This confirms that, unlike FIFO, (some) size-based scheduling policies are much less sensitive to the actual buffer size.



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

Figure 6.8: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 300MSS

6.6 Conclusion

This chapter presented the adaptation and evaluation of EFD to the case of IEEE 802.11 networks, the most common half duplex links effectively in use. There are basically two ways to do this adaptation: keep track of the volumes exchanged in both directions or simply count packets in a single direction. In fact, as long as the workload does not consist of flows with very disparate MSS, PEFD is a much simpler approach.

Compared to size-based scheduler with infinite flow states memory, EFD is marginally less efficient in combating the TCP unfairness problem than LARS or LASACK; this is especially evident for long lived flow experiments. Nevertheless, for a more realistic workload, this difference vanishes even for relatively short buffers. In brief, the EFD variants presented in this chapter are simple, low overhead schedulers that can effectively improve performance in wireless networks, without the usual drawbacks associated to size-based schedulers.

Chapter 7

The Impact of the Buffer Granularity on the Performance in WLAN

7.1 Motivation

When investigating the performance of scheduling disciplines in packet-switched networks, the buffer is typically a key factor to consider, specifically its (physical) memory size. Intuitively, large buffers can avoid packet loss but increase delay and jitter, while small buffer obviously worsens the packet loss, resulting in disappointed link utilization. The issue of router/switch buffer sizing, which becomes increasingly important in practice, has been extensively studied in the research community. To understand how much buffering is actually needed, many studies have been performed and several rules have been proposed - applicable in different parts of the network as they hold with various assumptions, including the well known "Bandwidth Delay Product"(BDP) rule of thumb [59], Small Buffers Rule [4], Drop-based Buffers Rule [10] and Tiny Buffers Rule [54, 13].

Instead of discussing how to dimension the buffer of a router/switch/access point interface and how buffer size affects network performance, we raise the concern of buffer granularity in this chapter, which inspires from our study of size-based scheduling disciplines over 802.11 Wireless LANs. We term the buffer granularity as the unit in which the buffer size of the network device interface is measured, and we use two units for that in our discussion - byte and packet. Note that networking devices generally limit the size of their queues by the number of packets they can hold as opposed to the number of bytes the packets are worth, although some devices indicate the memory in bytes by default by the manufacturer. In addition, we restrict our discussion to 802.11 Wireless LANs, in which the unfairness issue is commonly raised and highlighted [46].

In this chapter, we focus on the impact of the buffer granularity on TCP performance of scheduling policies. Since TCP accounts for more than 90% of the Internet traffic, a TCP centric approach to measure the impact of buffer granularity would be appropriate in practice. We consider TCP traffic only and report the results for TCP connections.

7.2 Methodology Description

When conducting simulations for scheduling disciplines, it is interesting to highlight the impact of having a buffer in bytes or in packets granularity for unidirectional and bidirectional traffic. In this chapter, we restrict ourselves to the case of single bottleneck link. The simulation setting is similar to what we did in Chapter 6. We report it for clarity. We consider a simple network configuration with 10 wired hosts and 10 wireless stations associated to a single access point, as depicted in Figure 7.1. We use the 802.11a protocol with nominal bit rate of 54Mb/s, with RTS/CTS disabled. Good and fair radio transmission conditions are guaranteed as the 10 wireless stations are at the same physical distance from the access point and in line of sight of each other. The 10 wired hosts are connected to a router with an output rate 10 times larger than its input rate, so that its output queue never builds up. With such a configuration, the bottleneck is the access point. We use QualNet 4.5 to obtain all simulation results. TCP NewReno is used with delayed ACK enabled in the simulations.

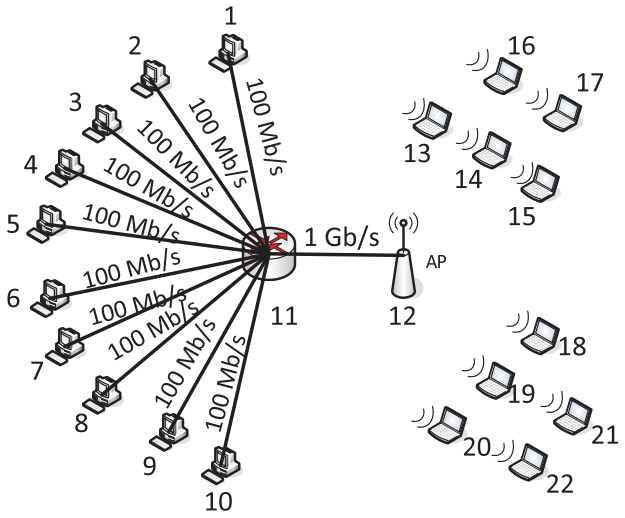


Figure 7.1: Network topology

We consider essentially two workloads. First, we use only long-lived (persistent) flows: while unrealistic, results obtained under such a workload enable to pinpoint easily the impact of the buffer granularity on the performance of a scheduling policy, due to the relative simplicity of the scenario.
Secondly, we consider a more realistic case of a mix of short and long flows. We generate the workload with the assumption that TCP connections arrive according to a Poisson process with rate λ and adjust λ so as to obtain a medium load of 10 Mbit/s, relatively to the maximum throughput of a single TCP transfer over 802.11a at 54 Mbit/s, which is merely 27.3 Mbit/s [16]. The workload consists of bulk TCP transfers of varying size, generated from a bounded Zipf distribution with an average size of about 60 KB (40 packets with size of 1500 bytes each), which

is in line with flow sizes observed on typical campus WLANs [39]. The minimum transfer size is 6 MSS, and the maximum transfer volume corresponds to 10 MB with a coefficient of variation[1] of 6, which controls how the mass of the distribution is split between short and long transfers. Note that bounded Zipf is a discrete equivalent of a continuous (bounded) Pareto distribution, and Pareto is a long tailed distribution usually adopted for modeling flows in the Internet. Each packet has a fixed size of 1500 bytes in our simulations.
A last important parameter of the workload, in a 802.11 scenario where the medium is managed in a half-duplex manner, is the ratio of download to upload traffic. We denote by λ_d and λ_u the arrival rate of TCP downloads and uploads respectively. For simplicity, we considered initially the symmetric load, $\frac{\lambda d}{\lambda u}$=1. Note that in contrast to router/switch in wired network, the access point buffer size tend to be small - typically ranging between 30 and 100 packets.
The disciplines to be discussed include LASACK, LARS, Run2C, BEFD, PEFD, EFDACK as well as FIFO and SCFQ. As in this chapter in the case of Run2C [5], we use a variant that takes into account the volume transferred in both directions (by tracking ACK number progress). We refer to it as Run2CACK.

7.3 The Case of Long-live Connections

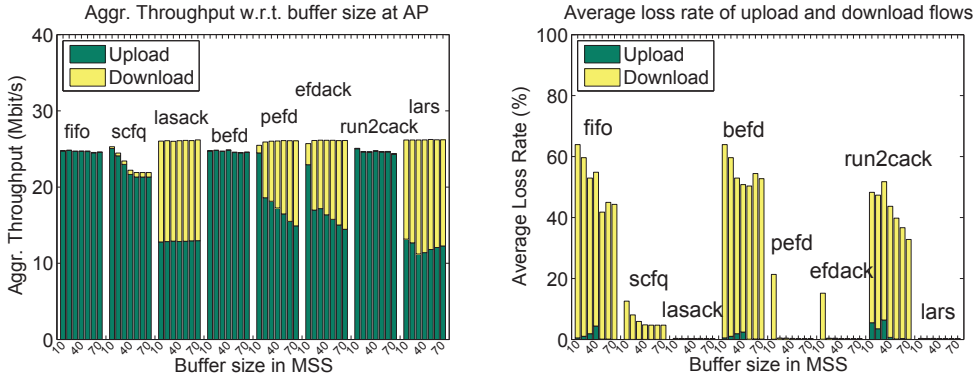
In this section, we consider a scenario of 5 uploads and 5 downloads, with the ratio of download to upload traffic equal to 1. Each simulation lasts 100 seconds. The TCP long-live transfers are triggered at time t=1s and kept active until the end of the simulation. The TCP unfairness problem has been widely observed and discussed in the infrastructure 802.11 WLANs. The studies in the community finally figure out the root behind this phenomenon, which lies in the competition to access the limited buffer of the AP. Conversely, unfairness drops and eventually vanishes for all scheduling disciplines as the buffer size increases, although at the cost of extreme queueing delays particularly for FIFO.
When the buffer size reaches around 300 packets for the particular scenario deployed in this section for the case of long-live connections, the unfairness vanishes and no packet losses are observed. Therefore, in order to highlight the impact of the buffer granularity on the performance of the scheduling disciplines, we restrict the buffer size of the AP to be small in our study. We focus on two metrics when investigating the impact of the buffer granularity - the aggregate throughput of uploads/downloads and the average loss rate of uploads and downloads.

7.3.1 Queue Size in Packets

We conduct the simulation for all aforementioned scheduling disciplines, in which the AP buffer is configured to be filled packet by packet, and the buffer full-checking is performed with the unit of the number of packets. We report the simulation results

[1]The CoV is defined as the ratio of the standard deviation to the mean of a distribution. The larger it is, the more skewed the distribution.

below for small buffer sizes from 10 to 70 packets. Figure 7.2 depicts the aggregate throughput of uploads against downloads and the corresponding average loss rate for both direction flows.



(a) Aggregate throughput

(b) Average loss rate

Figure 7.2: Queue size in packets: 5 uploads against 5 downloads

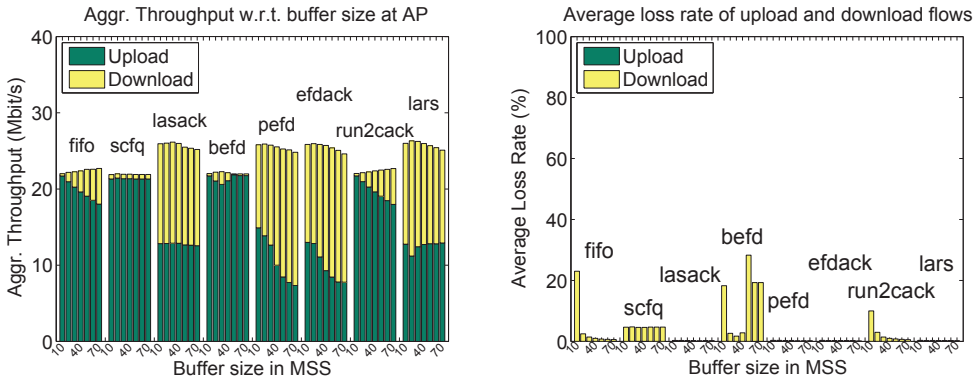
Basically, for FIFO, SCFQ, BEFD and Run2CACK, the downloads are observed to be highly impacted by uploads when competing for the bottleneck bandwidth, which are evidenced by the extremely high loss rate experienced by downloads - although it is high but not really extreme for the case of SCFQ. Note that when the buffer size is measured in terms of number of packets, the buffer is easy to be filled up quickly since there are at most $5 \times (43/2)$ outstanding ACK packets for the 5 upstream flows and 5×43 outstanding data packets for the 5 downstream flows coming to the buffer at any time (with delayed ACK enabled and the receiver's advertised window of 65 KB - equivalent to 43 MSS). The explanation for the pronounced unfairness experienced by legacy FIFO has been clearly understood (see [46]). With BEFD which counts the volumes in terms of bytes, the ACKs of uploads are always granted high priority due to their small size compared to regular data packets of downloads, resulting in uploads monopolizing the network capacity. Although Run2CACK keeps track of bidirectional traffic, long-lived connections quickly end up in the low priority queue, so that this policy degenerate to FIFO in this setup.

In contrast, the network capacity is fairly shared between uploads and downloads under LASACK and LARS. Meanwhile, PEFD and EFDACK are able to enforce a good level of fairness with negligible loss rate when the buffer size is larger than 20 packets. For these policies, fairness is achieved mainly from the effect of the scheduler itself, although the buffer granularity may slightly change the overall throughput.

7.3.2 Queue Size in Bytes

As an alternative, we switch the buffer granularity from the number of packets to bytes and re-run the simulations, keeping other network settings unchanged. In this

case, the AP buffer incorporates packets by accounting for their equivalent size in bytes, and test if the buffer is full is performed with the unit of bytes. We report the simulation results for the case in which the buffer size is measured in bytes in Figure 7.3. After changing the buffer granularity from packets to bytes, the performance improvements are noticeably observed for FIFO, Run2CACK and BEFD, while for the remaining policies the impact is limited.



(a) Aggregate throughput

(b) Average loss rate

Figure 7.3: Queue size in bytes: 5 uploads against 5 downloads

Note that, given the same buffer size[2], byte-based granularity will be able to afford more packets (ACKs of uploads or data packets of downloads) than packet-based granularity when uploads and downloads compete at the same time for the buffering. Specifically, for the scenario of 5 uploads against 5 downloads we adopt, 5 upstream flows will easily fill up the buffer by emitting at most $5 \times (43/2)$ ACKs simultaneously for the latter case, while significant space will be left for data packets of downloads to grab for the former case since $5 \times (43/2)$ ACKs with size of 40 bytes each make up only a small percentage of the buffer (For example, with the maximum buffer size of 30MSS, $5 \times (43/2) \times 40$bytes which comes to 4.3KB account for less than 10% of the buffer size). Consequently, more packets of downloads are able to be incorporated in the buffer and avoid being dropped, resulting in a drastically decreased loss rate for FIFO, Run2CACK and BEFD, as illustrated in Figure 7.3. In addition, fairness is improved as well for these three disciplines as more bandwidth is observed to be assigned to downloads. Since packets of downloads get more opportunities to enter the buffer due to the effect presented above, downloads obtain slightly higher aggregate throughput than uploads under PEFD and EFDACK, with still a good level of fairness.

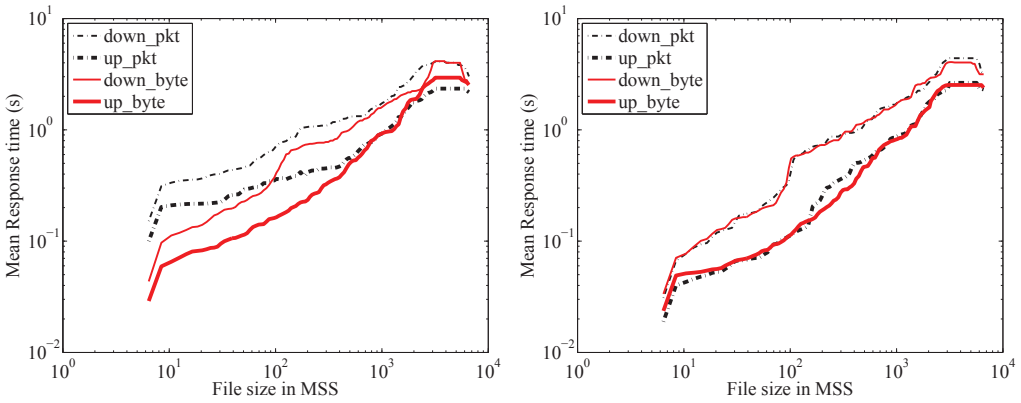
LASACK and LARS, as the schedulers themselves fairly treat upstream and downsream flows, are highly fair, with no packet loss experienced no matter the AP buffer is measured in packets or in bytes. Therefore, LASACK and LARS are almost in-

[2]For simplicity, we suppose that each IP packet has a fixed size of MSS - 1500 bytes throughout our discussion. Based on this assumption, it makes no difference when we specify the maximum buffer size for the simulation in packets or in equivalent bytes.

sensitive to buffer granularity.

7.4 Mixed Workload of Short and Long flows

We investigate the impact of buffer granularity by examining the conditional response time of uploads and downloads in this section, assuming a highly skewed (as the coefficient of variation is 6) flow size distribution. We run the simulations for a symmetric load of 10Mbit/s, setting the buffer size to be 30 MSS. The simulations are conducted in two scenarios with different buffer granularity - the unit of packets and bytes respectively. Each simulation lasts 1000 seconds. We demonstrate the results in Figure 7.4 and Figure 7.5, in which line styles along with colors are used to denote different scenarios while line widths are used to indicate traffic in two directions.

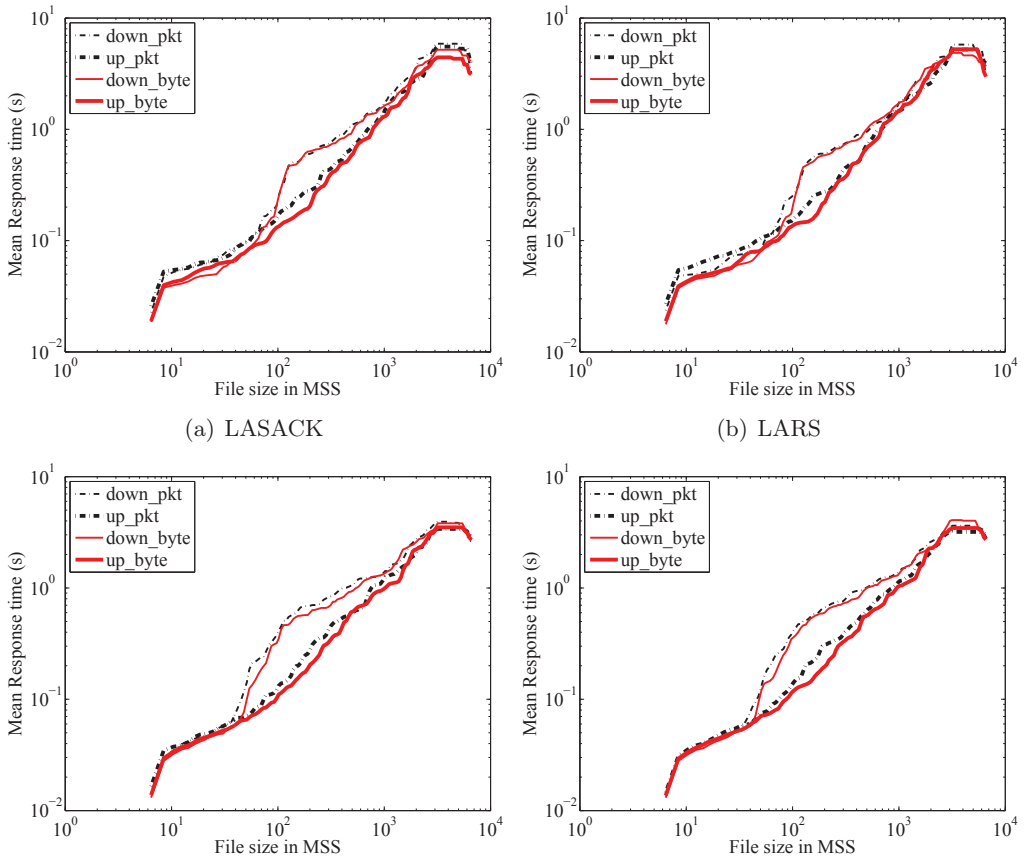


(a) FIFO (b) SCFQ

(c) BEFD (d) Run2CACK

Figure 7.4: Average response time, symmetric load, 10Mbit/s workload

In the case of mixed workload - which is believed to be closer to the reality, measuring the buffer with the unit of bytes is highly preferred for FIFO, Run2CACK and BEFD as it provides significantly lower conditional response time for the majority



(a) LASACK (b) LARS

(c) PEFD (d) EFDACK

Figure 7.5: Average response time, symmetric load, 10Mbit/s workload (cont.)

of the flows from both two directions, especially for small and medium size flows, although the fairness between uploads and downloads in terms of response time is not improved by observing the results in Figure 7.4. Recall that the unfairness in 802.11 WLANs lies in the competition for accessing the limited buffer of the AP between the upload and the download. When the buffer size is in bytes, the download is granted more opportunities to reside in the queue and then to be served, avoiding being dropped frequently as what happens in the scenario of packet-based buffer granularity.

Not surprisingly, SCFQ, LASACK, LARS, PEFD, EFDACK are observed to be insensitive to the buffer granularity in the case of mixed workload with heavy-tailed flow size distribution. However, the unfairness is quite pronounced for SCFQ, in terms of high performance discrepancy between uploads and downloads. Unlike SCFQ, the other policies (LASACK, LARS, PEFD, EFDACK) shown in Figure 7.5 enforce a good level of fairness for most of the flow sizes.

7.5 Conclusion

In this chapter, we have investigated the impact of the buffer granularity (instead of the buffer sizing) on the performance of scheduling disciplines over 802.11 WLANs. The discussion is conducted with two buffer granularities - packets and bytes, and two workload scenarios. We investigate the bottleneck link capacity sharing between uploads and downloads considering as metrics the aggregate throughput for the case of long-lived connections, and mean conditional response time in the case of more realistic workload with heavy-tailed size distribution. We conclude that measuring the buffer with the unit of bytes is highly preferred for FIFO, Run2CACK and BEFD, while LASACK, LARS and SCFQ are insensitive to the buffer granularity.

Part III

Workload Model for Enterprise Networks

Chapter 8

Traffic Analysis of Enterprise Networks

8.1 Introduction

Wide-area Internet traffic has been widely studied in many different environments from the research communities over the years [8, 22, 15, 36, 44, 6]. However, the traffic pattern and the performance issue within modern enterprise networks remains nearly unexplored. The likely reason to explain lies in the difficulty in adequately monitoring enterprise traffic and the belief of perfect performance in the enterprises in practice.

Recently, a noticeable amount of related work on enterprise networks have been published [43, 21, 40, 12]. The attention on the enterprise network stems from several aspects: on one hand, enterprise networks have evolved from site-centric wired networks where users' machines access application servers through a fixed infrastructure to the case where users are roaming, either from a wired to a wireless network or from inside the company to outside through a VPN access; one the other hand, the ever-increasing variety of applications used in Intranet, e.g. voice and video over IP, together with consolidation of servers through virtualization and of data through SAN (Storage Area Networks) both being eventually integrated to offer highly resilient services, have significantly increased the complexity of enterprise networks.

Recent studies by Ruoming Pang *et al.* [43] and Boris Nechaev *et al.* [40] have taken an initial step towards profiling the internal traffic in modern enterprise network in several aspects, trying to raise up again people's attention over it since most of the previous work over enterprise traffic available in the literature are rather over a decade old. Although their study are based on datasets collected from a single site (LBNL) with significant limitations such as unexpected anomalies of traffic missing, it still provides a good example of what modern enterprise traffic looks like. Our starting point here is to develop an understanding of the basic characteristics of modern enterprise traffic at various levels by examining the packet trace captured in another enterprise - Eurecom[1], instead of the LBNL trace[2], and if possible, further compare to the findings reported in [43] and [40]. Note that, our study is not limited

[1] Eurecom is a medium-size laboratory located in south of France, consisting of around 800 distinct hosts.

[2] The anonymized version is publicly released at http://www.icir.org/enterprise-tracing/.

to the enterprise internal traffic as [40], but also incorporate the traffic across wide-area network exchanging between local peers and remote peers located beyond the enterprise boundary, denoted as "external traffic" for the following study.
From the enterprise network's perspective, we define "*intranet traffic*" as the traffic exchanging within the enterprise, and "*internet traffic*" as the traffic generated by the communications between local peers and remote peers located outside of the enterprise. Thus, "*intranet traffic*" and "*internal traffic*" are equivalent to each other and interchangeable in this work, and so as "*internet traffic*" and "*external traffic*". The significant contribution of this work is contrasting the external and internal activity exposed in modern enterprise networks. In addition, special attention is given to find an automatic way to identify different roles (servers or clients) inside the enterprise networks, relying on a supervised machine learning approach. This becomes an important issue when one has to process anonymized enterprise traffic traces, *e.g.* the LBNL traces.

8.2 Datasets

The dataset used in this study are captured at our own network (Eurecom) on January 25 2010, with a duration of 24 hours. Eurecom is a medium size enterprise, consisting of hundreds of workstations and several tens of servers equipped with a variety of operating systems. Inside Eurecom network, users' machines access application servers in a wired or wireless manner. The traces are obtained by monitoring a number of individual switches (switches connecting subnets inside the enterprise and edge switches bridging enterprise network and the wide-area Internet), and further merged to form a more complete trace. In addition, duplicate traffic (both hardware and software duplicates) are erased from the trace. Thus, traffic flowing between local peers within Eurecom network termed as "*internal traffic*", and the ones exchanging between local peers and remote peers (located beyond the Eurecom boundary) termed as "*external traffic*" are all incorporated in a single trace.
Table 8.1 provides an overview of the collected Eurecom packet trace, together with the LBNL traces publicly released [32], in which the basic information of the datasets are given, including the number of connections/flows, number of distinct local hosts monitored, volume in bytes and volume in number of packets, along with the date on which the traces were captured and the duration. Compared to LBNL traces collected in year 2004 and 2005 with a duration of 1 hour at most, Eurecom trace consists of a larger traffic volume (439.3GB) and lasts longer - one full day.
To gain a global understanding of the enterprise traffic, we next take an examination of the traffic composition of the Eurecom trace. For this study, external and internal traffics are separated from the whole Eurecom trace, accounting for around 10% and 90% of the overall traffic volume (either in bytes or in number of packets) respectively - which is in line with our observation at host level in Section 8.3.3 - meaning that the majority of traffic observed is local to the enterprise. As we have a knowledge of what role a host plays (a server or a client) during each transfer at Eurecom

Table 8.1: Dataset characteristics

	Eurecom	LBNL1	LBNL2	LBNL3	LBNL4	LBNL5
Date	25/01/10	04/10/04	15/12/04	16/12/04	06/01/05	07/01/05
Duration	24 hr	10 min	1 hr	1 hr	1 hr	1 hr
# cnxs	1,506,538	76,311	392,832	217,472	126,683	154,981
Bytes	439.3GB	14.3GB	37.3GB	15.9GB	10.5GB	13.7GB
# packets	564.3M	17.7M	65.2M	28.9M	20.5M	26.4M
# local hosts	451	2,914	3,532	2,653	1,259	1,316

network, we classify the traffic as four categories as follows:

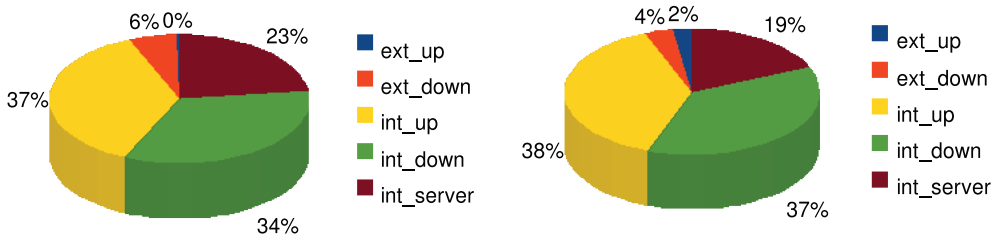
- local client and remote peer (lc-rp): traffic flowing across wide-area Internet between local clients within the enterprise and remote peers outside of the enterprise, like Web browsing.
- local server and remote peer (ls-rp): traffic coming from the communication between local servers and remote peers located beyond the enterprise boundary, for instance, automatic updating by contacting remote servers.
- local client and local server (lc-ls): traffic remaining within the enterprise, between local clients and local servers, which are highly expected in an enterprise network such as IMAP, DNS or distributed file system requesting service to local servers.
- local server and local server (ls-ls): traffic remaining within the enterprise, between local server and local server, for example periodical update or backup inside the enterprise.

This classification helps us to understand what kind of traffic dominates the enterprise traffic and further to explore the possible roots behind it. The traffic composition of the Eurecom trace is given in Table 8.2, in which the overall traffic is globally divided into two components: external traffic and internal traffic - in each, the absolute value of the volume (number of cnxs, bytes, and number of packets) and the corresponding percentage are clearly presented for each traffic category.
Table 8.2 reveals that transfers between local servers and remote peers are rarely observed in external traffic, while transfers between local clients and remote peers carry the majority of the external traffic volume, accounting for around 99% in total. This observation gives us an intuition that local clients contact remote peers much more frequently than local servers. By contrast, transfers between local clients and local servers account for more than two thirds of the internal traffic volume, while transfers between local servers take the rest - less than one third.
To understand the traffic breakdown in two directions (simply refer to upload and download) in enterprise network, we proceed a bit further on the decomposition of the enterprise network. We therefore restrict our attention to the transfers between

Table 8.2: Traffic composition (EURECOM)

trace	EURECOM			
traffic category	External		Internal	
	lc - rp	ls - rp	lc - ls	ls - ls
# cnxs	335,329	1,428	500,524	300,688
# cnxs(%)	99.6	0.4	62.2	37.4
bytes	26,159,335,855	278,365,145	258,862,570,217	83,937,654,563
bytes(%)	98.9	1.1	75.51	24.48
packets	34,230,743	435,295	390,422,041	96,022,700
packets(%)	98.7	1.3	80.25	19.74

clients and servers only, for both external and internal traffic. We define uploads as the transfers which originate from the client side and convey bytes from clients to servers, whereas downloads are the transfers initiated by the server side, with traffic flowing from servers to clients. The make-up of Eurecom trace in terms of uploads and downloads is illustrated in Figure 8.1 in bytes and in packets respectively, in which immature transfers (transfers terminated in the setup stage) are omitted. In this make up, five components are examined: external uploads, external downloads, internal uploads, internal downloads, and internal transfers between local servers. Note that we keep the internal transfers between local servers in the pie chart, considering the fact that this part accounts for nearly one fifth of the total volume, either in bytes or in packets.



(a) Volume in bytes

(b) Volume in packets

Figure 8.1: Traffic composition, Eurecom

Figure 8.1 shows that internal traffic makes up a significant share (more than 90%) of the total volume, while outbound transfers, which mainly consist of web browsing[3],

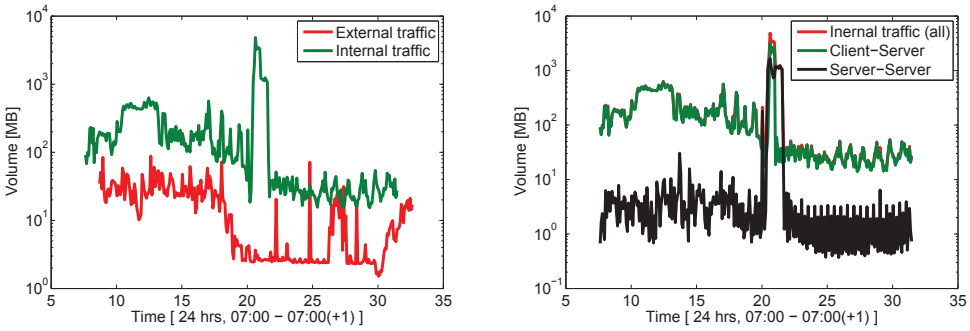
[3]Note that, peer to peer applications are generally blocked due to the organizational policy in

generally generate limited traffic. Interestingly, internal traffic is observed globally symmetric as uploads and downloads almost fairly share the volume, either in bytes or in packets. It does not hold for external traffic however.
Similar examination is supposed to perform on LBNL traces as well. However, it is difficult to conduct due to the lack of direct knowledge to adopt for LBNL traces. To this end, we have developed an approach to distinguish between servers and clients (role assignment problem) in Section 8.4.

8.3 Traffic Analysis for Enterprise Networks

8.3.1 Evolution of load over time

In traffic analysis, a classical technique is to look at the evolution of load over time to detect the busy hours. In Figure 8.2, we present the traffic load of the Eurecom trace within five-minute bins over 24 hours. From Figure 8.2(a), we found that interval communication reveals significantly higher load than external communication at most of the time. Moreover, it is evident that high load appears in the working period in a day - from 8:00 to 20:00, for both internal and external traffic. As a particular phenomenon, extremely high load for internal transfers can be observed from 20:00 to 22:00, because of the daily system backup procedures.



(a) External-Internal (b) Internal traffic

Figure 8.2: Evolution of load over time, Eurecom

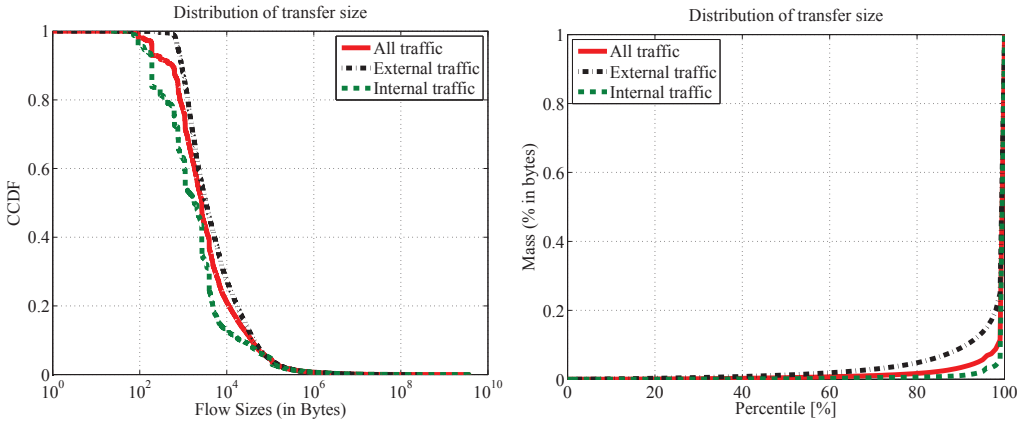
To further understand how internal load varies over time, two classes of transfers making up the internal communication - "client-server" and "server-server" are presented in Figure 8.2(b). It shows that "client-server" transfers carry the bulk of the internal traffic, hence dominates the trend of the internal load over time. In addition, fairly high load generated by two classes of transfers in the period from 20:00 to 22:00 provides an evidence on the interpretation of the phenomenon (backup) in Figure 8.2(a).

most of the enterprises. Also note that since 2010, the share of internet traffic has a bit increased with the rising of HTTP progressive download traffic, *e.g.* YouTube.

8.3.2 Flow-Level Characteristics

In a typical measurement, per-flow analysis has various advantages over per-packet analysis since a single flow (often regarded as a "connection") represents a group with the same 5-tuple packets, and holds abstract information such as the flow duration and flow sizes. Here, we analyze the basic flow-level characteristics for enterprise traffic.

We first examine the flow sizes of the Eurecom trace, whose distribution over day period is given in Figure 8.3. Here we focus on the period during which the traffic is observed to be stationary in Figure 8.2. From Figure 8.3, we confirm that the distribution of enterprise flow sizes exhibits a heavy tail - more than 90% of connections are small, while less than 5% of the largest connections carry the majority of the bytes, demonstrated by mass-weighted distribution in Figure 8.3(b).

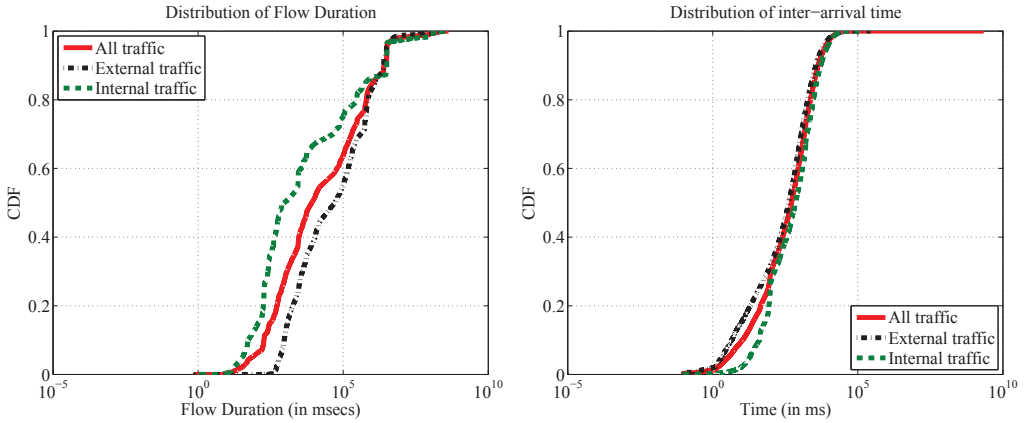


(a) CCDF (b) Mass-Weighted Distribution

Figure 8.3: The distribution of flow sizes, Eurecom

The distribution of flow durations and the distribution of flow inter-arrival time are respectively shown in Figure 8.4 (a) and (b). We observe from Figure 8.4(a) that internal flows have a duration ranging from 1ms to over 100s and taking a median of around 500ms, whereas the duration of external flows starts from 500ms up to a few hundreds seconds, with a median of around 10s, especially due to the RTT discrepancy in the magnitude between internal and external traffic - which is going to be examined in a later section. In addition, no evident differences between internal and external flows are observed concerning the distribution of flow inter-arrival time in Figure 8.4(b).

We next consider the distribution of inter-arrival time and the distribution of flow size, visually fitted with four well-known distributions: Exponential, LogNormal, Weibull and Pareto, by plotting the quantile-quantile plots of two samples - one from the real trace (Eurecom), the other one generated from the theoretical distribution. Visually, we are able to see how each distribution fits the real enterprise traffic (For simplicity, we do not distinguish external and internal traffic here, but simply take



(a) Distribution of flow durations

(b) Distribution of flow inter-arrival time

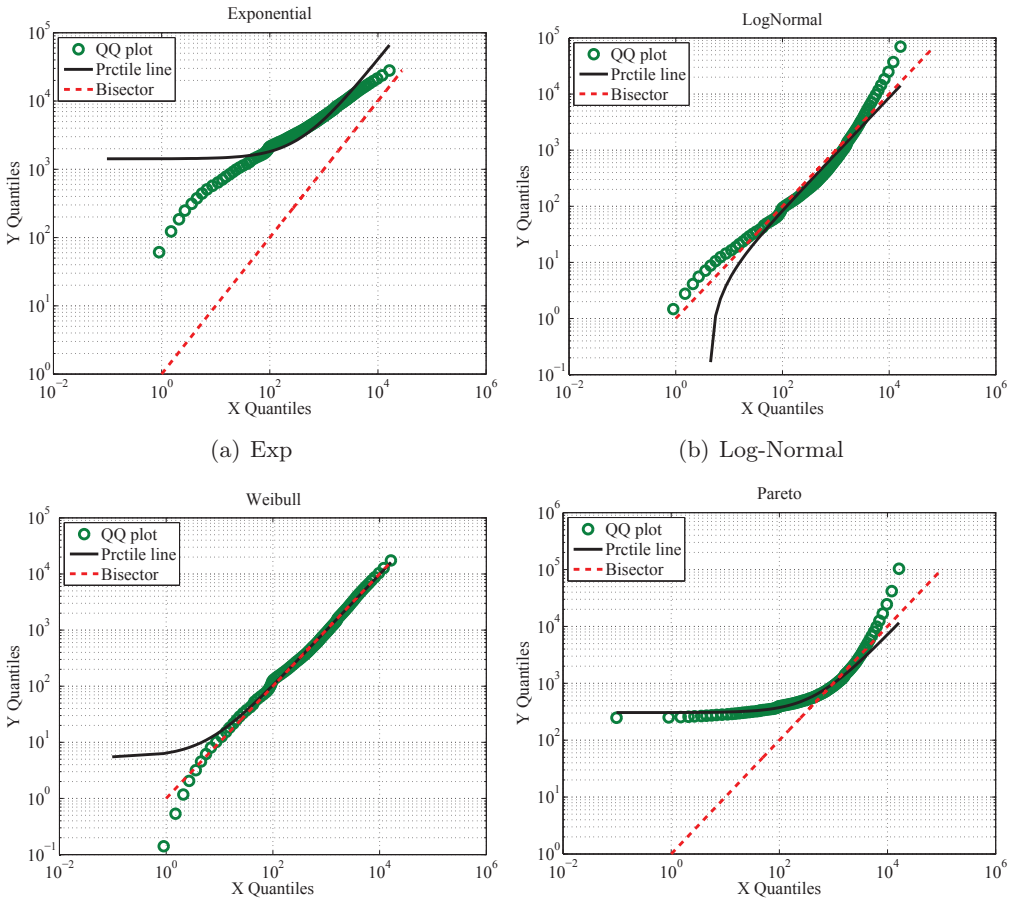
Figure 8.4: CDF of the distribution of the flow duration and flow inter-arrival time, Eurecom

the overall traffic for testing). We observe that for Eurecom traffic, Weibull fits the distribution of inter-arrival time best while LogNormal fits the distribution of flow sizes best, which are illustrated in Figure 8.5 and Figure 8.6 respectively.

8.3.3 Host-Level Characteristics

From our review of existing research, we observe that previous studies are either based on per-packet or per-flow analysis. So far, host-based analysis has rarely been studied and we consider it here as we believe that host-based measurements are valuable in understanding how traffic flows in an enterprise network.

How traffic flows in terms of external and internal traffic, and further into two directions have been globally assessed in Section 8.2 in a coarse manner. We further examine this issue with a host-based method in this section. For this purpose, we pick a number of local hosts within the enterprise, which communicate with local servers as well as with remote servers, i.e. these local clients generate both external and internal traffic. Finally, 224 distinct clients are chosen from Eurecom trace, named as M1, M2, ... and so on. We sum up the volume transferred related to each selected host based on whether it communicates with the peer inside or outside the enterprise, denoted as “internal” and “external” respectively from a host’s point of view. We then compute the ratio between these two quantities for each distinct host, in which the traffic is measured in bytes and in packets respectively. The traffic ratio of the hosts and the distribution of ratios are given in Figure 8.7. Interestingly, more local traffic are observed than remote traffic for most of the hosts - evidenced by the fact that most of the ratio values are greater than 1 as shown in Figure 8.7(a), which is in line with the intuition that local clients tend to access local peers (servers) more frequently than remote peers (servers). In addition, more than 75% of the hosts have a ratio of internal to external traffic volume in between 1 and 100,



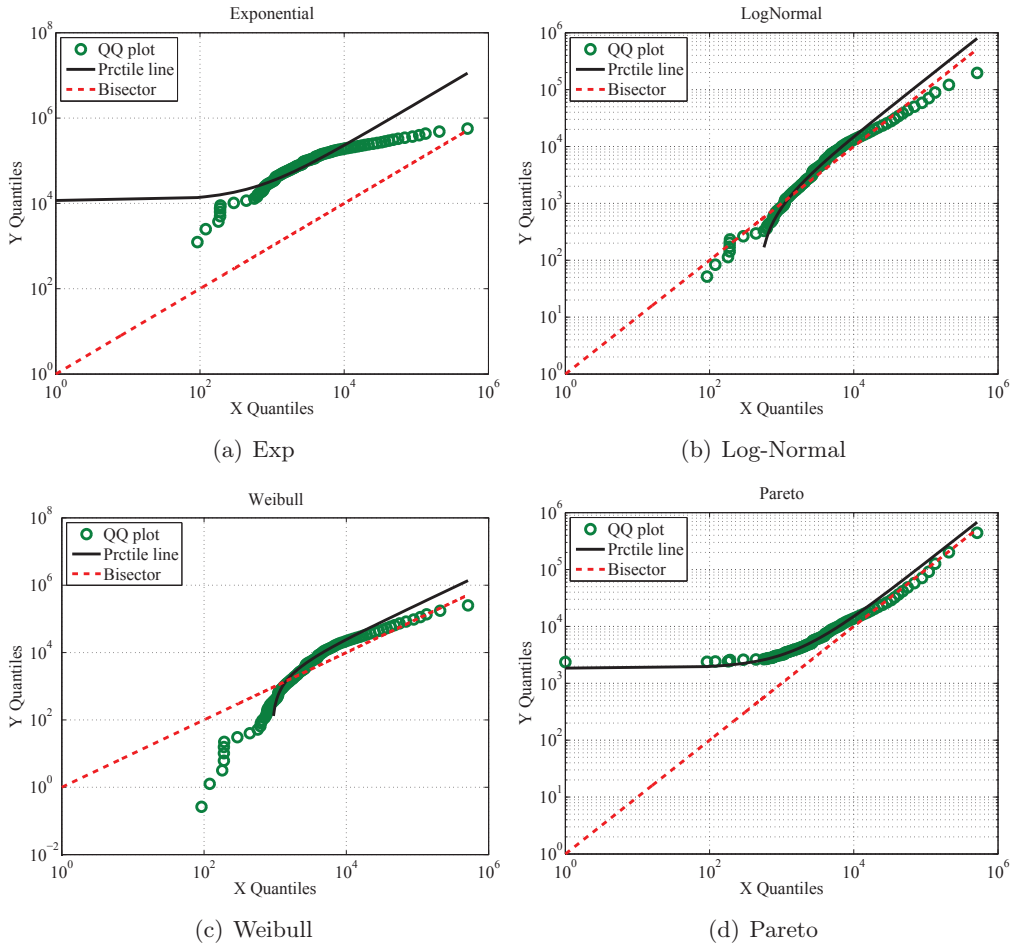
(a) Exp (b) Log-Normal

(c) Weibull (d) Pareto

Figure 8.5: QQ Plot, inter-arrival time, Eurecom

as shown in Figure 8.7(b), with a median around 10, either in bytes or in packets. We next turn our attention to the study of how traffic flows in two directions. To this end, we examine the symmetry level in terms of down-up ratio for internal and external traffic respectively. On one hand, we plot the down-up ratio for each host by measuring the traffic volume in bytes in Figure 8.8(a); one the other hand, the distribution of the down-up ratio is further provided in Figure 8.8(b). As expected, internal and external traffic exhibit significantly different symmetry level as internal uploads and downloads tend to fairly share the traffic, with a median ratio of around 1, whereas downloads are more preferred than uploads for external transfers, with a median ratio of around 10. An intuitive observation in Figure 8.8(a) is that points in the plot denoting the down-up ratio of internal traffic are distributed surrounding the baseline with value of 1, while the points for external traffic stand around a value of 10.

Finally, we note that external traffic is more prone to download than upload evidenced by the asymmetrical phenomenon observed in Figure 8.9(a) when the traffic



(a) Exp (b) Log-Normal

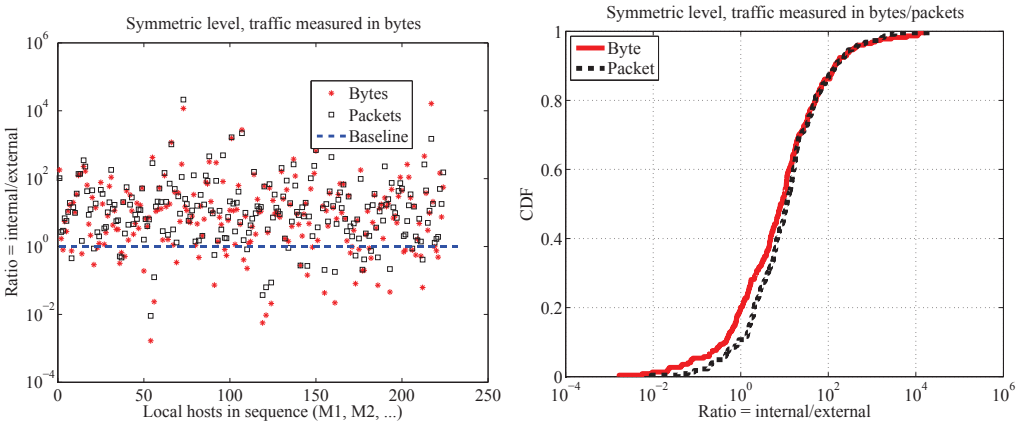
(c) Weibull (d) Pareto

Figure 8.6: QQ Plot, flow size, Eurecom

is measured in bytes, while it is more close to symmetrical when the traffic is measured in packets, shown in Figure 8.10(a). However, the case is opposite for internal traffic, see Figure 8.9(b) and Figure 8.10(b). The later observation means that there are more packets uploaded from clients to servers than in the reverse direction, but they are smaller in size.

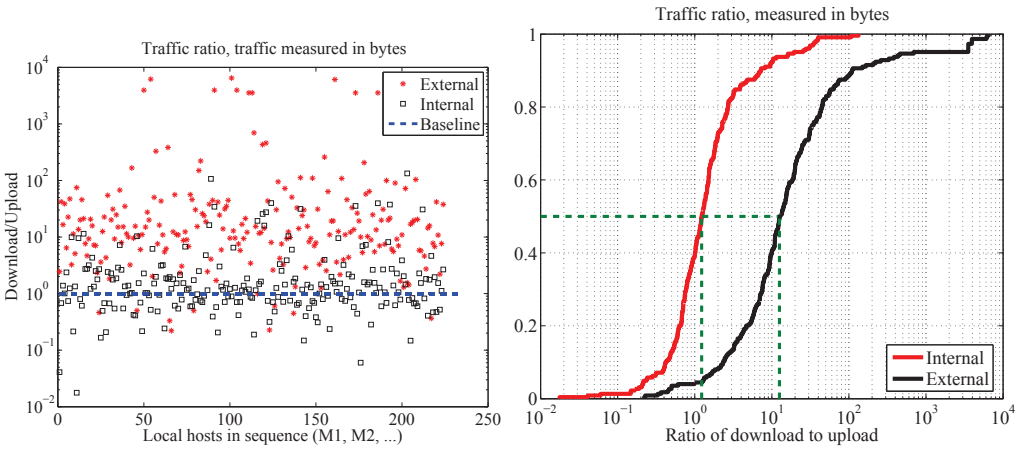
8.4 Role Identification

A problem raised up during our study on anonymized LBNL traces in which limited information is given for the sake of security consideration. Identifying server/client within the enterprise networks is necessary if we want to compare (qualitatively and not qualitatively) Eurecom and LBNL traffic traces. Detecting server/client is however not easy due to the increasing complexity of modern enterprise networks - note that within the enterprise it is not rare for lots of client machines to be servers (*e.g.*, as Windows file shares).



(a) Host-based plot (b) CDF

Figure 8.7: Host-based traffic ratio, Eurecom



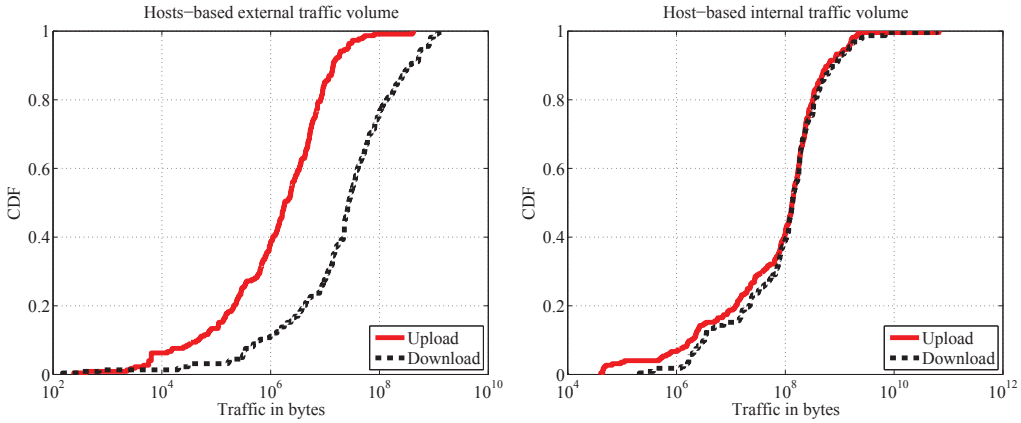
(a) Traffic in bytes (b) CDF

Figure 8.8: Host-based volume in two directions, Eurecom

Logically, a server and a client behave differently, which can be represented in various manners. However, not all the hosts take the role of "pure server" or "pure client" in the context of enterprise networks. Our starting point is to propose a method to automaticly identify representative server/client relying on the feature they expose.

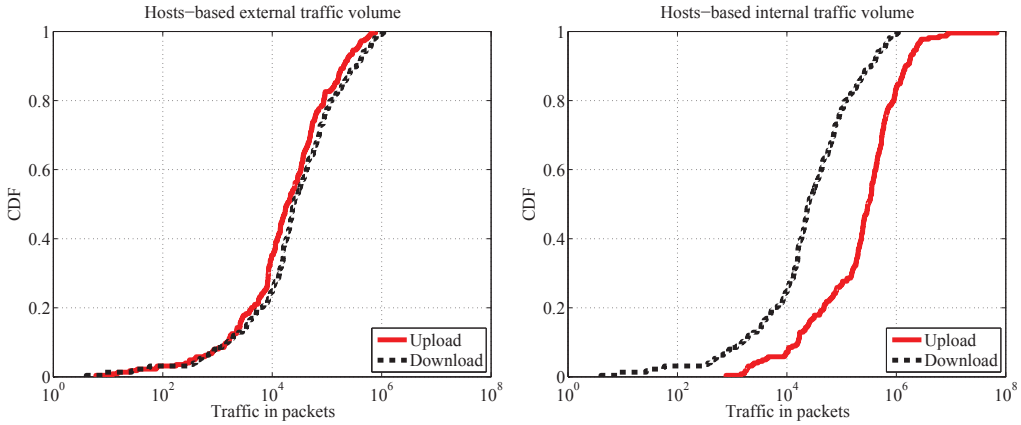
8.4.1 Method description

In our method, we propose to measure 6 features of a host - number of incoming connections, number of outgoing connections, number of distinct destination ports of incoming connections, number of distinct destination ports of outgoing connections, number of hosts that established a connection to it (*fan in*), and the number of hosts it connected to (*fan out*). We use a 6-tuple vector to illustrate the features



(a) External traffic in bytes (b) Internal traffic in bytes

Figure 8.9: Distribution of traffic volume in two directions, Eurecom



(a) External traffic in packets (b) Internal traffic in packets

Figure 8.10: Distribution of traffic volume in two directions, Eurecom

of a host, abbreviated as {*incnx, outcnx, dportin, dportout, fan-in, fan-out*} for the 6 metrics defined above. Our intuition is that servers are prone to have more incoming connections than outgoing connections, larger fan-in than fan-out, whereas it is opposite for the clients. In addition, the number of distinct destination ports indicate the applications implemented from the server side or the applications used from the client side.

The general idea is to extract sets of instances[4], consisting of six attributes each (*incnx, outcnx, dportin, dportout, fan-in* and *fan-out*) from the Eurecom trace, along with a tag specifying the role as we have a knowledge of the role each host takes in Eurecom network; as an important step in machine learning, the dataset

[4]Note that, we identify each host based on its unique IP address, and therefore each instance corresponds to a host.

is filtered before the learning process starts; then a machine learning algorithm (we test both Naive Bayes method and the decision tree scheme C4.5 in our study) is applied on the subset of the instances for training to derive a classifier (in fact a decision tree consisting of a set of rules), and the classifier verification is done on test set extracted from the rest of the instances; finally the verified classifier is used on LBNL traces which are anonymized to identify the role of a host (client or server). Considering the fact that the absolute values of the same metric measured may vary a lot from trace to trace, due to the scale of the enterprise network where traces are captured, different traffic load in different time period and whatever reason can be, we propose to apply normalization techniques (Linear transformation or Student's t transformation) to the six attributes of each instance before sent for deriving the classifier. The whole procedure represented above for the role identification issue is illustrated in Figure 8.11.

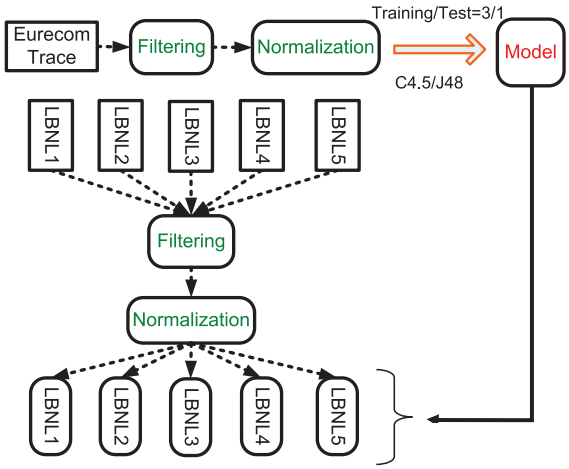


Figure 8.11: Procedure for role identification

8.4.2 Algorithm Validation

There are various approaches to determine the performance of classifiers. A simplest method is to set aside a certain amount of data for testing - this is referred to as cross-validation method - and the remainder is used for training. As commonly adopted in practice, we hold out one-fourth of the data for testing and use the remaining three-fourths for training in this work. The hold-out procedure is repeated four times by in turn taking every one-fourth of the data for testing and the rest for training each time.

To estimate the performance of a classifier, one way is to collect all estimates (success rate or accuracy) on test data and compute average and standard deviation of the accuracy. Note that accuracy is generally measured by counting the proportion of correctly predicted examples in an unseen test dataset.

Suppose that we measure the performance of the classifier on a test set and obtain a

certain numeric success rate, say 85% - meaning that we are likely to obtain a success rate close to 85% when we apply this classifier on a target dataset. But how close? within 5%?. To better represent how close the success rate on a new dataset to the estimate on the test set, we indroduce a success-rate confidence interval (originally defined in [62]), that is the success rate p lies within a certain specified interval with a certain specified confidence. Suppose that out of N trials, S are successes. The success rate f is defined as $f = S/N$. The central limit theorm says that, for large N (say, $N > 100$), the distribution of random variable f with mean p and variance $p(1-p)/N$, approaches the normal distribution. So that the probability of the random variable f, with mean p and variance $p(1-p)/N$, lies within a certain confidence range of width $2z$ is

$$P_r[-z < \frac{f-p}{\sqrt{p(1-p)/N}} < z] = c.$$

Finally, an expression for the confidence interval is given as:

$$p = (f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}})/(1 + \frac{z^2}{N}).$$

The $\pm$ in this expression gives two values to p, representing respectively the upper and lower confidence boundaries.

8.4.2.1 The test on Eurecom trace

To test whether our algorithm works and how well it performs, we start with the Eurecom trace, in which each instance in the dataset is tagged in advance - that is we know if a machine is a server or a client. We use two supervised machine learning methods - Naive Bayes and C4.5/J48 - to train the classifiers and further to evaluate their performance. The performance estimates of Naive Bayes and C4.5/J48 algorithm are given in Table 8.3 and Table 8.4 respectively, in which a full set of metrics are provided, including the mean and standard deviation of the success rate, the 80% and 90% level confidence intervals of the success rate, and so on.
To see which method is better, we next compare the estimate of the two machine learning methods. We observe that Naive Bayes method always has lower estimate (success rate values in several measurements) than C4.5/J4.8. Moreover, the success rate of the classifier derived from decision tree method C4.5/J48 is consistently high (around 95%) with small variance, meaning that the role identification algorithm works well when testing on Eurecom dataset. We therefore use C4.5/J48 method's model for later application on LBNL traces.

8.4.2.2 The application of the algorithm on LBNL traces

The dataset from Eurecom and those from LBNL are not comparable as these two enterprise networks are of different scales. In order to use the classifier derived from Eurecom dataset to help identifying the roles in LBNL, data normalization is thus necessary, as illustrated in Figure 8.11. As five LBNL traces are captured in the

Table 8.3: Performance estimate of Naive Bayes, Eurecom

		1st trial	2nd trial	3rd trial	4th trial
Normalized	testset size	113	113	113	112
	success rate(%)	73.45	74.34	76.11	70.54
	mean (%)	73.61			
	std	0.02			
	80%-CI (%)	[67.83, 78.41]	[68.76, 79.22]	[70.62, 80.84]	[64.76, 75.72]
	90%-CI (%)	[66.10, 79.70]	[67.04, 80.49]	[68.92, 82.06]	[63.01, 77.09]

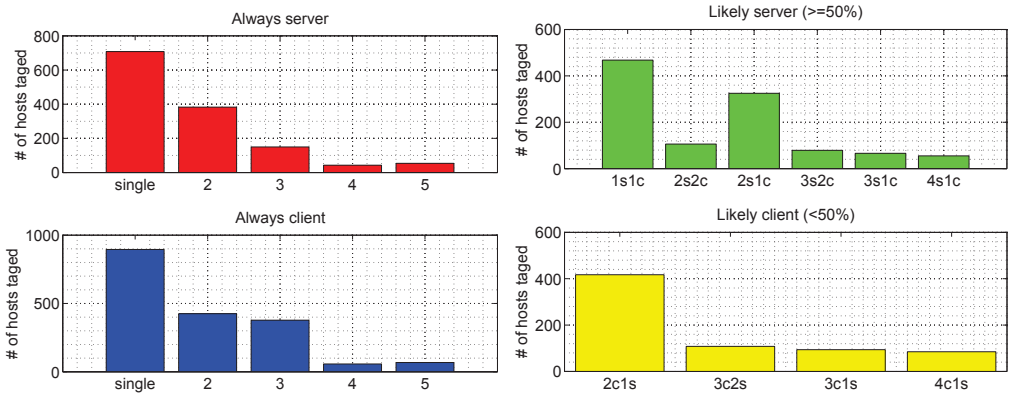
Table 8.4: Performance estimate of C4.5/J48, Eurecom

		1st trial	2nd trial	3rd trial	4th trial
Normalized	testset size	113	113	113	112
	success rate(%)	97.35	95.58	94.69	94.64
	mean (%)	95.56			
	std	0.01			
	80%-CI (%)	[94.63, 98.71]	[92.38, 97.47]	[91.30, 96.81]	[91.22, 96.78]
	90%-CI (%)	[93.53, 98.94]	[91.17, 97.83]	[90.04, 97.24]	[89.96, 97.21]

same lab close in time, we expect that a large amount of hosts (exclusively identified by the IP address) are commonly contained in these 5 traces.
We apply the model trained from Eurecom dataset to each LNBL dataset. We classify a host to be "*always server*" if this host is assigned as *a server* only in one or more datasets by the model, and a host is classified to be "*always client*" if this host is assigned as *a client* only in one or more datasets by the model. In addition, a host is put into the class of "*likely server*" in the case that this host is tagged as *a server* more times ($\geq$50%) than tagged as *a client* in those 5 datasets, otherwise classified as "*likely client*"($<$50%).
We report in Figure 8.12 the number of hosts finally identified in each class defined above, in which the number in the x axis for "always server" indicates the occurrence that a host is assigned as a server (a similar meaning for "always client"), and the term "3s2c" denotes that a host is tagged as *a server* 3 times and as *a client* 2 times. The role identification algorithm does not work for LBNL traces as we observe that only a fairly small number of hosts identified as a same role 3 to 5 times - meaning that the majority of the hosts are not consistently identified with the same role throughout the 5 datasets.

8.4.3 Per host traffic

For the above method to be applicable, we need to have enough traffic per host. However, the distribution of the traffic per host for Eurecom and LBNL traffic traces given in Figure 8.13 are significantly different, with less than 1MB for more than 80% of the hosts for LBNL traces. This can be the possible reason to explain the



(a) Always server/client (b) Likely server/client

Figure 8.12: Histogram, LBNL

failure of the algorithm on LBNL traces.

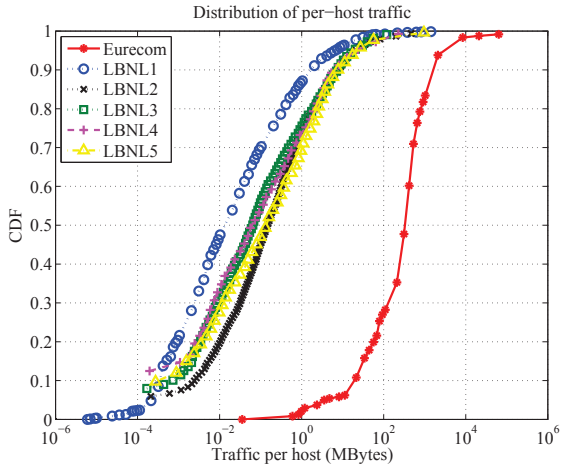


Figure 8.13: Distribution of traffic per host

8.5 Conclusion

We discuss in the chapter the enterprise traffic profiling. We have developed an understanding of the basic characteristics of modern enterprise traffic at various levels based on a medium-size laboratory packet trace (Eurecom). The significant contribution is to contrast the external and internal activity in modern enterprise networks - around 90% of the traffic never leaves the enterprise while the other 10% represents the data conveyed by the transfers between local hosts and remote peers outside the enterprise; data flows symmetrically in two directions (upload and download) for intranet traffic while uploads account for merely 10% of the Internet

traffic and the left 90% is carried by downloads. These findings can help to develop a workload model for typical enterprise networks, which is going to be addressed in next chapter. As an additional issue, a supervised machine learning approach is proposed to find an automatic way to identify different roles (servers or clients) inside the enterprise networks. The test of this algorithm on Eurecom trace is quite successful but fail on LBNL traces as we demonstrate that the traffic per host for LBNL traces are fairly small.

Chapter 9

Workload Model for Enterprise Networks

9.1 Introduction

Recently, enterprise networks have received more and more attention from the research community. Nowadays, the complexity of enterprise network is ever increasing as many different access methods (wired and wireless) are simultaneously deployed in the Intranet, and meanwhile large amount of newly emerging applications (for instance, video streaming) are put into use in modern enterprise. To the best of our knowledge, several aspects of the enterprise networks are still unexplored after the seminal step study of enterprise traffic by Ruoming Pang *et al.* [43] and the subsequent studies from other researchers [21, 40, 12]. We assessed the basic characteristics of enterprise traffic relying on a realistic medium-size enterprise trace (Eurecom) in Chapter 8, in particular we contrasted intranet and Internet activities related to the hosts inside the enterprise.

Our study of the enterprise traffic manifests several interesting aspects over the traffic flowing in an enterprise environment - around 90% of the traffic never leaves the enterprise while the other 10% represents the data conveyed by the transfers between local hosts and remote peers outside the enterprise; data flows symmetrically in two directions (upload and download) for intranet traffic while uploads account for merely 10% of the Internet traffic and the left 90% is carried by downloads. These new findings have never been reported in the previous work in the literature to the best of our knowledge, and can be helpful in developping a new workload model for modern enterprise networks. A workload model of the enterprise network based on these findings and subsequent assessments on enterprise traffic activities (for instance, RTT estimation), is proposed in a first stage in this chapter.

The objective of this chapter is to develop a workload model for modern enterprise networks. As illustrated in [23], the application can significantly affects the performance of data transfers, in particular the performance of short transfers in a variety of way. In addition to our former study which helps to understand how traffic flows in an enterprise structure, we also investigate the impact of the application on top. We then incorporate this effect to the development of enterprise workload model, and evaluate the model by replaying the traffic in the simulations.

9.2 Traffic Profiling: another perspective

In this section, we firstly represent in brief the methodology initially proposed in [23] with the purpose of investigating the effect introduced by the applications on the transfer time of short TCP transfers, and further in [24] for detecting TCP anomalies. We then profile the effect of the applications in an enterprise environment by adopting this methodology on a realistic enterprise packet trace (Eurecom), and further use the observations for workload modeling in a later section .

9.2.1 Methodology description

In general, a complete transfer can be decomposed as three phases: set-up, data transfer and tear-down. The set-up phase in most case corresponds to a complete three-way handshake for a TCP transfer, consisting of three control packets (SYN-SYN/ACK-ACK). The set-up time is thus counted as the time between the first control packet and the first data packet. The data transfer phase refers to the duration between the first and the last data packets transferred in a connection, including the data packet retransmission if any. The tear-down phase, in which at least one control packet with FIN or RST flag is observed for a complete transfer, is the duration between the last data packet and the last control packet in a connection. We exclude the set-up and the tear-down phases in our analysis for simplicity, focusing on the data transfer phase only. In our discussion, a train (or a block) is define as a sequence of successive data packets flowing one after another with the same direction from one party to the other, before the direction is shift. We term A (or client) and B (or server) respectively the two parties involved in a transfer, in which A (or client) is the initiator of a transfer, and B is the remote party of a transfer in our representation. The methodology presented in [23] introduces three time components for the phase of data transfer, all of which summing up to the data transfer time of a transfer:

- The client (or server) **warm up times** - after receiving the last data packet of a train from the other party, the time a client (or a server) spends before it begins sending the first data packet for the new train. It can be for instance the thinking time on a client side or the data preparation time on a server side.

- The **theoretical time** for data transfers on the client or the server side - the time an ideal transfer takes over the same path with the same amount of data packets to transfer, with the assumption of the same RTT value for all data packets. The way to compute is as follows: we record the total number of distinct data packets sent by A or B. We next compute the duration that an ideal TCP layer with an initial congestion of 1, delayed acknowledgment turned on, an infinite capacity, an RTT equal to RTT_{A-B} and the same number of packets to send as A or B would take to complete the transmission of all those packets. We term those duration $T_{theory}(A)$ and $T_{theory}(B)$.

- **Pacing** on the client or server side - the time difference between the actual transfer time on one side and the sum of the warm-up times and the theoretical time on the same side, reflecting the additional constrains added by the applications or others.

Figure 9.1 depicts a set of components of a typical transfer. Note that the methodology described above is application agnostic. When replaying the traffic to the simulations presented later, we normalize the pacing by dividing the total pacing time per direction by the total number of data packets transferred in the corresponding direction.

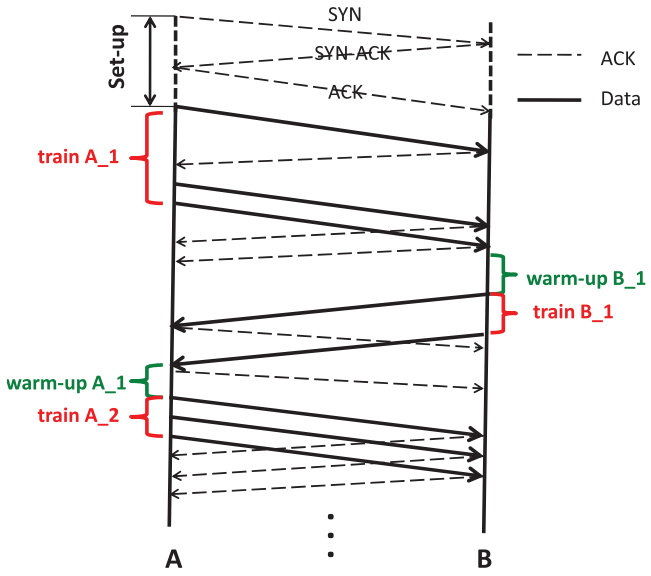
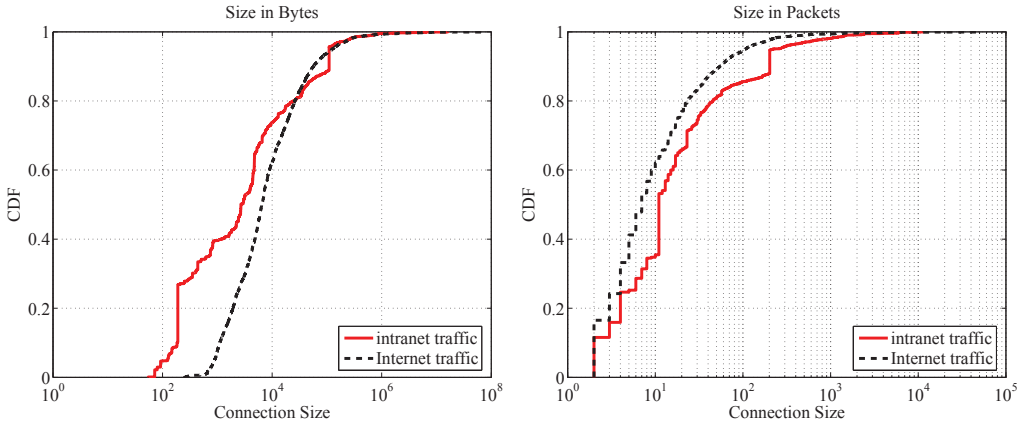


Figure 9.1: Decomposition of a typical TCP transfer

9.2.2 Traffic profiling

We restrict our analysis on complete connections. A well-behaved connection is defined as the connection with a complete three-way handshake, which transfers at least one data packet in each direction, and is finally teared down with at least one control packet with a FIN or RST flag seen. We pick 3000 intranet well-behaved connections, as well as 3000 Internet well-behaved connections from the full-day Eurecom trace for the study.
Remind that, one connection is called "intranet connection" if the other peer involved is also a local host - meaning that data transfers never leave the enterprise, otherwise called "Internet connection" if the other peer involved is outside the enterprise, in the sense that data flows across the enterprise boundary. These definitions are in line with the ones in Chapter 8.

Figure 9.2 depicts the cumulative distribution of well-behaved transfer size using bytes and packets for intranet and Internet traffic respectively. We observe that intranet connections generally consist of more packets than Internet connections while carrying less bytes, meaning that the packet sizes from intranet traffic are relatively smaller. In addition, the distribution of volume (in packets) ratio in two directions for each connection is given in Figure 9.3(a), as well as the distribution of average RTT estimation for each connection shown in Figure 9.3(b). We observe that data in most of the intranet connections (more than 80%) flow in two directions in a regular way with a consistent ratio of 1, whereas the ratio for Internet connections grows up to a value of 10000, with a noticeable variance. This observation highlights again the symmetric feature of intranet traffic, as presented in Chapter 8 Section 8.3.3.

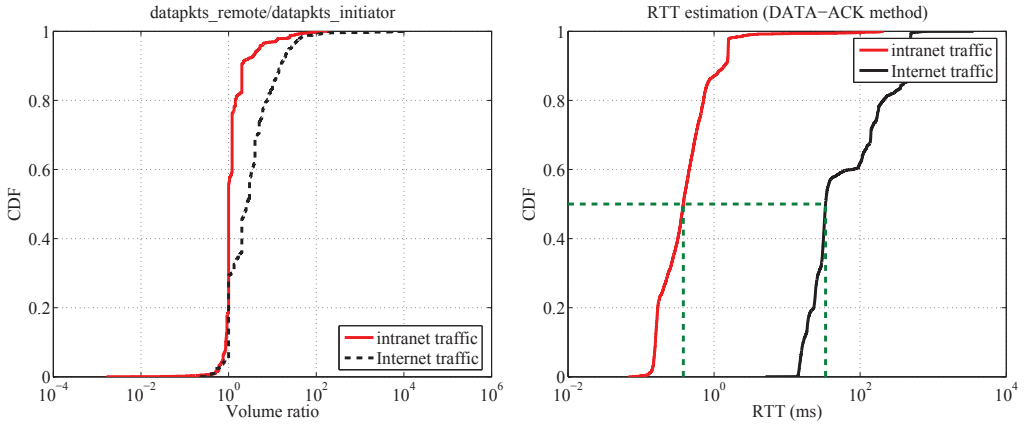


(a) Size in Bytes (b) Size in Packets

Figure 9.2: Distribution of traffic volume

The RTTs shown in figure are estimated with the DATA-ACK method, rather than using SYN-SYN/ACK-ACK approach. We first observe that the RTTs for intranet and Internet traffic have similar shapes, but with strikingly different magnitudes. The RTTs of the intranet connections are clearly smaller than the ones of the Internet connections, with a median of 0.38 ms and 33.6 ms for the former and the later case respectively. RTTs observed with Internet traffic is in line with previous measurement studies.

We next report the distribution of warm-up times of the two parties involved in a connection for intranet and Internet traffic respectively in Figure 9.4. We observe that the warm-up times of the two parties in intranet connections are similarly distributed, in which the warm-up times of the initiator are slightly larger than the ones of the remote party. This observation implies that the two parties inside the enterprise have a similar behavior in an intranet transfer, without taking a strict role of client or server. In contrast, the distribution of warm-up times of the two parties in Internet connections are apparently different. As a large portion (around 60%) of Internet initiator's warm-up times are less than 0.1ms while more than 80% of

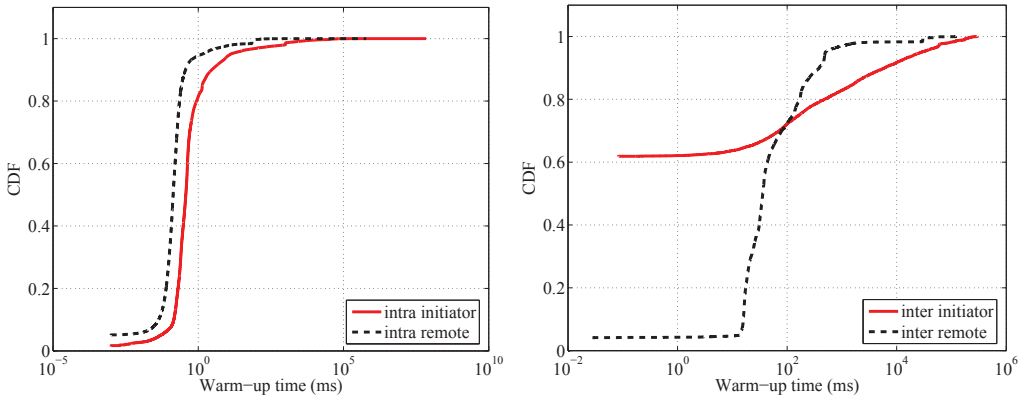


(a) Distribution of down-up ratio

(b) RTT

Figure 9.3: Distribution of down-up ratio and RTT

Internet remote party's warm-up times are larger than 10ms, we thus believe that as one of the indicators of the impact of the applications, the warm-up times are likely dominated by the remote party (likely remote servers) for the Internet connections.



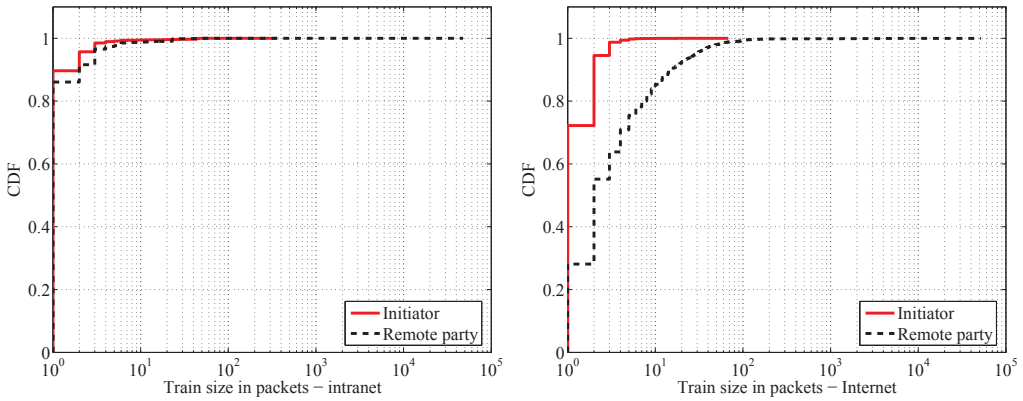
(a) Warm-up time, intranet

(b) Warm-up times, Internet

Figure 9.4: Distribution of warm-up times

We present the distribution of train sizes of the two parties (initiator and remote party) for intranet and Internet traffic respectively in Figure 9.5. We distinguished between initiator and remote party when examining the train sizes as we expect that the remote party acts more like a server sending large amount of packets in each train/block. This hypothesis holds for Internet traffic as we do observe from Figure 9.5(b) that train sizes sent by the remote party is significantly larger than those sent by the initiator. Moreover, more than 95% of the initiator trains have a size of less than 3 data packets. However, the train sizes sent by the two parties of the intranet transfers are consistently small - more than 80% of the initiator (or remote party) train sizes are less than 2 data packets. This may be due to the fact

that most of the intranet transfers are simply sequences of request-response pairs with few packets only (likely one packet) for each direction.



(a) train size, intranet

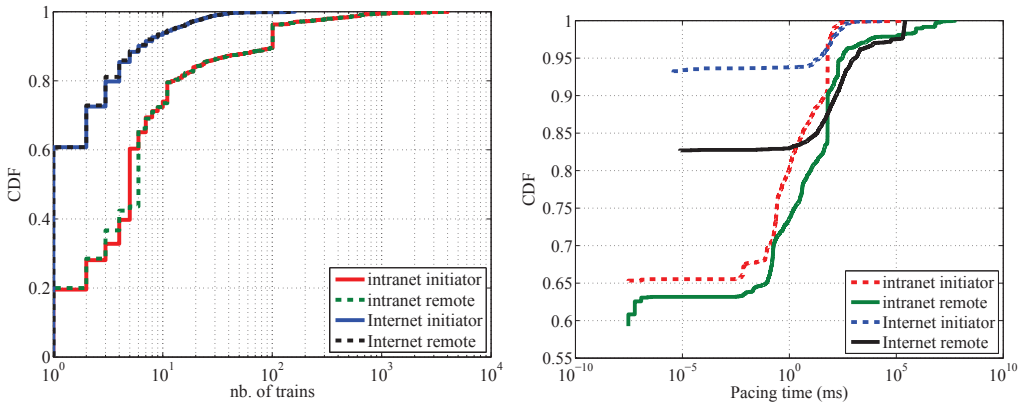
(b) train size, Internet

Figure 9.5: Distribution of train sizes

At the same time, we also assessed the distribution of the number of trains for intranet and Internet transfers respectively, shown in Figure 9.6(a). Nearly the same distributions can be observed for the initiator and the remote party, either for intranet or Internet transfers. This is simply due to the way we define a train/bloc and the fact that client/server applications are usually request-response style. More interestingly, intranet transfers tend to be broken into more trains as compared to the Internet transfers, meaning that an intranet transfer likely consists of more request-response pairs than a Internet transfer. We finally examine in Figure 9.6(b) the distribution of the pacing time, introduced by the applications on top. If we simply distinguish between the intranet and the Internet transfers, we can observe that the majority of the pacing times (more than 80%) in Internet transfers are pretty small, while a considerable portion of the intranet pacing times (around 30%) are significantly large ranging from 1ms to 1000s. We therefore conclude that intranet transfers are more likely to be affected by the applications that introduces large pacing time during the transfer.

9.3 Workload modeling for Enterprise Networks

It is commonly believed in the research community that [14, 60, 3], developing an appropriate model – which in general consists of a set of parameters, including network configurations, workload generation rules, etc. – can help facilitating the evaluation of new proposals (protocols, algorithms, etc.), in order to compare and contrast themselves using a predefined framework. Each component is required to be carefully designed through extensive studies of real cases. A validated model can then be used to predict the behavior of new proposals in real networks. To generalize a model which is applicable to all cases might be fairly difficult as the



(a) number of trains (b) pacing time

Figure 9.6: Distribution of nb. of trains and the pacing time

observation of different network environments varies quite a lot from case to case in reality.

In this section, we devise two new workload models for an enterprise environment that enable to shed a new light on the performance of scheduling policies in typical enterprise networks - based on our findings on enterprise traffic study in Chapter 8 and in Section 9.2 of this chapter. To exemplify those models, we use a subset of the scheduling disciplines discussed in the previous part, namely FIFO, SCFQ, Run2C, LAS and EFD. These two models (termed as "*the topological model*" and "*the apps model*" respectively) are separately designed with different objectives: one model emphasizes the effect of the enterprise topological structure – in which local hosts contact intranet and Internet servers simultaneously with diverse RTT ranges and traffic flows in a regular pattern, while the other model incorporates the impact of the applications – which in practice alter the flow of packets through the interaction with TCP. Throughout this section, we call "*the legacy model*" the general model without one of the additional features presented above.

We present and evaluate our models using a dumbbell topology which has been widely deployed in the research community. The wired dumbbell topology is given in Figure 9.7, in which two groups of hosts (group of clients 1 to 10, and group of servers 13 to 22) are connected to two routers (nodes 11 and 12) by a link each with a bandwidth of 100Mbit/s, while two routers in the middle are connected by a 10Mbit/s link. The intermediate link is therefore the bottleneck link with the settings given above.

As a variant of the dumbbell topology adapted to wireless network, local clients simply change the access method from wired line to wireless channel through a single access point. The wireless topology is depicted in Figure 9.8. We use the 802.11a protocol with nominal bit rate of 54Mbit/s, with RTS/CTS disabled. Good and fair radio transmission conditions are ideally guaranteed. The 10 wired servers are connected to a router with an output rate 10 times larger than its input rate

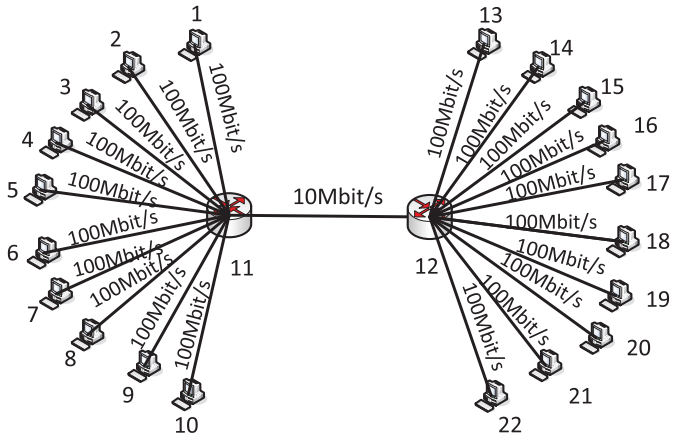


Figure 9.7: Wired network topology

(suppose that traffic flows from right to left), so that its output queue never builds up. With such a configuration, the bottleneck if any, is the access point.

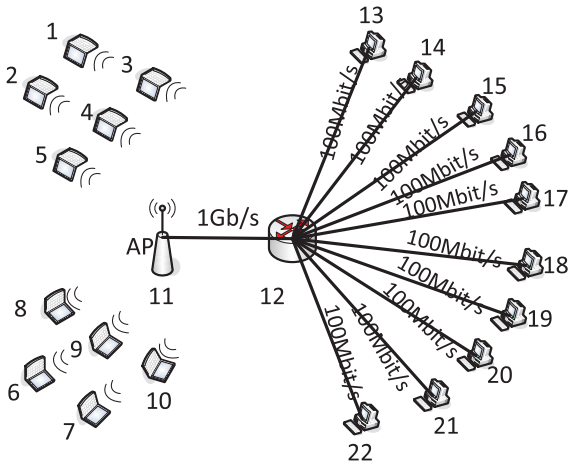


Figure 9.8: Wireless network topology

In all cases without special declaration, nodes 13 to 22 simply represent general servers. In topological model, nodes 13 to 17 denote the group of intranet servers, while nodes 18 to 22 represent the group of Internet servers.

9.3.1 The topological model

We factored all the findings from enterprise traffic profiling such as RTT discrepancy between intranet and Internet traffic in our topological model, and evaluate this model's effect on the performance of the scheduling policies for the cases of both wired and wireless networks. The workload used here consists of Poisson arrivals with heavy tail flow size distribution.

Based on the study of the enterprise traffic in Section 9.2.2, we divide the whole traffic into two parts: intranet traffic accounting for 90% and Internet traffic for 10%. The intranet traffic is equally shared in two directions while the downloads and the uploads represent respectively 90% and 10% of the Internet traffic. Table 9.1 summarizes the above description. For the legacy model, the traffic is divided into downloads and uploads, with fractions of 54% and 46% respectively, which is equivalent to the case of the topological model.

Table 9.1: The way traffic flows in the enterprise

Wired Network/Wireless Network					
the legacy model		the topological model			
download	upload	intranet traffic		Internet traffic	
		download	upload	download	upload
0.54	0.46	0.45	0.45	0.09	0.01

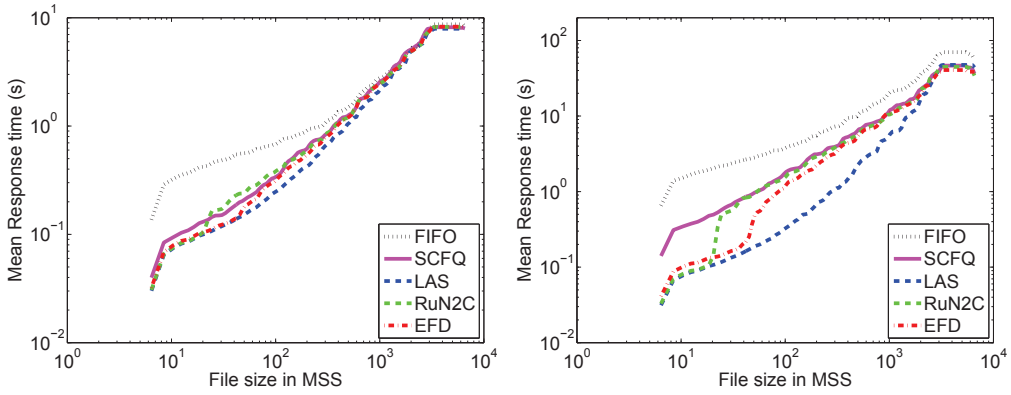
In addition, we choose 1 ms and 100 ms respectively as the average RTT of the intranet and Internet transfers in our evaluation. To produce such RTTs, the one-way propagation delay of each (physical) link for wired network are given in Table 9.2. In contrast, we take the weighted average value of RTTs for the legacy model, that is $1ms * 0.9 + 100ms * 0.1 = 10.9ms$ - therefore it is 5.45 ms for the one way delay of each path from client to server (or in the reverse direction). The setting for the wired part of the wireless network is similar to the one given in Table 9.2 for wired network, except that the delay for the wireless link is set as the empirical value given by the QualNet simulator by default.

Table 9.2: Parameter setting - link delay

Wired Network			
the legacy model		the topological model	
physical link	delay(ms)	physical link	delay(ms)
x-11, 11-12	0	x-11, 11-12	0
12-y	5.45	12-w	0.5
		12-v	50
x = {1, 2, ..., 10}, y = {13, 14, ..., 22} w = {13, 14, ..., 17}, v = {18, 19, ..., 22}			

Due to the fact that different RTT ranges are observed for the intranet and Internet transfers in the traffic study and two different values (1ms and 100ms) are assigned to these two types of traffic in our evaluation, we use two terms “low latency traffic” and “high latency traffic”, interchangeably to “intranet traffic” and “Internet traffic”

respectively in this section.
The simulation results told us that, the overall performance of flows with varying size in the two models (the legacy and the topological models) are consistently similar to each other. We then discriminate between low and high latency traffic for the topological model. We report the performance of scheduling policies in terms of mean response time for the two models respectively in Figure 9.9 and Figure 9.10, in which the buffer size is set to 300 MSS. For the case of the topological model, we observe that the low latency traffic provides a performance that is quite close to that of the overall traffic, which evidences the dominant role of the intranet traffic (as it represents the majority of the traffic). In addition, no apparent difference is observed among scheduling policies for the high latency traffic for underload case. The likely reason lies in the fact that the response time of high latency traffic is dominated by the delay over the long path, while it is similar as the queueing time for low latency traffic. Therefore, controlling the packets' behavior in the queue as what scheduling policies usually do, does not lead to the change on the response time of high latency traffic. The discrepancy on the performance of scheduling policies becomes pronounced when the bottleneck link is congested (*i.e.* in overload case), in particular for high latency traffic – due to the packet losses experienced which may significantly affect the response time of the flows.

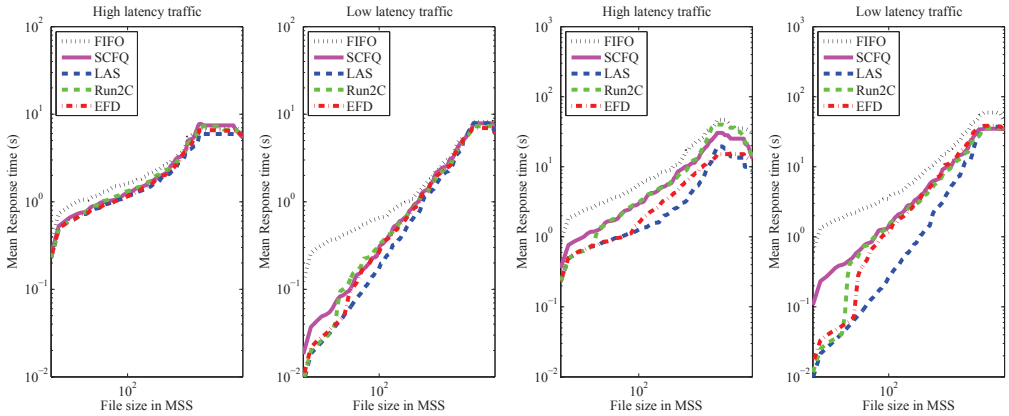


(a) workload of 8Mbit/s (underload)

(b) workload of 15Mbit/s (overload)

Figure 9.9: Mean response time - the legacy model - 300MSS - wired network

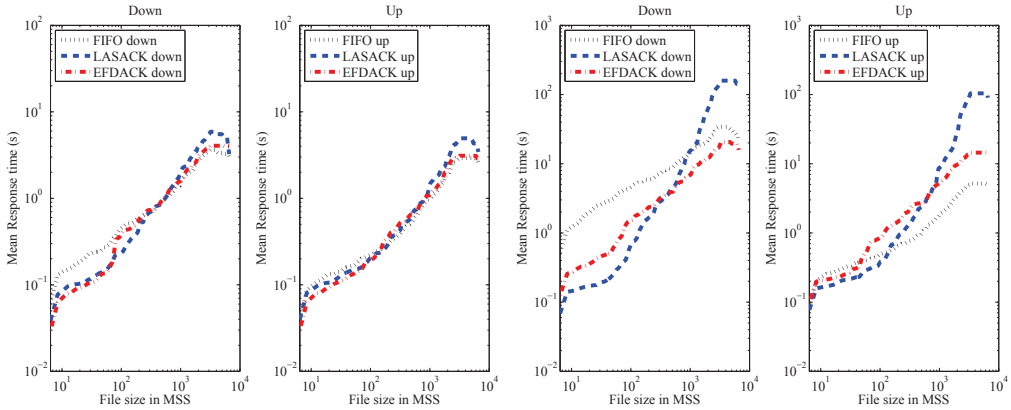
We next examine the impact imposed by the topological model to the wireless network for the case in which the access point buffer size is 30 MSS. As similar performances is globally observed in the comparison between the legacy model and the topological model, we report in Figure 9.11 the result for the legacy model only. When digging into the two types of traffic for the topological model as before – see Figure 9.12, the discrepancy among scheduling policies for high latency traffic starts to emerge even when the load is moderate, and becomes stronger for the case of high load, and finally significantly contributes to the global performance, meaning that the low latency traffic no longer takes the absolute dominance. The likely reason lies in the fact that the small access point queue is prone to build up, resulting in



(a) workload of 8Mbit/s (underload) (b) workload of 15Mbit/s (overload)

Figure 9.10: Mean response time - the topological model - 300MSS - wired network

consistent packet losses even when the load is moderate.

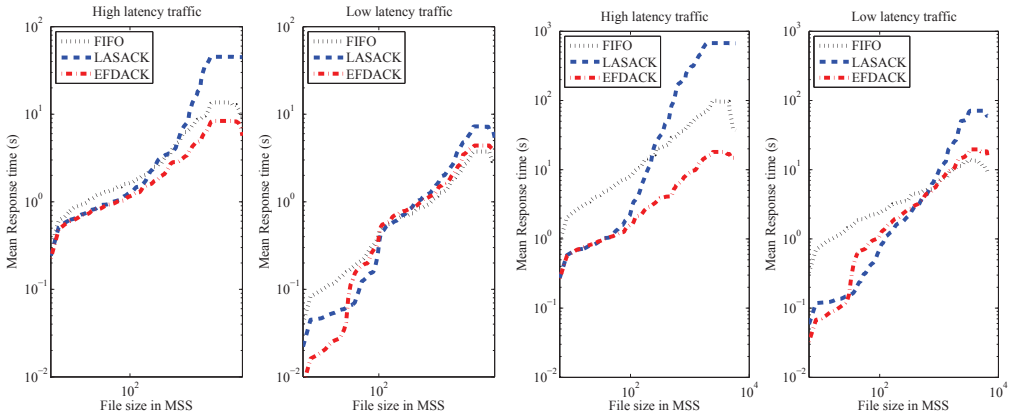


(a) workload of 10Mbit/s (medium load) (b) workload of 20Mbit/s (high load)

Figure 9.11: Mean response time - the legacy model - 30MSS - Wireless LANs

9.3.2 The apps model

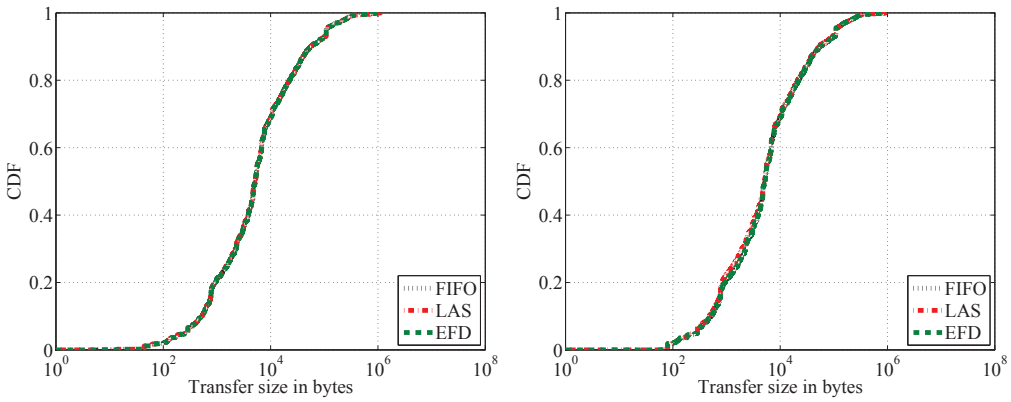
To understand how the application on top controls the packet behaviors and eventually affects the network performance in a real enterprise network, we study the Eurecom traffic trace in several aspects in Section 9.2 based on a break down methodology originally presented in [23]. We factor all these findings to in our workload modeling, and use it to assess the impact of the applications on top. The model developed is called "the apps model", to differentiate from the legacy model.
We use the traffic protocol called "TRAFFIC-TRACE" in QualNet – a UDP-based traffic generating application – to reproduce the traffic, which reflects exactly the traffic behavior in Eurecom by incorporating several factors such as flow size, packet



(a) workload of 10Mbit/s (medium load) (b) workload of 20Mbit/s (high load)

Figure 9.12: Mean response time - the topological model - 30MSS - Wireless LANs

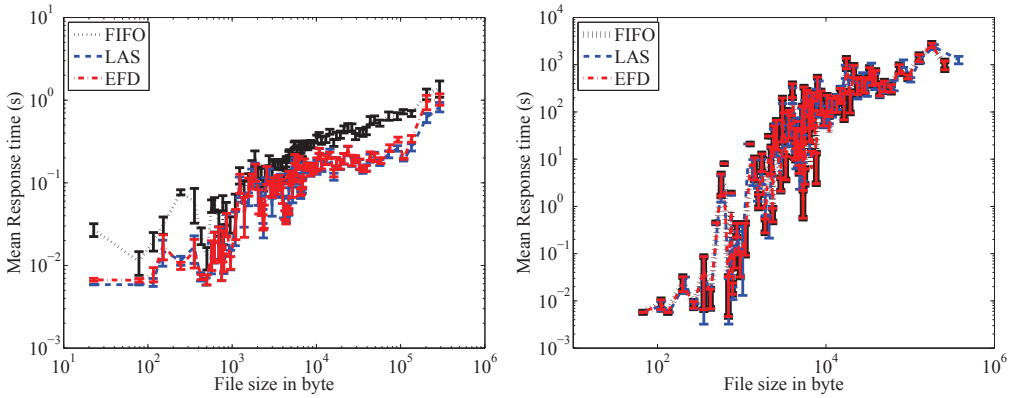
size and packet inter-arrival time. The generated traffic is then used to feed the apps model. The ideal case would be to replay the traffic in a TCP manner with a appropriate model for the pacing and warm up times (at the application level, when it gives packets to TCP). However, due to time constrains, we leave it to the future work. In contrast, we keep the same flow size information as the apps model, where each flow corresponds to a TCP flow using "FTP/GENERIC" to generate traffic for the legacy model. The evaluations are conducted in a wired dumbbell topology the same as the one shown in Figure 9.7. We focus our analysis on those transfers completed without any packet loss. The size distribution of those transfers for the two models are shown in Figure 9.13. They are the same as the load considered is low. Note again that this is the distribution of completed flows. If ever the load was higher, we could observe discrepancies.



(a) the legacy model (b) the apps model

Figure 9.13: Transfer sizes distribution - workload of 8Mbit/s - 300MSS - wired network

The performance of the scheduling policies under the two models is given in Figure 9.14. Apparently, the results indicate that taking into account the impact of the application tends to blur the differences between the various scheduling policies. However, this observation can not lead to a general conclusion as we have only scratched the surface of things since TCP was not taken into account in our evaluation.



(a) the legacy model

(b) the apps model

Figure 9.14: Mean response time with CI - workload of 8Mbit/s - 300MSS - wired networks

9.3.3 Discussion

We have proposed two new models to more realistically assess the impact of scheduling disciplines in the case of a typical mid-size network like the one of Eurecom. Due to time constraints (the end of the thesis) we only presented preliminary results. However, those results already indicate the relative impact of scheduling policies (since we took scheduling disciplines as an example of mechanism here - but the approach could be generalized to other mechanisms). For the the case of the topological model, we observed that the scheduling disciplines mostly impact the intranet flows and not so much the Internet flows, because of the different order of magnitudes of the latency and queuing times. For the apps model, the preliminary results suggest that the application on top of the transport layer can greatly lower the impact of the scheduling disciplines as it diminishes the burstiness of the traffic, hence the size of the buffer, which limits de facto the impact of the scheduling disciplines (in the extreme case, the queue would be consistently empty, hence all scheduling policies would behave similarly). We do agree that we need more work to better understand the relation between the topology, the application behavior and the extent of the impact of a QoS mechanism like a scheduling discipline, but we believe that our first results show that the approach is worth being pursued. The next natural step will be to develop a TCP model of a source that incorporates the impact of the application

(pacing and warm-up times) and also mixes the topological and the apps model.

Part IV

Conclusions and Future Perspectives

Chapter 10

Conclusions and Future Perspectives

With its successful evolution over the years, the Internet has become mandatory in our daily life and work, making information sharing around the world easier and faster. There has been an immense amount of effort on various aspects of the Internet, which gradually pushes forward the development of the Internet. Still, there are many issues that lack of study or remain unsolved in this wild domain. We discuss in this thesis, QoS solutions to improve the user perceived performance in terms of delay, loss rate, *etc.*, as well as the methodology for measuring enterprise networks and further for its performance improvement.

This thesis contains original work in several fields such as size-based scheduling in wired network and wireless LANs, enterprise traffic profiling, and enterprise workload modeling. In what follows, we present a summary of the results obtained in this thesis.

As a solution to the QoS problem in the Internet, Early Flow Discard (EFD) is proposed in this thesis, motivated by the observed high variability property of flow sizes and the expectation of eliminating the drawbacks associated with existing size-based schedulers. We showed the effectiveness of the proposed algorithm (EFD) when deployed in bottleneck nodes to service flows in wired network with single bottleneck link. Extensive simulations reveal that, EFD retains the most desirable property of more resource intensive size-based methods, namely low response time for short flows, while limiting lock-outs of large flows and effectively protecting low/medium rate multimedia transfers. In particular, EFD is able to significantly decrease the overhead of flow state keeping by one order of magnitude compared to full flow state keeping methods, like LAS. In addition, EFD is easy to implement in practice. In the performance evaluation of EFD, we examined several aspects, such as the average conditional response time, the overhead of flow state keeping, the starvation of long transfers, the impact to multimedia transfers, etc, and compare its performance to a wide range of other scheduling policies (FIFO, SCFQ, LAS, Run2C and LARS). As further work, one could investigate the performance of EFD discipline in terms of other metrics, *e.g.* slowdown, application-based loss rate, etc, and studying the performance of EFD in congested heterogeneous networks such as networks that simultaneously support UDP and TCP applications, TCP networks with heterogeneous propagation delay, and network with multiple congested links.

We tried to model EFD analytically to explain the simulation results. The difficulty lies in the flow fragmentation phenomenon which is hard to described in a mathe-

matical form, resulting from the flow state keeping mechanism introduced in EFD. Our study showed that subflow-level analysis can not help explaining the flow-level results. Although the gain is limited, we are able to explain the simulation results for flows with size below the threshold *th* with a model newly developed for shifting from subflows to flows. Our model is able to deduce the discrepancy between EFD's variants in terms of average response time seen in the simulations. As a future work, deriving a complete analytical model for EFD is worthwhile.

We studied the possibility of applying EFD to 802.11 Wireless LANs. The challenge comes from several aspects: on one hand, the access point downlink queue naturally builds up in infrastructure 802.11 networks; on the other hand, EFD needs to take into account bi-directional traffic even though EFD applies to downlink buffer only. Our analysis of EFD and its adaptations to 802.11 WLANs showed that, EFD's two adaptations – keep track of the volumes exchanged in both directions or simply count packets in a single direction - are effective in enforcing a good level of fairness, and at the same time are able to grab the full benefit of size-based scheduling. We also investigated the impact of the buffer granularity (in bytes or in packets) on the performance of scheduling policies over 802.11 WLANs. We conclude that measuring the buffer with the unit of bytes is highly preferred for FIFO, Run2CACK and BEFD, while LASACK, LARS and SCFQ are insensitive to the buffer granularity. Note that, one of the difficulty of the deployment of EFD, and many other size-based scheduling policies in the Internet in practice, is to identify the bottlenecks in the Internet, which strongly relys on the network tomography of the Internet.

We finally present our traffic analysis of enterprise network based on a realistic traffic trace of medium size research lab (Eurecom). We observed that intranet traffic account for the majority of the traffic load in an enterprise network (nearly 90%), while Intranet traffic takes a small fraction left (10%). In addition, the different RTT magnitudes of intranet and Internet traffic – with a median of around 1ms and 100ms respectively – are key differences between those two types of traffic. When focusing on the two direction transfers, we found that traffic in two directions flow symmetrically for intranet traffic while an asymmetry with ratio of around 10 was observed for Internet traffic. We factored all these findings in our workload modeling. Furthermore, we also assessed the impact of the applications on top by replaying the workload extracted from the real trace in the simulations. The results told us that the impact of the application on top may significantly affect the performance of the scheduling policies, therefore it should be considered as an important factor when one is designing a scheduler for a QoS solution.

As future work in this direction, it seems important to put more effort in devising new workload model that would enable to predict more accurately the performance of QoS solutions in specific environment, esp. enterprise networks. It seems also crucial to move "out of the lab" and put some effort into implementing our solutions, *e.g.*, using the Click Modular router, and perform some experiment in a real environment. This could be achieved with not too much effort by replacing one access point in an enterprise by our modified AP and measure the performance experienced by clients, provided that the WLAN is used not only to access the Internet but also

to access internal services within the company.

Résumé

1 Introduction

1.1 Internet et le protocole TCP/IP

L'histoire de l'Internet commence à partir ARPANET, un réseau de données expérimental construit dans les années 1960 par le Département américain de la Défense, reliant les universités américaines et la communauté de recherche d'entreprise pour l'échange d'informations. Il a été con?u à l'origine avec la capacité de délivrer individuellement des paquets de source à la destination à travers le réseau. Les protocoles TCP/IP développé plus tard, a permis d'interconnecter des réseaux différents dans le monde, en fournissant un service universel. Aujourd'hui, un ensemble de réseaux interconnectés à travers le monde est connue sous le nom d'Internet.
Le soi-disant Transport Control Protocol (TCP) est basée sur deux principes: la réglementation et la reconnaissance, seminally établies par Cerf et Khan [4] en 1974. Plus tard, Jacobson [13] a ajouté plusieurs caractéristiques importantes et mis TCP très proche de ce qu'elle ressemble aujourd'hui. TCP (une description formelle est donnée dans [24]) est un protocole fiable, un protocole orienté connexion, qui permet de transmettre l'information d'une machine à une autre dans le réseau sans erreurs. Comme le protocole dominant dans l'Internet, le protocole TCP / IP a continué à évoluer au fil des ans pour répondre aux besoins croissants de l'Internet et des petits réseaux privés.
L'Internet est capable de fournir une infrastructure générale, à laquelle une large gamme d'applications peuvent bien fonctionner, y compris la navigation Web, e-mail, transfert de fichiers, accès à distance, et ainsi de suite. La capacité à soutenir une gamme d'applications est essentielle, mais le défi devient plus sévère et plus sévère que les applications de plus en plus avec les nouveaux besoins sont déployés dans l'Internet, comme le streaming multimédia, peer-to-peer (P2P), etc. L'ensemble des applications qui dominent l'Internet a changé à partir de HTTP et FTP pour les applications P2P, et plus récemment, le streaming HTTP.
L'infrastructure IP est en principe destiné à faire de son mieux pour livrer des paquets, sans fournir aucune garantie pour le service qu'un paquet recevra. Dans un réseau comme ca, tous les utilisateurs de bénéficier du service "best effort". Quand un lien est congestionné, les paquets sont rejetés parce que la file d'attente est débordé. En cas d'événement perte, la retransmission des paquets perdus est assurée par le protocole TCP. Bien que ce service "best effort" bien fonctionne pour certaines applications, il ne peut pas satisfaire les besoins de nombreuses applications nouvelles qui sont sensibles à la perte de paquets et la latence grande, par

exemple, le streaming multimédia populaires dans notre vie quotidienne. Nouvelle architecture pour l'allocation des ressources est donc nécessaire pour l'Internet pour soutenir l'assurance des ressources et différents niveaux de qualité de service (QoS).

1.2 Motivation de la thèse

Dans l'évaluation d'Internet au cours des années, un grand nombre d'applications apparaissentémerger avec exigences diverses, telles que le temps de réponse faible, le taux de perte de garantie et taux de données. Cependant, l'Internet a limité la capacité de gestion des ressources à l'intérieur du réseau à partir du moment qu'il a été con?u et ne peut donner aucune garantie pour les utilisateurs finaux. Aujourd'hui, l'Internet soutient un service best-effort encore et le besoin de différenciation de service persiste. Pour cela, les chercheurs ont essayé de re- concevoir l'Internet, de sorte que les différents types d'application peuvent tre simultanément supporté avec des exigences de service minimales satisfaits. En conséquence, les différents mécanismes de QoS ont été proposées, avec un ensemble de protocoles de dicter le périphérique réseau pour servir des applications en lice en suivant un ensemble de règles prédéfinies. En général, les algorithmes d'ordonnancement de paquets, en collaboration avec les directions tampons, sont couramment utilisés pour gérer l'utilisation des ressources du réseau d'une manière efficace.
L'héritage FIFO / drop-tail schéma, déployé dans les routeurs / commutateurs d'aujourd'hui, est censé favoriser les transferts longs au niveau de flux, qui à l'inverse limite fortement la transmission des transferts courts - on voit la nécessité d' amélioration parce que les flux courts sont en général liés à des applications interactives telles que e-mail, navigation sur le Web et DNS requte / réponse. La question du partage des ressources dans les réseaux informatiques a été étudiée pendant des décennies et de nombreux algorithmes d'ordonnancement ont d'abord été développé dans le cadre de la planification des t?ches dans les systèmes d'exploitation. Ordonnancement de paquets a été réactivé dans le milieu de la recherche dans la dernière décennie en raison des études de distributions de taille d'emploi dans une variété de contextes, y compris la taille des fichiers sur le Web, les transferts de FTP fichiers, UNIX tailles emploi, et plus encore. Dans tous ces cas, la distribution de la taille d'emploi a été démontré à présentent la fonction de lourde tail, et tre bien modélisée par une distribution de Pareto, ou d'autres distributions avec une queue en loi de puissance. Cette découverte nouvelle appels à la réévaluation des politiques d'ordonnancement avec la charge de la queue lourde dans l'Internet, en particulier pour les politiques d'ordonnancement fondés sur la taille.
Motivé par la propriété de haute variabilité du trafic Internet, un certain nombre de politiques d'ordonnancement fondés sur la taille ont été proposées. The Shortest Remaining Processing Time (SRPT) est connu pour tre optimal [27], dans le sens o elle minimise le temps moyen de réponse des transferts. Bien que séduisante, SRPT n'est pas pratique parce que il reuquires la connaissance de la taille des flux - ce qui n'est pas réalisable pour la plupart des appareils du réseau (point d'accès routeur,

etc.). Par conséquent, une plus grande attention est accordée aux aveugles politiques d'ordonnancement fondés sur la taille, *i.e.* des politiques d'ordonnancement qui ne sont pas conscients de la taille des flux. Pour résoudre ce problème, plusieurs méthodes séminales ont été proposées, *i.e.* LAS [25], Run2C [2], et LARS [12]. En dépit de leur caractéristique unique – de donner un temps de réponse basse pour les petits flux – les principales raisons qui empchent ces approches d'ordonnancement fondés sur la taille du déploiement sont liés aux préoccupations suivantes: complexes de maintien de l'état de flux, de la famine des flux de long, prenant en compte que le montant cumulé des octets de chaque flux mais pas le taux, et ainsi de suite. Un des buts de cette thèse est à la recherche d'une solution politique d'ordonnancement qui peuvent tre utilisés dans les réseaux filaires, afin de améliorer la performance globale de l'utilisateur per?ue, par la fa?on de favoriser les flux courts sans les inconvénients habituels qui est associée à la taille de base d'ordonnancement. Nous également essayons de résoudre le problème injustice pour TCP indiqué dans 802.11 Wireless LAN, à l'aide des disciplines d'ordonnancement à la couche réseau, maintenir le protocole de couche inférieure (MAC couche) inchangée.
Une autre motivation derrière cette thèse est lié aux réseaux d'entreprise. Aujourd'hui, les réseaux d'entreprise ont évolué à partir du site centrées sur les réseaux c?blés o les utilisateurs d'accéder aux serveurs d'applications gr?ce à une infrastructure fixe, au cas o les utilisateurs sont en itinérance, soit à partir d'un réseau filaire à un réseau sans fil ou à partir de l'intérieur de l'entreprise à l'extérieur par un accès VPN. En outre, la variété sans cesse croissante des applications utilisées dans les intranets, par exemple, voix et vidéo sur IP, ainsi que la consolidation des serveurs gr?ce à la virtualisation et de données à travers SAN (Storage Area Networks), les deux étant éventuellement intégrées à offrir des services hautement élastiques, ont considérablement accru la complexité des réseaux d'entreprise. Nous nous attendons à de nouvelles caractéristiques émergentes à travers l'étude de trafic de l'entreprise moderne. Un autre objectif de cette thèse est d'explorer les nouvelles fonctionnalités de trafic de l'entreprise et d'étudier l'impact des applications sur les performances de TCP, afin de aide à la modélisation de travail modélisation d'entreprise.

1.3 Contributions et Outline de thèse

Nous avons fait plusieurs contributions dans cette thèse. La première contribution est la proposition d'une nouvelle discipline d'ordonnancement basé sur la taille - Early Flow Discard (EFD), qui remplit simultanément plusieurs objectifs: (i) le bas temps de réponse plus à petits flux; (ii) le faible co?t de comptabilité, *i.e.* le nombre de flux reste toujours faible; (iii) En différenciant les flux en fonction des volumes, mais aussi sur la base de taux; (iv) éviter la famine des flux de long. EFD n'est pas limitée à une politique d'ordonnancement mais incorpore également une politique de gestion de tampon, o le paquet à la plus petite priorité se rebut quand la file d'attente est pleine, par opposition à drop-tail qui tombe aveuglément

les paquets à l'arrivée. Dans le chapitre **??**, nous évaluons la performance d'EFD en réseau filaire sous la distribution de taille de flux avec une propriété de haute variabilité en utilisant la propriété plusieurs mesures, et la comparer à d'autres politiques d'ordonnancement (LAS, Run2C et LARS - le FIFO héritage est constituée aussi pour la comparaison parce FIFO est la norme actuelle de facto). Nous considérons deux régimes de charge - sous charge et surcharge. En général, nous montrons par des simulations approfondies que EFD surpasse ou au moins obtient une performance similaire aux autres politiques séminales, avec l'avantage de réduction des co?ts significative sur le suivi de flux. En outre, nous démontrons en outré la capacité d'EFD de protéger efficacement les transferts de taux bas / moyen multimédias.

La deuxième contribution est le développement d'un modèle analytique pour EFD, ce qui aide à expliquer les résultats de simulation. Dans le chapitre **??**, nous examinons d'abord les modèles couramment utilisés pour FIFO, SCFQ, LAS et Run2C, et les utiliser pour valider nos résultats de simulation. Nous présentons puis la difficulté de dérivation d'un modèle analytique pour la discipline EFD, en creusant dans la relation entre les flux et les sous-flux fragmenté des flux d'origine. Comme point de départ, nous essayons d'expliquer les résultats des flux-niveau basés sur l'analyse des sous-flux-niveau, mais nous ne parvenons pas principalement à cause de deux raisons: d'une part, la distribution des tailles de sous-flux dans la file d'attente prioritaire est beaucoup moins inégale que la distribution des tailles de flux d'origine, mais pas exactement déterministe; d'autre part, le processus d'inter-arrivée des sous-flux en file d'attente haute priorité n'est plus processus de Poisson. Nous puis passons au problème de la relative performance de sous-flux-niveau à la performance de flux-niveau en EFD. Nous enfin développons un modèle qui est capable de lier avec succès entre les flux et les sous-flux, et re-produire les résultats de la simulation d'une manière analytique.

La troisième contribution de cette thèse est l'analyse d'EFD, et son applicabilité dans un environnement de 802.11 Wireless LAN, o le problème injustice des TCP est résolu. Initialement, EFD était con?u dans un réseau c?blé et évaluées dans le cas des flux unidirectionnels. En revanche, les données coule dans les deux directions et les transferts de deux directions partagent le milieu (qui est "half duplex") dans les réseaux 802.11. En outre, la taille du tampon du point d'accès (AP) est généralement faible, donc il tend à construire. Dans le chapitre **??**, deux fa?ons pour l'adaptation des EFD en 802.11 WLAN sont proposées: garder une trace des volumes échangés dans les deux directions, ou simplement compter les paquets dans une seule direction. Nous évaluons la performance de l'EFD et de ses adaptations, et on les compare aux disciplines de l'état de l'art. Pour l'enqute performances, nous considérons plusieurs facteurs: (1) deux différentes charges - les connexions de longue durée et mélange des transferts court et long; (2) la taille du tampon petits et grands de point d'accès; (3) divers niveaux symétrique entre upload et download. Les résultats des simulations montrent que les deux variantes de EFD - PEFD et EFDACK, capable de forcer un bon niveau d'équité sans avoir à payer une pénalité

en termes de dégradation des performances. En outre, PEFD et EFDACK peut effectivement améliorer les performances des réseaux sans fil, sans les inconvénients habituels associés à d'ordonnancement basé sur la taille. Nous soulevons le problème de granularité tampon dans le chapitre **??**, qui inspire de notre étude des disciplines d'ordonnancement basé sur la taille sur 802.11 Wireless LANSs dans le chapitre **??**. Nous appelons la granularité tampon de l'unité dans lequel la taille du tampon de l'interface de dispositif de réseau est mesurée. Dans le chapitre **??**, nous étudions l'impact de la granularité tampon (au lieu de la taille du tampon) sur la performance des disciplines d'ordonnancement sur 802.11 WLANs. La discussion est menée avec deux granularités tampons - des paquets et d'octets, et deux scénarios de charge. Nous étudions le partage de la capacité de goulot d'étranglement entre uploads et downloads, considérant que des indicateurs le débit agrégé pour le cas de connexions de longue durée, et le temps moyen de réponse conditionnelle dans le cas de charge plus réaliste avec distribution de la taille de la haute variabilité. Nous concluons que la mesure de la tampon de l'unité d'octets est hautement préférable de FIFO, Run2CACK et BEFD, tandis que LASACK, LARS et SCFQ sont insensibles à la granularité tampon.
Notre quatrième contribution est le profilage de trafic de l'entreprise, qui est menée dans le chapitre **??**. Nous développons une compréhension des caractéristiques de base du trafic de l'entreprise moderne à différents niveaux basés sur une trace de taille moyenne laboratoire (Eurecom). La contribution importante est de comparer l'activité interne et externe dans les réseaux d'entreprise modernes. Comme une autre question, une approche supervisé des machine learning est proposé de trouver une fa?on automatique pour identifier les différents r?les (serveurs ou clients) dans les réseaux d'entreprise.
La dernière contribution de cette thèse, dans le chapitre ref chap: ch8, sont les deux nouveaux modèles de charge proposées pour le réseau de l'entreprise. Le premier modèle spécifie comment le trafic coule dans le trafic intranet et Internet, et dans deux directions respectivement sur la base des nouveaux résultats du modèle de trafic de l'entreprise à travers l'étude. Le second modèle re-joue la charge extrait de la trace réelle, prise ainsi en considération l'impact des applications sur le dessus.

2 Etat de l'art

2.1 La taille de base d'ordonnancement

La taille de base d'ordonnancement a re?u beaucoup d'attention de la communauté de la recherche avec des applications aux serveurs Web [28], le trafic Internet [2, 26, 29] ou les réseaux 3G [1, 16]. L'idée principale est de favoriser les flux courts au détriment des flux long, parce que les flux courts sont en général liés à des applications interactives telles que e-mail, navigation sur le Web et DNS requte / réponse; contrairement flux longs qui représentent le trafic de fond. Une telle stratégie fonctionne à condition que flux de longs ne sont pas complètement affamé,

et ce généralement contient sans autre intervention pour le trafic Internet o les flux courts représentent qu'une petite partie de la charge et ne peuvent donc pas monopoliser la bande passante.
Classiquement, les politiques d'ordonnancement fondés sur la taille sont divisés en politiques d'ordonnancement aveugles et non aveugles. Une politique d'ordonnancement aveugle n'est pas conscient de la taille du travail[1] tandis qu'un non-aveugle est. Non politiques d'ordonnancement aveugles sont applicables aux serveurs [28], o le volume de travail est liée à la taille du contenu à transférer. Un exemple typique de la politique non aveugle est la politique Shortest Remaining Processing Time (SRPT), ce qui est optimal parmi toutes les stratégies d'ordonnancement, en ce sens que elle minimise le moyen temps de réponse. Pour obtenir cette propriété, SRPT s'appuie sur une stratégie simple: toujours servir le client qui est le plus proche de l'achèvement.
Dans le cas des appareils de réseau (routeurs, points d'accès, etc), la taille du travail, i.e. le nombre total d'octets à transférer, n'est pas connu à l'avance. Plusieurs aveugles politiques d'ordonnancement fondés sur la taille ont été proposées. Le politique Least Attained Service (LAS) cite Rai04size-basedscheduling fonde sa décision d'ordonnancement de la quantité de services re?us à ce jour par un flux. LAS est connu pour tre optimale si la distribution des tailles de flux a un taux de risque diminue (DHR) car il devient, dans ce contexte, un cas particulier de la Gittins politique optimale [7]. Certains représentants de la famille de Multi-Level Processor Sharing (MLPS) politiques d'ordonnancement [14] ont également été proposées pour favoriser les flux courts. Une politique MLPS se compose de plusieurs niveaux correspondant à des quantités différentes de service atteint d'emplois, avec éventuellement une politique d'ordonnancement différente à chaque niveau. Dans [2], Run2C, qui est un cas spécifique de la politique MLPS, à savoir PS + PS, est proposé et opposée à LAS. Avec Run2C, emplois courts, qui sont définis comme des emplois plus courtes que d'un certain seuil, sont desservis avec la plus haute priorité, alors que emplois de longue durée sont entretenus dans une PS file d'attente. Run2C possède les caractéristiques principales: (i) Comme les emplois de (moyen et) longue durée partager une PS file d'attente, ils sont moins pénalisés que sous LAS; (ii) Il est prouvé analytiquement dans [2] qu'une M/G/1/PS + PS file d'attente offre un temps de réponse plus faible moyenne d'une file d'attente M/G/1/PS, qui est le modèle classique d'un appareil de réseau avec une politique d'ordonnancement FIFO et partagé par les transferts TCP homogènes; (iii) Run2C évite le phénomène de lock-out observé sous LAS [12], o un flux à long pourrait tre bloqué pour une grande quantité de temps par un autre flux de long.
Run2C et LAS partagent un certain nombre d'inconvénients. La comptabilité des flux est complexe. LAS a besoin pour maintenir un état par flux. Run2C doit vérifier, pour chaque paquet entrant, si il appartient à un flux court ou un flux long. Ce dernier est réalisé dans [2], gr?ce à une modification du protocole TCP

[1]Job est une entité générique en file d'attente théorie. Dans le cadre de cette thèse, un emploi correspondant à un flux.

de fa?on à coder dans le TCP numéro de séquence le nombre d'octets envoyés par le flux de la mesure. Une telle approche, qui nécessite une modification globale de tous les h?tes d'extrémité, est discutable[2]. En outre, à la fois LAS et Run2C classer les flux basés sur le nombre cumulé d'octets qu'ils ont envoyé, sans prendre le débit en compte.
Least Attained Recent Service (LARS) est un ordonnancement basé sur la taille con?us pour prendre en compte pour le taux [12]. Il consiste à une variante du LAS, o le nombre d'octets envoyés par chaque flux décro?t avec le temps selon un facteur de décoloration β. LARS est capable de traiter différemment deux flux qui ont envoyé une quantité similaire d'octets, mais à des taux différents, et il limite aussi la durée d'un flux par un autre flux de long à un maximum valeur ajustable.
En dépit de leurs caractéristiques uniques, les politiques d'ordonnancement fondés sur la taille n'ont pas encore été déplacé hors de le laboratoire. Nous croyons que les principales raisons de ce manque d'adoption sont liés aux préoccupations suivantes au sujet des approches de planification fondés sur la taille:

- Politiques d'ordonnancement fondés sur la taille sont essentiellement l'état complet: chaque flux doit tre suivi individuellement. Mme si on peut affirmer que ces politiques doivent tre déployés à des liens de goulot d'étranglement qui sont sans doute à l'orée de réseau – par conséquent à un endroit o le nombre de flux parallèles est modéré – la croyance commune est que les mécanismes stateful sont à éviter en premier lieu.

- Politiques d'ordonnancement fondés sur la taille sont considérées comme trop pénaliser les flux de long. Malgré tous ses défauts, la politique d'ordonnancement et de gestion de mémoire tampon héritage, FIFO/drop tail, ne discrimine pas les flux de longues tandis que les solutions d'ordonnancement fondés sur la taille tendent à se répercuter à la fois le temps de réponse moyen des flux, mais aussi leur variance parce que flux de longues pourrait lock-out les uns des autres.

- Comme leur nom l'indique, les politiques de planification fondés sur la taille considérer une seule dimension d'un flux, à savoir sa taille cumulée. Cependant, les transferts persistants de faible taux souvent transmettre trafic important, *e.g.*, la voix sur IP conversations. En conséquence, il est naturel de représenter le taux et le montant cumulé d'octets de chaque flux.

Un certain nombre de travaux, tels que Run2C et LARS présenté ci-dessus, répondre partiellement aux inconvénients précités des politiques d'ordonnancement fondés sur la taille. Pourtant, au meilleur de notre connaissance, aucun d'eux ne remplissent simultanément les objectifs ci-dessus. Dans cette thèse, nous proposons une nouvelle politique d'ordonnancement, EFD, qui adresses ces objectifs simultanément. Nous

[2]D'autres travaux visent à favoriser les flux courts, en marquant les paquets à la périphérie du réseau, afin de soulager l'ordonnanceur à partir de la comptabilité de flux [21]. Cependant, le déploiement de DiffServ n'est pas envisagée dans un proche avenir à l'échelle d'Internet.

avons d'abord étudier sa performance dans un réseau filaire dans la partie I, puis dans un réseau sans fil dans la partie II.

2.2 L'amélioration des performances dans 802.11 WLANs en utilisant des stratégies d'ordonnancement fondés sur la taille

Dans une infrastructure typique 802.11 WLANs, les stations mobiles équipées avec 802.11 interface communiquer avec un Access Point (AP) sur un canal sans fil, et le AP transmet le trafic vers et depuis le réseau c?blé. Dans de nombreux cas, par exemple l'entreprise, le wireless LAN est le goulot d'étranglement des performances parce que les utilisateurs utilisent généralement un lien avec 100 Mbit/s ou plus grande capacité d'accéder à l'Internet d'aujourd'hui.
Différent du réseau c?blé, il a deux propriétés essentielles – d'une part, le protocole est en semi-duplex, ce qui signifie que uploads et downloads part le milieu; d'autre part, le point d'accès n'est pas accordé une priorité suffisante pour accéder le milieu sous DCF, ce qui signifie que sa file d'attente, qui est généralement de 30 à 100 paquets, a tendance à s'accumuler.
Un problème de performance critique, connu sous le nom "TCP Unfairness" [23], se produit lorsque le trafic TCP est transférée sur un réseau 802.11. Ce problème injustice découle de l'accès à l'égalité des chances à le milieu de l'AP et les stations sans fil dans une cellule sans fil. Comme les stations mobiles échanger du trafic avec le réseau filaire uniquement par l'AP, celui-ci mérite qu'on lui donne plus de chance d'accéder au canal mais il est limité par la méthode d'accès égal définie par la norme 802.11 DCF (Distributed Coordination Function), conduisant au fait que le point d'accès est un goulot d'étranglement qui limite le débit global de perte de trames à cause de débordement de tampon. En outre, lorsque le trafic TCP sont transmis sur wireless LAN, la concurrence entre les TCP ACKs des uploads et des paquets de données TCP des downloads à la mémoire tampon du point d'accès aggrave encore l'injustice et se dégrade finalement la performance globale – car la mémoire tampon de l'AP, qui est généralement faible, a tendance à s'accumuler, entra?nant des pertes de paquets – rppelons que le protocole TCP réagit différemment à la perte de paquets de données et la perte d'ACK.
De nombreux auteurs ont proposé des solutions pour résoudre le TCP problème injustice au niveau des couches différentes: transport, network, or MAC layer [23, 3, 17, 11, 29]. Pilosof *et al.* [23] proposé de modifier la fentre de réception dans TCP ACK à arpenter les sources sur les stations sans fil et de fournir de cette manière plus de bande passante pour le trafic de downloads. Plusieurs auteurs se propose de résoudre le problème injustice en utilisant une appropriée MAC méthode d'accès. Leith textit et al. Cite Leith2005, Leith05tcpfairness proposé de choisir les paramètres appropriés de IEEE 802.11e pour assurer l'équité entre les TCP uploads et les TCP downloads. AAP (Asymmetric Access Point) [18, 10, 9] définit la fentre de contention de l'AP à une valeur constante alors que stations sans fil utilisent la méthode Idle Sense accès. Idle Sense est un MAC protocole de remplacement

à 802.11 qui fait varier la fentre de contention utilisant une approche AIMD, de manière à atteindre une plus grande équité de la DCF héritage qui tend à punir quelques stations quand affirmation est observée. En revanche, avec Idle Sense, toutes les stations ont une fentre de contention similaire qui varie selon le montant global de tentatives de transmission sur le milieu qu'une station peut estimer en observant le canal sans fil. Avec cette fa?on, l'AP est capable d'obtenir deux fois la capacité de transmission de la somme de toutes les stations actives indépendamment du nombre de stations en conflit.
D'autres auteurs considérés les solutions au niveau IP, laissant les protocoles de couche inférieure, en particulier la couche MAC inchangé. Plusieurs stratégies de planification fondés sur la taille ont été prouvés pour tre en mesure de forcer l'équité entre les transferts TCP, et en mme temps d'améliorer la réactivité de connexions courtes et des applications interactives. LASACK [29] comme une extension de la LAS, atténue l'impact de la réactivité des flux TCP ACK en affectant une priorité à un paquet TCP ACK qui est une fonction du nombre d'octets envoyés par le flux correspondant. De cette fa?on, LASACK force l'équité entre upload et download connexions TCP et améliorer l'interactivité per?ue par les utilisateurs finaux. Celle-ci est définie comme la capacité du réseau pour maintenir un temps de réponse faible aux flux courts qui sont générées par les applications interactives des utilisateurs, tel que email et la navigation Web. LARS [12] qui applique une décroissance temporelle du volume de données associées à chaque flux, offre des performances similaires à LASACK, mais évite de lock-out et prend en compte le volume et le taux pour l'ordonnancement.
Politiques d'ordonnancement fondés sur la taille sont fortement recommandés pour tre utilisés pour 802.11 wireless LANs à améliorer l'équité du niveau de débit et de l'interactivité, comme ils sont déployés au niveau IP du point d'accès seulement, laissant protocole d'autres couches inchangé.

2.3 Réseaux d'entreprise

Nous présentons les principaux résultats obtenus dans l'analyse des réseaux d'entreprise puisque, dans la dernière partie de ce travail, nous présentons les résultats de l'analyse d'une trace capturée au large Eurecom et présente les résultats préliminaires de l'utilisation de cette trace et les informations collectées sur le réseau pour concevoir les nouveaux modèles de simulation de charge. Nous illustrons l'utilisation de ces modèles de charge sur certaines des politiques de planification fondés sur la taille que nous avons étudiées dans les deux premières parties de la thèse.
Le trafic Internet à large zone a été largement étudiée dans de nombreux environnements différents à partir des milieux de la recherche au cours des années cite-Caceres89,Gusella90,fowler91,leland94,paxson95,benson10. Cependant, le modèle de trafic et le problème de performances dans les réseaux d'entreprise modernes demeure inexplorée. La raison probable réside dans la difficulté d'un suivi adéquat trafic de l'entreprise et la conviction de la bonne performance des réseaux d'entreprise

dans la pratique.
Nous avons pour objectif de présenter un aper?u des activités de recherche en se concentrant sur la question des réseaux d'entreprise. En général, la grande majorité des études font usage de mesures collectées dans les réseaux d'entreprise filaires ou sans fil, composé de campus, des laboratoires de recherche, *etc.*. La plupart des études de réseau de l'entreprise reposent sur habituellement la trace du niveau des paquets ou du niveau de débit, complétées par d'autres sources telles que SNMP ou syslog données.
Une grande partie des études sont appuyés sur les traces capturées à lien d'accès d'une entreprise, à partir de laquelle l'activité de réseau avec l'Internet externes peuvent tre facilement caractérisées, mais il ne nous éclaire pas sur l'activité dans les réseaux d'entreprise. Récemment, des études ont été menées sur les mesures faite à un routeurs de c?ur d'entreprise [22, 19]. Ils ne reposent pas sur une technique avancée de l'exploration de données, mais plut?t présenter des statistiques descriptives de déduire les performances des réseaux d'entreprise. D'autres études ont mesuré la communication sur les h?tes d'extrémité eux-mmes [6]. Avec cette méthode, tout le trafic lié à chaque h?te d'extrémité est constituée pour l'analyse, y compris le trafic traversant la frontière de l'entreprise dans la communication entre pairs locaux et pairs distants en dehors de l'entreprise. Cependant, il manque de la connaissance de ce qui se passe dans les environs, telles que la charge du réseau. Les auteurs dans [20] ont présenté un certain nombre de techniques pour l'étalonnage des traces de paquets capturés à différents ports de commutation Ethernet, comme levier sémantique TCP pour identifier une perte de mesure, en utilisant prévu réplication de paquets de diffusion pour pointer vers manquant événements de traces, et ainsi de suite.
Les auteurs dans [22] ont fourni une première caractérisation du trafic interne de l'entreprise a enregistré sur un site de grande taille – LBNL (Lawrence Berkeley National Laboratory). Les traces de paquets couvrent plus de 100 heures, pendant laquelle l'activité sur un total de plusieurs milliers de serveurs internes appara?t, bien qu'ils ne pouvaient pas saisir une instance au moment donné tout le trafic circulant à l'intérieur du réseau, que compte tenu de la grande taille et plus encore la structure complexe du réseau LBNL. Ils ont d'abord examiné l'information de base sur le volume de trafic interne et externe, à venir avec une ventilation générale des principales composantes de le trafic. Ils ont également examiné la localité de sources de trafic et les destinations en examinant le fan-in et fan-out des pairs locaux, étant donné que certains pairs locaux sont des serveurs accessibles depuis Internet. Ils ont finalement examiné les caractéristiques des applications qui dominent le trafic. Cet article est essentiellement descriptif, mais ils mis en évidence certains phénomènes spécifiques comme l'existence de défaillances d'établir des connexions spécifiques en interne. Ils ont aussi résolu le problème de charge du point de vue de l'h?te d'extrémité en calculant la quantité de retransmissions TCP vécues par les connexions. Ils ont observé que le taux de retransmission TCP peut atteindre jusqu'à 1%, ce qui est beaucoup moins que l'observation pour le trafic Internet mais

toujours étonnamment grand pour le trafic intranet.
Dans [19], les auteurs présentent une première étape vers la compréhension de la performance TCP dans les réseaux d'entreprise. En particulier, ils ont fondé leur analyse sur un ensemble de données constitué de traces de paquets au commutateur de niveau pris au LBNL en quelques mois, qui est la mme que celle utilisée dans [22]. Ils ont évalué la prévalence des transactions TCP brisés, application utilisée, le débit des connexions TCP, et les phénomènes qui influencent la performance, tels que les retransmissions, la livraison de out-of-order, et la corruption de paquets. En général, ils ont confirmé la présomption commune que les connexions d'entreprise ont faible taux de pertes.
Au meilleur de notre connaissance, aucune étude n'a été conduite à l'aide de grandes et récentes traces recueillies dans un réseau d'entreprise comme celle que nous recueilli à Eurecom. Dans cette perspective, l'étude de la mesure que nous réalisons dans la dernière partie de cette thèse est le premier en son genre. Une difficulté est cependant d'évaluer son représentant. Nous avons cependant rencontrons ici un problème qui est rencontré par la plupart des études d'analyse du trafic effectué par la communauté de mesure, mme pour les traces d'Internet, comme par exemple, les différentes habitudes de l'utilisateur conduire à différentes observations de traces de trafic collectées pour les ISPs européens, américains et asiatiques.

3 Early Flow Discard (EFD) pour l'ordonnancement des paquets

EFD appartient à la famille de la politique d'ordonnancement Multi-Level Processor Sharing. EFD a deux files d'attente. La file d'attente de faible priorité est servi uniquement si la file d'attente de haute priorité est vide. Les files d'attente sont drainées dans un mode de FIFO au niveau des paquets (qui est en général modélisé comme une file d'attente PS au niveau du débit). En termes de mise en ?uvre, une file d'attente physique pour le stockage paquet est divisé en deux files d'attente virtuelles. La première partie de la file d'attente physique est dédié à la file d'attente de haute priorité alors que la seconde partie est la file d'attente de faible priorité. Un pointeur est utilisé pour indiquer la position du dernier paquet de l'virtuel file d'attente haute priorité. Cette idée est similaire à celui qui est proposé dans le mécanisme de la Croix-Protect [15]. Nous portons maintenant notre attention sur la gestion des flux dans EFD et les opérations enqueuing et dequeuing. Nous discutons également de la politique spatiale utilisée lorsque la file d'attente physique est pleine.
EFD maintient une table de flux actifs, o les flux sont définis comme des ensembles de paquets qui part d'une identité commune, consistant en une 5-tuple: adresses source et destination, les ports source et de destination et le numéro de protocole. Les flux restent dans la table à condition que il est un paquet correspondant dans la tampon et jeté lorsque le dernier paquet quitte. Par conséquent, une connexion

TCP (ou un transfert UDP) peut tre divisé dans le temps en plusieurs fragments qui sont traités indépendamment par l'ordonnanceur. Notez que contrairement à la plupart des mécanismes d'ordonnancement qui gardent par états de flux, EFD n'a pas besoin d'utiliser n'importe quel mécanisme de collecte des ordures pour nettoyer la table de flux. Cela se fait automatiquement au moment du départ du dernier paquet du flux. Une entrée de flux assure le suivi des plusieurs attributs, y compris l'identité des flux, contre la taille de flux, nombre de paquets dans la file d'attente. Pour chaque paquet entrant, une recherche est effectuée dans la table de flux de EFD. Une entrée de flux est créé si la recherche échoue et le paquet est mis à la fin de la file d'attente haute priorité. Sinon, le compteur de la taille de flux de l'entrée de flux correspondante est comparée à un seuil prédéfini *th*. Si le compteur de la taille de flux dépasse *th*, alors le paquet est placé à la fin de la file d'attente de faible priorité, sinon le paquet est inséré à la fin de la file d'attente haute priorité. Le but de *th* est de favoriser le démarrage de chaque flux. Dans nos simulations, nous utilisons un *th* valeur de 20 paquets (jusqu'à 30 KB pour les paquets de 1500 octets chacun). De toute évidence, si une connexion est divisée en plusieurs fragments, du point de vue de l'ordonnanceur puis à chaque fois qu'il va traiter chaque fragment comme un unique et d'assigner le début (en de?à du seuil *th*) de chaque fragment d'une priorité élevée, en dirigeant les paquets constituant le début de chaque fragment dans la file d'attente haute priorité. Nous croyons que cela a un sens comme cela se produit uniquement si la connexion n'a pas été actif pendant un temps significatif – il n'a pas été engorgé pendant un certain temps – et peut donc tre considéré comme frais.

Quand un paquet quitte la file d'attente ou est perdu, il diminue le nombre de paquets en attente de l'entrée de flux correspondant. L'entrée de flux séjours dans le tableau à condition que au moins une de ses paquets est dans la file d'attente. Donc **Par conséquent, la taille du tableau des flux de est délimitée par la taille de la file d'attente physique** en paquets [3]. En effet, dans le pire des cas, il ya autant d'entrées que les flux distincts dans la file d'attente physique, chaque avec une paquet.

Cette politique garantit que le tableau des flux reste de petite taille. Aussi, si un flux envoie à un taux élevé pendant une courte période de temps, ses paquets seront dirigés vers la file d'attente de faible priorité seulement pour la période de temps limitée au cours de laquelle le flux est retardée: EFD est sensible à burstiness débit. Quand un paquet arrive à une file d'attente qui est plein, EFD insère d'abord l'arrivée paquet à sa position appropriée dans la file d'attente, et alors abandonne le paquet qui est à la fin de la file d'attente (physique). Cette stratégie de tampon donne implicitement la priorité d'espace pour les flux à court [4], qui diffère de la politique drop-tail de gestion de tampon traditionnelle. En raison de la discussion dans le paragraphe ci-dessus, un flux de court est une partie d'une connexion dont le

[3] Dans la plupart sinon tous les équipements actifs - routeurs, points d'accès - les files d'attente sont comptés dans les paquets et non en octets.

[4]

débit est modéré. Cette approche est similaire au mécanisme Knock-Out dans [5] et la gestion des tampons proposé de LAS dans [25]. Comme les flux de longues dans l'Internet sont pour la plupart des flux TCP, on peut s'attendre à ce que ils vont se remettre de l'événement de perte avec une retransmission rapide; contrairement flux courts qui pourraient time out.
Algorithme 1 représente l'algorithme en pseudo-code, qui vous aidera dans la description de l'ordonnancement EFD. Notez que les états de flux sont efficacement gérées dans EFD en supprimant les entrées de flux dans la table de flux dès que le dernier paquet d'un flux dans la table de flux quitte la file d'attente. Par conséquent, l'existence d'une entrée de l'écoulement dans le tableau des flux, implique que il existe au moins une de ses paquets actuellement dans la file d'attente.
Simulations de réseaux étendus révélé que EFD, comme un ordonnanceur aveugle, conserve les bonnes propriétés de LAS tels que les temps de réponse des petites aux flux courts. En outre, une diminution significative de co?t de comptabilité, d'au moins un ordre de grandeur est obtenue par rapport à LAS, qui est convaincant d'un point de vue pratique. Le lock-out qui constituent le Achilles' heel de LAS sont évités dans EFD, similaire à Run2C. Contrairement à LAS et Run2C, EFD prend intrinsèquement le volume et le débit en compte dans sa décision de planification en raison de la fa?on dont dans lequel la comptabilité est exécutée. Nous avons aussi démontré que EFD peut protéger efficacement basse / moyenne des flux multimédia dans la plupart des situations.

4 L'analyse de la discipline EFD dans 802.11WLANs

Nous considérons que le infrastructure basée WLAN typique o les stations mobiles équipées de l'interface 802.11, communiquent avec un point d'accès (AP) sur une canal sans fil et le point d'accès transmet le trafic vers et à partir de le réseau c?blé. Dans de nombreux cas, le réseau local sans fil est le goulot d'étranglement des performances, *par exemple*, les entreprises ou les laboratoires utilisent fréquemment des liens d'accès à Internet à 100 Mbit/s ou plus grande capacité.
Le protocole de transport TCP est utilisé pour contr?ler la grande majorité des transferts de données en volume (octets envoyés) et la majorité des flux. Lorsque le trafic TCP est relayée par un réseau 802.11, un problème de performance important, connu sous le nom "TCP Unfairness", se produit. Cela se produit lorsque les paquets de données de downloads, à partir du réseau c?blé, et les acknowledgments de TCP niveau des ajouts concurrence pour accéder à la tampon de liaison descendante du point d'accès. Le tampon au point d'accès tend à faire le plein parce que la fonction de coordination distribuée (DCF) à la couche MAC n'est pas suffisante pour donner la priorité du point d'accès par rapport à les autres stations dans la cellule [23]. Plusieurs solutions ont été étudiées à différents niveaux de la pile de protocole (MAC, IP, Transport) pour résoudre le problème "TCP unfairness" [3, 17, 11, 29].
Nous avons étudié les performances de la politique EFD (Early Flow Discard) dans

Algorithm 1 : l'algorithme Early Flow Discard

```
1: function packet_arrival(p)
2: # Un nouveau paquet p de flux F arrive
3: if aucun paquet de F sont présents dans la file d'attente then
4:    créer une entrée de flux de R(F) pour F;
5:    # p est un paquet de priorité élevée
6:    if la file d'attente est pleine then
7:       if seuls les paquets de haute priorité dans la file d'attente then
8:          p est tombé;
9:          retour;
10:      else
11:         le dernier paquet de la file d'attente de faible priorité est rayé;
12:         p est inséré à la fin de la file d'attente haute priorité;
13:      end if
14:   else
15:      p est inséré à la fin de la file d'attente haute priorité;
16:   end if
17: else
18:   # au moins un paquet de F résider dans la file d'attente, de sorte qu'une entrée de flux pour F existe
      dans la table
19:   if nombre d'octets déjà servi de flux F < seuil th then
20:      # p est un paquet de priorité élevée
21:      if la file d'attente est pleine then
22:         if seuls les paquets de haute priorité dans la file d'attente then
23:            p est tombé;
24:            retour;
25:         else
26:            le dernier paquet de la file d'attente de faible priorité est rayé;
27:            p est inséré à la fin de la file d'attente haute priorité;
28:            mettre à jour l'entrée de flux R(F) dans le tableau;
29:         end if
30:      else
31:         p est inséré à la fin de la file d'attente haute priorité;
32:         mettre à jour l'entrée de flux R(F) dans le tableau;
33:      end if
34:   else
35:      # p est un paquet de faible priorité
36:      if la file d'attente est pleine then
37:         p est tombé;
38:         retour;
39:      else
40:         p est mise à la fin de la file d'attente de faible priorité;
41:         mettre à jour l'entrée de flux R(F) dans le tableau;
42:      end if
43:   end if
44: end if
45:
46: function packet_departure(p)
47: # Un paquet p de flux F quitte en raison de la cessation de service ou largage
48: if pas plus de paquets de flux F sont dans la file d'attente après le départ de p then
49:    l'entrée de flux R(F) est supprimée de la table;
50: else
51:    mettre à jour l'entrée de flux R(F) dans le tableau;
52: end if
```

les réseaux 802.11, o les tailles de tampon ont tendance à tre plus petits car ils varient généralement entre 30 et 100 paquets.
Nos contributions sont les suivantes:

- Nous proposons deux adaptations d'EFD dans les réseaux WLAN, EFDACK et PEFD, que visent à atténuer le problème "TCP unfairness". EFDACK garde la trace de la quantité d'octets transmis par chaque flux dans les deux directions, ce qui nécessite la lecture des segments TCP (le champ du numéro de réception) dans les paquets IP. C'est la mme idée que celle de LASACK [29]. En revanche, PEFD garde la trace du nombre de paquets et ne fait aucune distinction entre les unloads et les downloads.

- Nous comparons EFDACK et PEFD de politiques d'ordonnancement state-of-the-art, Run2C, LASACK, LARS et aussi FIFO et SCFQ.

- Nous démontrons que les deux modifications de EFD soit mieux que d'autres politiques d'ordonnancement ou effectuer mme mais avec une baisse des frais généraux en termes de comptabilité de flux[5].

- Nous démontrons que PEFD, qui ne nécessite pas l'inspection des paquets TCP réalise de manière similaire à EFDACK, sauf lorsque la taille de tampon devient trop petit.

- Nous étendons la conception originale de EFD en considérant politiques d'ordonnancement alternatives pour les files d'attente prioritaires haute et basse et de discuter de leur impact.

4.1 L'adaptation de EFD à aux liens half-duplex

La politique EFD originale représente les volumes en octets. Une alternative est de compter les volumes en termes de nombre de paquets. Dans ce document, nous nous référons à ces deux saveurs EFD que BEFD (Byte basé EFD) et PEFD (paquets basé EFD) respectivement. Pour illustrer la différence entre ces deux options, nous considérons le cas d'un réseau local sans fil avec une upload et une download. À la tampon de l'AP, on observe, dans la direction aval, le flux de paquets de le download et le flux de paquets ACK de l'upload. Comme les paquets de données sont généralement MSS paquets tandis ACKs sont 40 paquets d'octets, on voit clairement que comptage des volumes en octets ou paquets aura un impact significatif de la priorité accordée au flux ACK: lors du comptage en octets, sa priorité seront systématiquement maximale tandis que la concurrence entre l'upload et le download sera plus équitable pour le comptage de paquets.
En plus de BEFD et PEFD, nous introduisons une variante de EFD qui tient compte de la nature semi-duplex du protocole de couche MAC. Elle attribue une

[5]L'avantage d'EFD concernant le co?t a été clairement justifiée dans le chapitre **??**. Pour éviter la redondance, nous ne discutons pas de la consommation de mémoire dans ce document que les deux modifications de EFD naturellement hériter de cette bonne propriété d'EFD.

taille service virtuel TCP ACK en tenant compte de la quantité totale de trafic de données qui a été transférée par le flux à ce jour, obtenus par l'intermédiaire du numéro d'acknowledgment TCP dans l'en-tte TCP. Nous appelons EFDACK cette politique d'ordonnancement. En considérant l'exemple mme comme ci-dessus d'une cellule WLAN avec un upload et un download, et en supposant que les flux sont suivis en continu par l'ordonnancement, la priorité d'un paquet est lié à la quantité totale d'octets envoyés par le upload. Bien que EFDACK utilise les informations de niveau TCP, il peut aussi gérer les flux UDP. L'avantage de TCP ici, c'est qu'il permet à l'ordonnanceur de déduire ce qui a été envoyé dans l'autre sens, contrairement UDP. Cela signifie que EFDACK friandises UDP flux ce serait full duplex (par exemple, les transferts VoIP) que les flux simplex, *i.e.* il représente seul direction de transfert.
Essentiellement, l'EFD originale et son adaptation pour réseau 802.11 - EFDACK, sont les régimes FIFO+FIFO puisque les paquets dans chaque file d'attente virtuelle sont vidangés en utilisant la discipline FIFO au niveau des paquets. Nous étudions également dans ce chapitre l'impact des disciplines d'ordonnancement alternatives déployées aux files d'attente à priorité élevée et faible. En particulier, nous considérons deux candidats, FIFO et LAS, ce qui conduit à quatre combinaisons: FIFO+FIFO, LAS+FIFO, FIFO+LAS, LAS+LAS.
Un dernier point à mentionner est que chacune des politiques d'ordonnancement que nous considérons est jumelé à un schéma de gestion de mémoire tampon. Pour FIFO ou SCFQ (une implémentation de Processor Sharing pour les réseaux de paquets [8]), c'est la drop-tail. En revanche, pour les politiques d'ordonnancement fondés sur la taille, lorsque la file d'attente est pleine, le paquet nouvellement arrivé se voit attribuer une priorité selon la politique d'ordonnancement et c'est le paquet avec la plus petite priorité qui est éliminée.
Nous considérons essentiellement deux charges. Tout d'abord, nous utilisons seulement à long terme des flux: tout irréalistes, les résultats obtenus dans une telle charge permettent d'identifier facilement quelques caractéristiques fondamentales d'une politique d'ordonnancement, en raison de la relative simplicité du scénario. Deuxièmement, nous considérons un cas plus réaliste d'un mélange de flux à court et à des flux de long. Nous résumons les paramètres de simulation dans le tableau 1.

4.2 L'affaire des connexions de longue durée

Dans cette section, nous évaluons l'équité des disciplines suivantes: FIFO, BEFD, PEFD, EFDACK, LASCAK, LARS, Run2C et SCFQ pour le cas de transferts TCP de longue durée de vie, afin de mettre en évidence l'impact de la nature semi-duplex des liaisons sans fil 802.11. Dans le cas d'Run2C, on utilise une variante qui prend en compte le volume transféré dans deux directions (par le suivi des progrès du nombre ACK), sinon il ne ferait qu'aggraver l'injustice. Nous nous référons à elle comme Run2CACK.

Table 1: Paramètres de simulation

Simulator		QualNet 4.5		
MAC protocol		802.11a@54Mbit/s		
Workload	long-lived cnxs	buffer size	10-70 MSS	
		composition	5 uploads vs. 5 downloads	
	mixed workload	buffer size	30MSS / 300 MSS	
		transfer size distr.	bounded Zipf	
		load regimes	medium	10 Mbit/s
			high	20 Mbit/s
		traffic ratio	sym.	$\lambda_d/\lambda_u = 1$
			asym.	$\lambda_d/\lambda_u = 10$

Chaque simulation QualNet dure 100 secondes. Nous considérons un scénario avec 5 uploads et 5 downloads. Le problème de l'iniquité TCP re?oit plus prononcée avec la diminution de la taille du tampon [23]. C'est parce que la racine du problème réside dans la concurrence pour accéder à la tampon de l'AP. Inversement, l'injustice finalement dispara?t pour toutes les disciplines d'ordonnancement lorsque l'augmentation de taille de tampon, mais au prix de délais d 'attente extrmes pour *par exemple* FIFO. Dans nos simulations, nous avons considéré des tailles de buffer de 10 à 500 paquets. Nous avons observé que les pertes ne sont pas respectées lorsque le tampon atteint près de 300 paquets. En effet, parce que la fentre annoncée du récepteur est réglée à 65 KB, ce qui équivaut à 43 MSS, au plus 5×43 en circulation des paquets de données pour les 5 flux en aval et $5 \times (43/2)$ en circulation paquets ACK pour les 5 flux en amont peut tre dans la mémoire tampon à un moment quelconque (avec delayed ACK). Pour des valeurs supérieures à 300 paquets, toutes les politiques sont justes, bien que le temps de réponse explose pour FIFO.

We report below on results for small buffer sizes from 10 to 70 packets. Figure 1 représente l'ensemble débit à long terme du les flux de les uploads and les downloads, en prenant la moyenne de 30 simulations indépendantes.

L'injustice marquée entre les uplodas et les downloads vécue par le FIFO héritage est clairement illustré par la figure 1 lorsque la taille du tampon est faible. Par ailleurs, nous observons de le rapport des débits agrégés de le upload and de le download que l'original EFD (*i.e.* BEFD) est encore moins équitable que FIFO, parce que les uplodas très empcher les téléchargements et obtenir un débit de 2 à 3 ordres de grandeur plus grand que celui de les downloads lorsque la taille du tampon est faible. Cela est d? à la priorité élevée accordé aux ACKs tel que mentionné dans la section **??**. Avec petit tampon, cette faible priorité traduit par des taux de perte élevés pour les downloads de sous BEFD et Run2C. En revanche, les taux de pertes connu sous LASACK, PEFD, EFDACK et LARS sont négligeables (avec

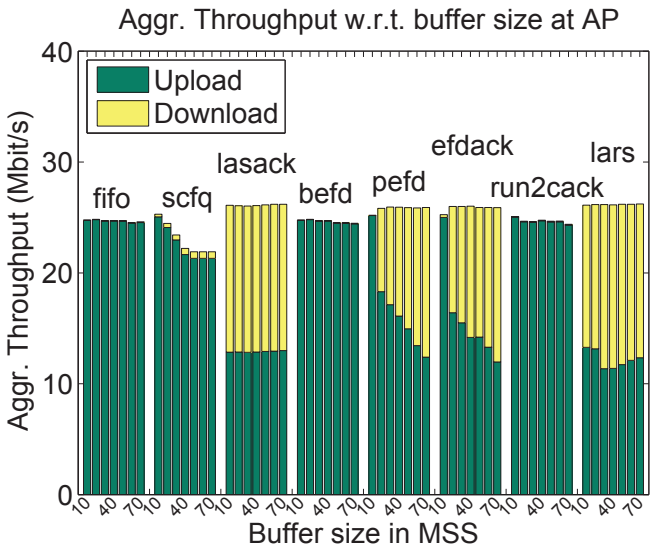


Figure 1: Long-lived connections: 5 uploads against 5 downloads

un tampon de plus de 20 paquets). Bien que Run2CACK garde la trace de trafic bidirectionnel, les connexions de longue durée rapidement se retrouver dans la file d'attente de faible priorité, de sorte que cette politique dégénère en FIFO dans cette configuration.
Figure 1 démontre encore que la capacité du réseau est répartie équitablement entre les uploads et les downloads sous LASACK [11] et sous LARS [12]. Pendant ce temps, PEFD et EFDACK sont capables de forcer un bon niveau d'équité – beaucoup mieux que FIFO, SCFQ et BEFD mais pas aussi parfait que LASACK ou LARS – lorsque la taille de mémoire tampon est supérieure à 20 paquets. Un point intéressant est que l'équité n'est pas obtenue au détriment de la dégradation des performances avec le fait que les débits agrégés sous PEFD et EFDACK sont plus grandes que celles du FIFO et SCFQ.

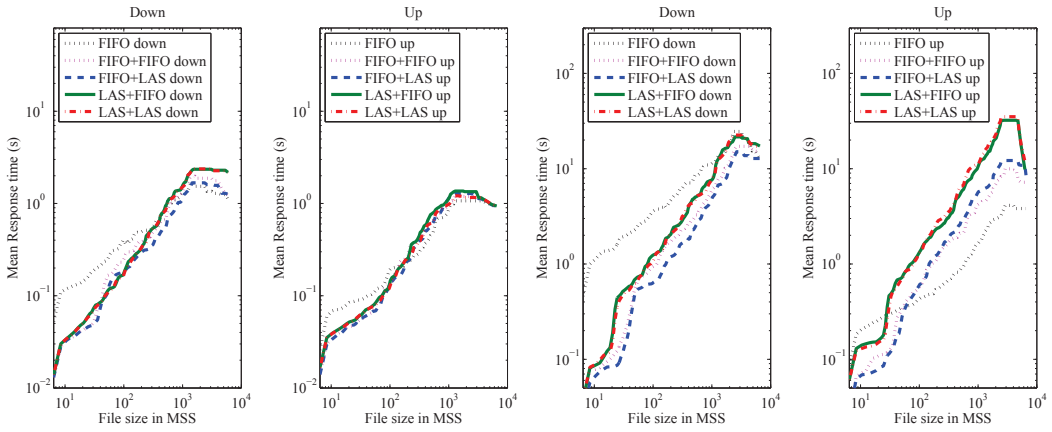
4.3 Évaluation de la performance à l'aide des charges réalistes

Dans cette section, nous avons d'abord étudier l'impact de la variation de la discipline d'ordonnancement pour les régimes similaires à EFD. Nous considérons 4 combinaisons de disciplines: FIFO+FIFO, LAS+FIFO, FIFO+LAS, LAS+LAS en deux versions différentes correspondant à un seuil soit en octets ou en paquets. Nous concluons que l'original FIFO+FIFO est un bon candidat et donc se concentrer sur la PEFD original et le EFDACK origine dans les analyses ultérieures.
Nous comparons ensuite PEFD et EFDACK à FIFO, LARS, LASACK et Run2CACK. Nous examinons le temps de réponse conditionnelle de les uploads et les downloads, en supposant une très asymétrique (le coefficient de variation est 6) distribution de la taille de flux. Enfin, nous discutons de l'impact de la taille du tampon à l'AP sur la performance des politiques d'ordonnancement dans les réseaux 802.11.

Les paramètres de simulation sont donnés dans le tableau **??**, et chaque simulation dure 5000s. Certaines connexions ne sont pas terminées à la fin de la simulation en raison de la fin prématurée de la simulation, mais sous forte charge et pour des simulations qui sont longs suffisamment comme dans notre cas, la raison principale est qu'ils ont été mis de c?té par l'ordonnanceur. Nous présentons les résultats de performance seulement pour les connexions qui ont complété un transfert. Dans cette section, nous ne représentons pas sur les figures, les intervalles de confiance (pour chaque taille de flux) que, compte tenu du nombre de courbes par la figure, ils ont tendance à occulter les graphiques. Pourtant, ils nous ont permis de vérifier que les simulations ont été longues suffisamment pour tirer des conclusions basées sur les temps de réponse moyens conditionnelles. Nous avons mis ces figures et les tableaux relatifs à l'intervalle de confiance dans l'Annexe **??**.

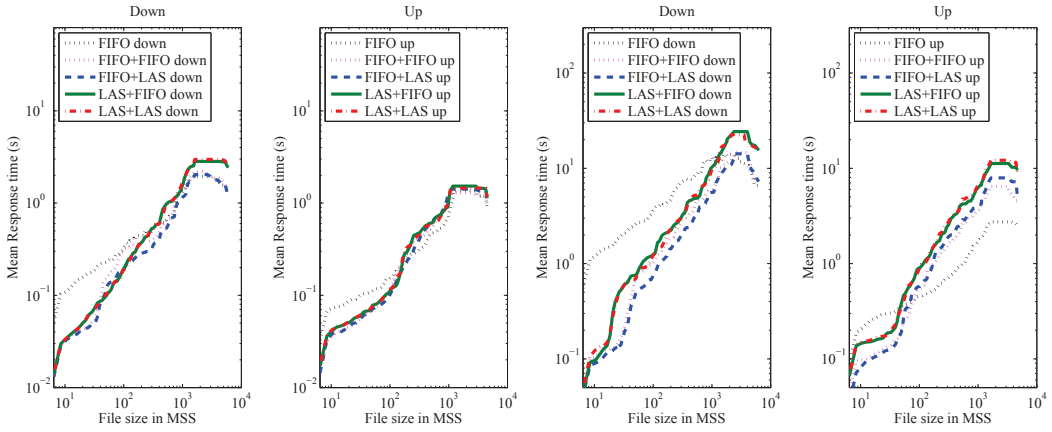
4.3.1 Comparaison des variantes EFD

Dans cette partie, nous considérons quatre variantes de EFD: LAS+FIFO, FIFO+LAS, LAS+LAS ainsi que FIFO+FIFO lui-mme. Pour chaque variante, nous avons deux saveurs, en fonction de l'option de comptabilité qui est soit en octets ou en paquets. Avant d'entrer dans les détails, nous avons besoin d'expliciter la manière dans lequel LAS est utilisé ici. Il s'agit de l'ordonnanceur EFD qui affecte les volumes, que ce soit en paquets ou en octets en fonction de la stratégie. Chaque paquet est donc marquée par un volume associé et, lorsque LAS est utilisé, il gère la file d'attente o il est appliqué de telle sorte que les paquets sont toujours triées dans l'ordre croissant de leur volume associé.
Nous avons effectué des simulations pour une charge symétrique et 10 Mbit/s (charge moyenne) et 20 Mbit/s (charge élevée) respectivement. La taille du tampon est 30 paquets. Moyenne des temps de réponse conditionnelles des régimes à base de octet sont représentés dans la figure 2 alors que le cas pour les régimes à base de paquets sont illustrés dans la figure 3. Les résultats obtenus avec une charge asymétrique sont qualitativement similaires et nous ne les présentons ici.
Nous observons sur la figure 2(a) que les 4 régimes donnent des résultats semblables. Ils offrent un temps de réponse inférieur à flux court par rapport à FIFO, mais au prix d'une légère augmentation du temps d'exécution pour les flux de longues lorsque la charge offerte est modéré à 10 Mbit/s. Un effet similaire pour le cas du scénario basé sur les paquets est visible dans la figure 3(a). Lorsque la charge est élevée, le comportement des 4 régimes différents se distinguent en particulier pour le scénario basé sur les octets. FIFO+LAS propose essentiellement le meilleur temps de réponse pour les deux scénarios, comme illustré dans la figure 2(b) et la figure 3(b). FIFO+FIFO effectue tout près de FIFO+LAS pour le scénario basé sur les octets. Utilisation LAS dans la file d'attente haute priorité semble préjudiciable. Bien que l'utilisation de LAS est différente de la politique initiale LAS qui a une parfaite connaissance de l'histoire de chaque flux, nous croyons que la mauvaise performance obtenue lorsque LAS est utilisé dans la file d'attente prioritaire est la



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

Figure 2: Comparison between various queueing policies in EFD queues – Average response time, symmetric load, byte-based



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

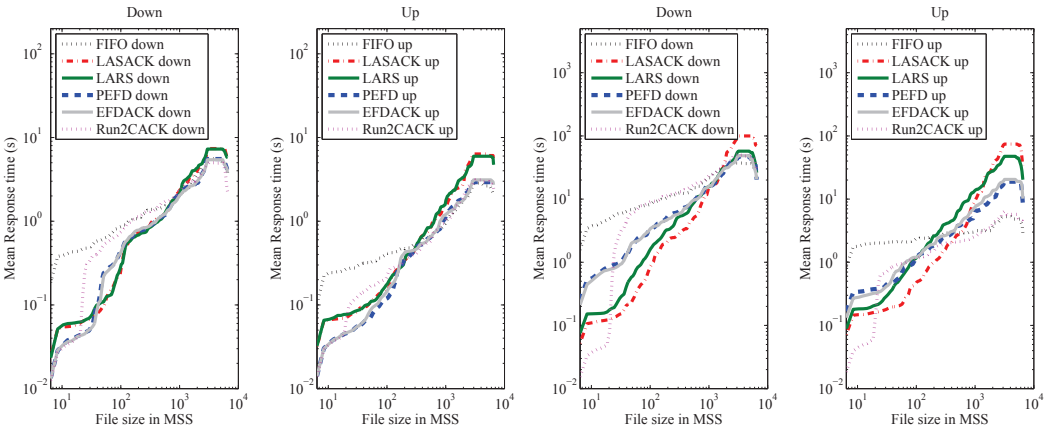
Figure 3: Comparison between various queueing policies in EFD queues – Average response time, symmetric load, packet-based

conséquence de la mauvaise performance de LAS lorsque la distribution a une faible variabilité.

En conclusion, la modification de la discipline de queues de chaque file d'attente individuelle dans un ordonnanceur EFD (raisonnement sur le paquet ou octets) semble bénéfique seulement pour la file d'attente de faible priorité et peut avoir un effet néfaste sur la haute priorité. Dans l'ensemble, le bénéfice de LAS dans la file d'attente de faible priorité semble limitée par rapport à la complexité accrue. Donc nous seulement considérons les saveurs originales, à savoir PEFD et EFDACK dans le reste de ce chapitre.

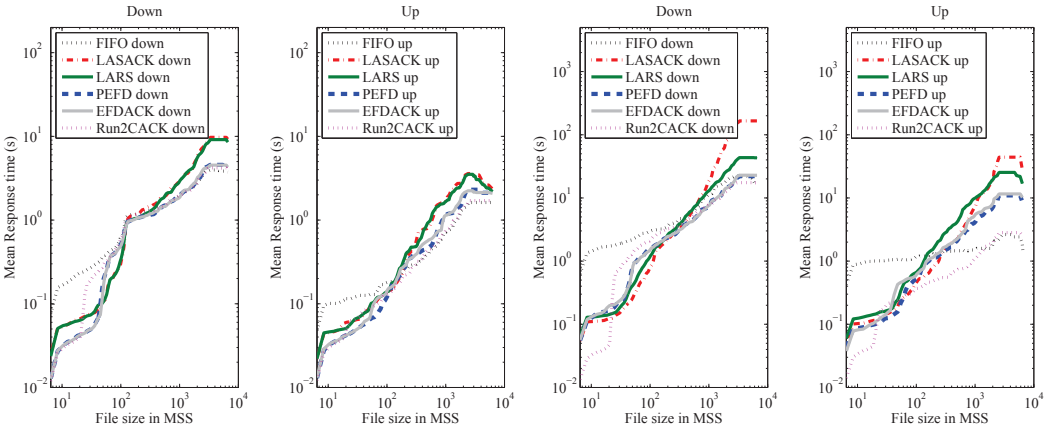
4.3.2 Impact de la charge et le ratio de symétrie

Nous présentons les résultats de simulation pour les 10 et 20 Mbit/s et pour les scénarios symétriques ($\frac{\lambda d}{\lambda u}$=1) et asymétrique ($\frac{\lambda d}{\lambda u}$=10). La taille du tampon est 30 paquets. Temps de réponse conditionnelles des uploads et des downloads sont représentés aux figures 4 et 5 respectivement. Le temps de réponse est défini comme étant le temps nécessaire pour une connexion TCP d'une taille donnée pour achever son transfert (mise en place, le transfert de données et démontage).



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

Figure 4: Comparison of EFD variants for a symmetric workload: average response time – AP buffer of 30MSS



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

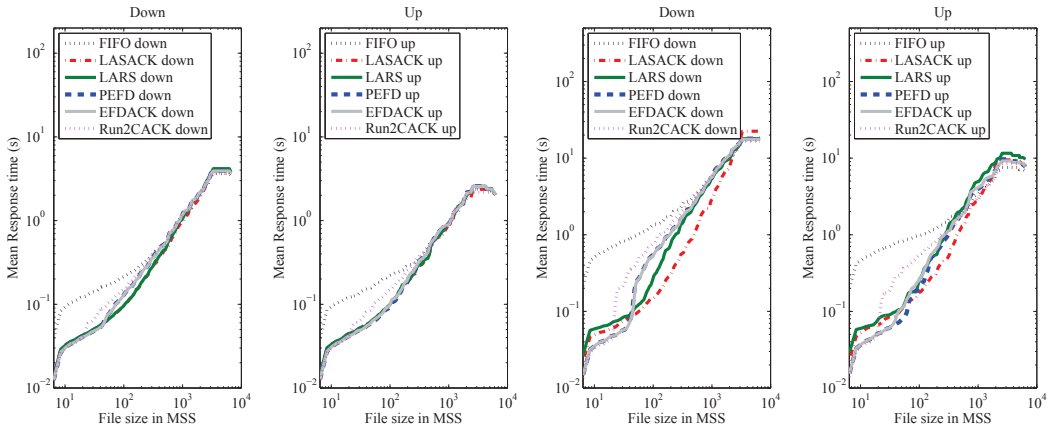
Figure 5: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 30MSS

Nous observons tout d'abord que sous FIFO, pour tous les scénarios et toutes les conditions de charge - encore une charge modérée - le problème TCP injustice

est visible. C'est donc un problème de performance pour chaque réseau 802.11 opérationnel.
En revanche, nous observons que toutes les politiques d'ordonnancement fondés sur la taille atténuer le problème injustice TCP, alors que accorder une haute priorité à flux court, dont les performances améliorent considérablement par rapport à FIFO. Ceux-ci sont obtenus au prix d'une augmentation négligeable du temps de réponse des flux de long.
Une remarque importante est que nous présentons les temps de réponse conditionnels en fonction de la taille des flux de manière à voir l'impact des disciplines d'ordonnancement sur chaque taille de flux. Cependant, d'un point de vue qui on serait peut-tre mieux rendre compte de l'expérience utilisateur, on aurait pu considérer les percentiles de taille de flux sur l'axe des x. Cela aurait amplifié le c?té gauche de chaque parcelle parce que les flux courts représentent la majorité des flux, *par exemple*, le quantile 90-ème est inférieure à 50 paquets environ, ce qui signifie que 90% des flux de conna?tre une amélioration significative avec les politiques d'ordonnancement fondés sur la taille que nous considérons.
Les figures montrent que LASACK performances légèrement meilleures que PEFFD et EFDACK, en particulier pour les flux de taille moyenne. Il s'agit d'un effet secondaire du seuil utilisé dans PEFD et EFDACK. Dans l'ensemble, le message à emporter est que PEFD et EFDACK sont capables de se comporter presque aussi bien que les politiques d'ordonnancement fondés sur la taille de l'état-of-the-art qui gardent la trace de tous les flux (à la différence des politiques comme EFD qui ont une mémoire "limitées dans le tampon").
Ici, Run2CACK utilise le mme seuil que EFD de décider dans quelle file d'attente d'un paquet doit aller. Mais en raison de sa mémoire infinie, les flux de partir plus t?t dans la file d'attente de faible priorité. En fait, Run2CACK donne une transition plus marquée que EFD, avec une protection forte des flux à court préjudiciable à ceux de taille moyenne, de sorte qu'il est en fait plus sensible à la valeur du seuil de transition.

4.3.3 L'impact de la taille du tampon à l'AP

Nous avons considéré tailles de mémoire tampon allant de 10 à 500 paquets. Nous avons choisi deux valeurs représentatives: 30 et 300 paquets. Des simulations sont effectuées dans un scénario de charge asymétrique. Les résultats sont présentés respectivement dans les figures 5 et 6.
Lorsque la taille du tampon est grande - 300 SMS par exemple, il n'est pas plus injuste entre les uploads et les downloads encore avec FIFO indépendamment de la charge, parce que la file d'attente déborde rarement. Néanmoins, ceci est obtenu au prix de très longues périodes passées dans la file d'attente de le downlink AP.
En comparant avec la figure 5, PEFD, EFDACK et LASACK ne souffrent pas bénéficier de plus d'espace de la tampon. Ceci est en accord avec nos résultats antérieurs et les résultats obtenus dans la EFD d'origine, bien que la taille du buffer



(a) Workload of 10Mbit/s (b) Workload of 20Mbit/s

Figure 6: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 300MSS

est directement liée à l'ordonnanceur "mémoire". Cela confirme que, contrairement FIFO, (un peu) des stratégies d'ordonnancement fondés sur la taille sont beaucoup moins sensibles à la taille du tampon réelle.

5 La conclusion et les perspectives d'avenir

Avec l'évolution de succès au cours des années, l'Internet est devenu obligatoire dans notre vie quotidienne et notre travail, ce qui rend le partage d'informations à travers le monde plus facile et plus rapide. Il ya eu une énorme quantité d'efforts sur divers aspects de l'Internet, qui pousse peu à peu vers l'avant le développement de l'Internet. Pourtant, il ya de nombreuses questions qui n'ont pas d'étude ou restent non résolus dans ce domaine sauvage. Nous discutons dans cette thèse, les solutions de QoS pour améliorer la performance per?ue par l'utilisateur en termes de délai, taux de perte, *etc.*, ainsi que la méthodologie de mesure pour les réseaux d'entreprise et supplémentaire pour amélioration de ses performances.

Cette thèse contient des ?uvres originales dans plusieurs domaines tel que l'ordonnancement taille basée sur un réseau c?blé et des réseaux sans fil, le profilage trafic de l'entreprise, et modélisation de la charge de l'entreprise. Dans ce qui suit, nous présentons un résumé des résultats obtenus dans cette thèse.

En tant que solution au problème de QoS dans l'Internet, Early Flow Discard (EFD) est proposé dans cette thèse, motivée par la propriété haute variabilité des tailles de flux et l'attente d'éliminer les inconvénients liés à existants l'ordonnanceurs taille basée. Nous avons montré l'efficacité de l'algorithme proposé (EFD) lorsque déployés dans les n?uds de goulot d'étranglement au service les flux dans le réseau c?blé avec un goulot d'étranglement. De nombreuses simulations montrent que, EFD conserve la propriété la plus souhaitable des méthodes taille basée, à savoir

le temps de réponse faible pour les flux à court, alors que limitant le lock-out des flux longs et protéger efficacement les transferts multimédias de taux bas/moyen. En particulier, EFD est capable de diminuer significative les frais généraux d'état d'écoulement garder par un ordre de grandeur par rapport aux l'methood maintenir l'état plein débit, comme les LAS. En outre, EFD est facile à mettre en ?uvre dans la pratique. Dans l'évaluation performances d'EFD, nous avons examiné plusieurs aspects, tels que le temps de réponse conditionnelle moyenne, les frais généraux de la tenue de l'état de flux, la famine de longs transferts, l'impact sur les transferts multimédia, etc, et comparer sa performance à un large éventail de politiques d'ordonnancement d'autres (FIFO, SCFQ, LAS, Run2C and LARS). Comme des travaux plus approfondie, on pourrait étudier la performance de la discipline EFD en termes de d'autres paramètres, *par exemple* le ralentissement, le taux de perte basé sur les demandes, etc, et étudier la performance d'EFD dans des réseaux hétérogènes encombrés tels que les réseaux qui soutiennent simultanément des applications UDP et TCP, des réseaux TCP avec un délai de propagation hétérogène, et le réseau avec de multiples liens congestionnés.

Nous avons essayé de modèle EFD analytique pour expliquer les résultats de la simulation. La difficulté réside dans le phénomène de fragmentation de flux qui est difficile à décrit sous une forme mathématique, résultant du mécanisme introduit en EFD. Notre étude a montré que l'analyse au niveau sous-flux ne peut pas aider à expliquer les résultats au niveau des flux. Bien que le gain est limité, nous sommes en mesure d'expliquer les résultats de la simulation pour les flux avec une taille en dessous du seuil textit e avec un modèle développé pour le déplacement de sous-flux à flux. Notre modèle est capable de déduire la différence entre les variantes EFD en termes de temps de réponse moyen observé dans les simulations. Pour tre un travail futur, dériver un modèle analytique complet pour EFD vaut la peine.

Nous avons étudié la possibilité d'appliquer EFD à 802.11 Wireless LANs. Le défi provient de plusieurs aspects: d'une part, la file d'attente du point d'accès de liaison descendante se forme naturellement dans les infrastructures réseaux 802.11, d'autre part, EFD doit prendre en compte le trafic bidirectionnel, mme si EFD s'applique à la mémoire tampon de liaison descendante seulement. Notre analyse de l'EFD et de ses adaptations au WLAN 802.11 a montré que, les deux adaptations de EFD – assurer le suivi des volumes échangés dans les deux directions ou tout simplement compter les paquets dans une seule direction - sont efficaces pour faire respecter un bon niveau d'équité, et dans le mme temps sont en mesure de saisir l'avantage d'ordonnancement taille basée. Nous avons également étudié l'impact de la granularité tampon (en octets ou en paquets) sur la performance des politiques d'ordonnancement sur 802.11 WLANs. Nous concluons que la mesure de la mémoire tampon de l'unité d'octets est hautement préférable pour FIFO, Run2CACK et BEFD, tandis que LASACK, LARS et SCFQ sont insensibles à la granularité tampon.

Notez que, l'une des difficultés de le déploiement d'EFD, et de nombreuses autres stratégies d'ordonnancement taille basée de l'Internet, dans la pratique, est d'identifier

les goulots d'étranglement dans l'Internet, qui relys fortement de la tomographie de réseau de l'Internet.
Enfin, nous présentons notre analyse du trafic d'un réseau d'entreprise basée sur une trace du trafic réaliste de recherche en laboratoire (Eurecom) avec une taille moyenne. Nous avons observé que le trafic intranet représentent la majorité de la charge de trafic dans un réseau d'entreprise (près de 90%), alors que le trafic Internet prend une petite fraction (10%). En outre, les grandeurs RTT différentes de trafic intranet et internet - avec une médiane d'environ 1 ms et 100 ms respectivement - sont les principales différences entre ces deux types de trafic. En se concentrant sur les transferts de deux directions, nous avons constaté que la trafic dans les deux directions transmettent symétriquement pour le trafic intranet alors que une asymétrie par rapport d'environ 10 a été observée pour le trafic Internet. Nous pondérées tous ces résultats dans notre modélisation de la charge. Par ailleurs, nous avons également évalué l'impact des applications sur le dessus en rejouant la charge extraite de la trace réelle dans les simulations. Les résultats nous ont dit que l'impact de l'application sur le dessus peut affecter significativement la performance des politiques d'ordonnancement, par conséquent, il doit tre considéré comme un facteur important lorsque l'on con?oit un programmateur pour un ordonnanceur QoS.
Pour les travaux futurs dans ce sens, il nous semble important de mettre plus d'efforts dans la conception de modèle de charge nouvelle qui permettrait de prédire avec plus de précision la performance des solutions de QoS dans un environnement spécifique, esp. les réseaux d'entreprise. Il para?t également essentiel de se déplacer "out of the lab" et de mettre plus d'efforts dans la mise en ?uvre de nos solutions, *par exemple*, en utilisant le routeur Click Modular, et d'effectuer une expérience dans un environnement réel. Ceci pourrait tre réalisé avec un effort pas beaucoup en rempla?ant un point d'accès dans une entreprise par nos AP modifiés et mesurer la performance vécue par les clients, à condition que le WLAN est utilisé non seulement pour accéder à Internet, mais aussi d'accéder aux services internes de l'entreprise.

Bibliography

[1] *QualNet 4.5.* Scalable Networks.

[2] Samuli Aalto and Pasi Lassila. Impact of size-based scheduling on flow level performance in wireless downlink data channels. *Managing Traffic Performance in Converged Networks*, pages 1096–1107, 2007.

[3] Lachlan Andrew, Cesar Marcondes, Sally Floyd, Lawrence Dunn, Romaric Guillier, Wang Gang, Lars Eggert, Sangtae Ha, and Injong Rhee. Towards a Common TCP Evaluation Suite. In *PFLDnet 2008*, March 2008.

[4] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 281–292, New York, NY, USA, 2004. ACM.

[5] Konstantin Avrachenkov, Urtzi Ayesta, Patrick Brown, and Eeva Nyberg. Differentiation between short and long tcp flows: Predictability of the response time. In *Proc. IEEE INFOCOM*, 2004.

[6] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *Computer Communication Review*, pages 92–99, 2010.

[7] Marco Bottigliengo, Claudio Casetti, Carla-Fabiana Chiasserini, and Michela Meo. Short-term Fairness for TCP Flows in 802.11b WLANs. In *IEEE INFOCOM*, 2004.

[8] Ramon Caceres. Measurements of wide area internet traffic. Technical Report UCB/CSD-89-550, EECS Department, University of California, Berkeley, Dec 1989.

[9] Vinton G. Cerf and Robert E. Khan. A Protocol for Packet Network Intercommunication. *IEEE TRANSACTIONS ON COMMUNICATIONS*, 22:637–648, 1974.

[10] Amogh Dhamdhere and Constantine Dovrolis. Open issues in router buffer sizing. *SIGCOMM Comput. Commun. Rev.*, 36(1):87–92, January 2006.

[11] Dinil Mon Divakaran, Giovanna Carofiglio, Eitan Altman, and Pascale Vicat-Blanc Primet. A flow scheduler architecture. In *Networking*, pages 122–134, 2010.

[12] Taoufik En-Najjary and Guillaume Urvoy-Keller. A first look at traffic classification in enterprise networks. In *Proceedings of the 6th International Wireless*

Communications and Mobile Computing Conference, IWCMC '10, pages 764–768, 2010.

[13] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Routers with very small buffers. In *IN IEEE INFOCOM*, 2006.

[14] Sally Floyd and Eddie Kohler. Internet research needs better models. *SIGCOMM Comput. Commun. Rev.*, 33(1):29–34, January 2003.

[15] H.J. Fowler and W.E. Leland. Local area network traffic characteristics, with implications for broadband network congestion management. *IEEE Journal on Selected Areas in Communications*, 9(7):1139 –1149, sep 1991.

[16] Matthew Gast. When Is 54 Not Equal to 54? A Look at 802.11a, b, and g Throughput. Website, 2003. http://www.oreillynet.com/pub/a/wireless/2003/08/08/wireless_throughput.html.

[17] Frédéric Giroire, Jaideep Chandrashekar, Gianluca Iannaccone, Konstantina Papagiannaki, Eve M. Schooler, and Nina Taft. The cubicle vs. the coffee shop: behavioral modes in enterprise end-users. In *Proceedings of the 9th international conference on Passive and active network measurement*, PAM'08, pages 202–211, 2008.

[18] J.C. Gittins. *Multi-armed bandit allocation indices*. Wiley-Interscience, 1989.

[19] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM '94. Networking for Global Communications., 13th Proceedings IEEE*, pages 636 –646 vol.2, June 1994.

[20] Yan Grunenberger, Martin Heusse, Franck Rousseau, and Andrzej Duda. Experience with an implementation of the Idle Sense wireless access method. In *Proceedings of the 2007 ACM CoNEXT conference*, CoNEXT '07, pages 24:1–24:12, 2007.

[21] Saikat Guha, Jaideep Chandrashekar, Nina Taft, and Konstantina Papagiannaki. How healthy are today's enterprise networks? In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 145–150, 2008.

[22] R. Gusella. A measurement study of diskless workstation traffic on an ethernet. *IEEE Transactions on Communications*, 38(9):1557 –1568, sep 1990.

[23] Aymen Hafsaoui, Denis Collange, and Guillaume Urvoy-Keller. Revisiting the Performance of Short TCP Transfers. In *Networking*, pages 260–273, 2009.

[24] Aymen Hafsaoui, Guillaume Urvoy-Keller, Matti Siekkinen, and Denis Collange. Detecting and Profiling TCP Connections Experiencing Abnormal Performance. In *TMA*, pages 107–120, 2012.

[25] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.*, 21(2):207–233, May 2003.

[26] Martin Heusse, Franck Rousseau, Romaric Guillier, and Andrzej Duda. Idle sense: An optimal access method for high throughput and fairness in rate diverse wireless LANs. In *In ACM SIGCOMM*, pages 121–132. ACM Press, 2005.

[27] Martin Heusse, Guillaume Urvoy-Keller, and Andrzej Duda. Layer 2 vs. Layer 3 Mechanisms for Improving TCP Performance in 802.11 Wireless LANs. In *ITC*, 2008.

[28] Martin Heusse, Guillaume Urvoy-Keller, Andrzej Duda, and Timothy X. Brown. Least attained recent service for packet scheduling over wireless lans. In *WoWMoM 2010*, 2010.

[29] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pages 314–329, 1988.

[30] Leonard Kleinrock. *Computer Applications, Volume 2, Queueing Systems*. Wiley-Interscience, 1 edition, April 1976.

[31] A Kortebi, S Oueslati, and J Roberts. Cross-protect: Implicit service differentiation and admission control. In *IEEE HPSR*, 2004.

[32] Lawrence Berkeley National Laboratory. LBNL/ICSI Enterprise Tracing Project. Website, 2011. http://www.icir.org/enterprise-tracing.

[33] Pasi Lassila and Samuli Aalto. Combining opportunistic and size-based scheduling in wireless systems. In *MSWiM '08: Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 323–332, New York, NY, USA, 2008. ACM.

[34] D. J. Leith and P. Clifford. Using the 802.11e EDCF to Achieve TCP Upload Fairness over WLAN Links. In *Proceedings of the Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, WIOPT '05, pages 109–118, Washington, DC, USA, 2005.

[35] D. J. Leith, P. Clifford, D. Malone, and A. Ng. TCP fairness in 802.11e WLANs. *IEEE Communications Letters*, 9:964–966, 2005.

[36] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1 –15, feb 1994.

[37] Elena Lopez-Aguilera, Martin Heusse, Yan Grunenberger, Franck Rousseau, Andrzej Duda, and Jordi Casademont. An Asymmetric Access Point for Solving

the Unfairness Problem in WLANs. *IEEE Transactions on Mobile Computing*, 7(10):1213–1227, October 2008.

[38] Elena Lopez-Aguilera, Martin Heusse, Yan Grunenberger, Franck Rousseau, Andrzej Duda, and Jordi Casademont. An Asymmetric Access Point for Solving the Unfairness Problem in WLANs. *IEEE Transactions on Mobile Computing*, 7(10):1213–1227, October 2008.

[39] Xiaoqiao Meng, Starsky H. Y. Wong, Yuan Yuan, and Songwu Lu. Characterizing flows in large wireless data networks. In *MOBICOM*, pages 174–186, 2004.

[40] Boris Nechaev, Mark Allman, Vern Paxson, and Andrei Gurtov. A preliminary analysis of tcp performance in an enterprise network. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 7–7, Berkeley, CA, USA, 2010. USENIX Association.

[41] Boris Nechaev, Vern Paxson, Mark Allman, and Andrei Gurtov. On calibrating enterprise switch measurements. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 143–155, New York, NY, USA, 2009.

[42] Wael Noureddine and Fouad Tobagi. Improving the performance of interactive tcp applications using service differentiation. In *Computer Networks Journal*, pages 2002–354. IEEE, 2002.

[43] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, and Brian Tierney. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.

[44] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226 –244, jun 1995.

[45] Marcin Pietrzyk, Louis Plissonneau, Guillaume Urvoy-Keller, and Taoufik En-Najjary. On profiling residential customers. In *TMA*, 2011.

[46] Saar Pilosof, Ramachandran Ramjee, Danny Raz, Ran Ramjee, Yuval Shavitt, and Prasun Sinha. Understanding TCP fairness over Wireless LAN. In *IEEE INFOCOM*, pages 863–872, 2003.

[47] J. Postel. Transmission control protocol. RFC 793 (standard), 1981.

[48] K. Psounis, A. Ghosh, B. Prabhakar, and G. Wang. Sift: A simple algorithm for tracking elephant flows, and taking advantage of power laws. In *43rd Annual Allerton Conference on Control, Communication and Computing*, 2005.

[49] Konstantinos Psounis, Arpita Ghosh, and Balaji Prabhakar. Sift: A low-complexity scheduler for reducing flow delays in the internet. Technical Report CENG-2004-01, University of Southern California, 2004.

[50] Idris A. Rai, Ernst W. Biersack, and Guillaume Urvoy-keller. Size-based scheduling to improve the performance of short tcp flows. *IEEE Network*, pages 12–17, vol.19, 2004.

[51] Idris A. Rai, Guillaume Urvoy-keller, and Ernst W. Biersack. Size-based Scheduling with Differentiated Services to Improve Response Time of Highly Varying Flow Sizes. In *in Proceedings of the 15th ITC Specialist Seminar, Internet Traffic Engineering and Traffic Management*, 2002.

[52] Idris A. Rai, Guillaume Urvoy-Keller, Mary K. Vernon, and Ernst W. Biersack. Performance analysis of las-based scheduling disciplines in a packet switched network. In *SIGMETRICS 2004/PERFORMANCE 2004: Proceedings of the joint international conference on Measurement and modeling of computer systems*, volume 32, pages 106–117, New York, NY, USA, June 2004. ACM Press.

[53] Idris A. Rai, Guillaume Urvoy-keller, Mary K. Vernon, and Ernst W. Biersack. Performance models for las-based scheduling disciplines in a packet switched network. In *In ACM SIGMETRICS*, page 106117, 2004.

[54] Gaurav Raina and Damon Wischik. Buffer sizes for large multiplexers: Tcp queueing theory and instability analysis. In *EuroNGI*, 2005.

[55] L.E. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687–690, 1968.

[56] Bianca Schroeder and Mor Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Trans. Internet Technol.*, (1):20–52, vol.6, 2006.

[57] B. Sikdar, S. Kalyanaraman, and K. S. Vastola. Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK. In *In Proc. of the IEEE GLOBECOM*, pages 25–29, 2001.

[58] Guillaume Urvoy-Keller and André-Luc Beylot. Improving flow level fairness and interactivity in WLANs using size-based scheduling policies. In *MSWiM '08: Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 333–340. ACM, 2008.

[59] Curtis Villamizar and Cheng Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, October 1994.

[60] D. Wei, P. Cao, and S. Low. Time for a TCP Benchmark Suite?, 2005.

[61] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN

traffic at the source level. *IEEE/ACM Transaction on Networking*, 5(1):71–86, 1997.

[62] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. ELSEVIER, second edition, 2005.

Appendix A

Results for Chapter 6

A.1 Figures showing the mean and the confidence interval

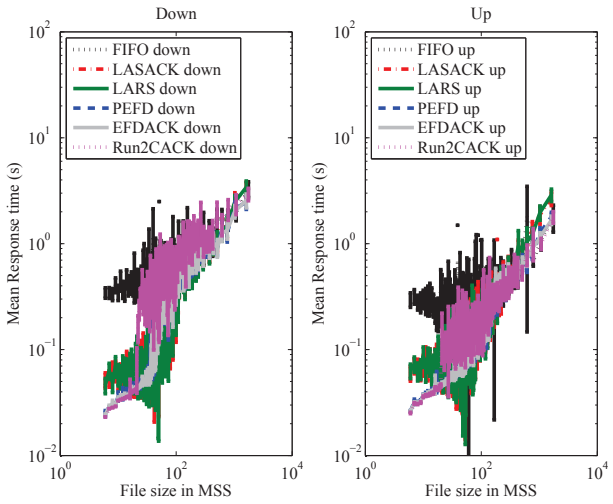


Figure A.1: Comparison of EFD variants for a symmetric workload: average response time – AP buffer of 30MSS, workload of 10Mbit/s

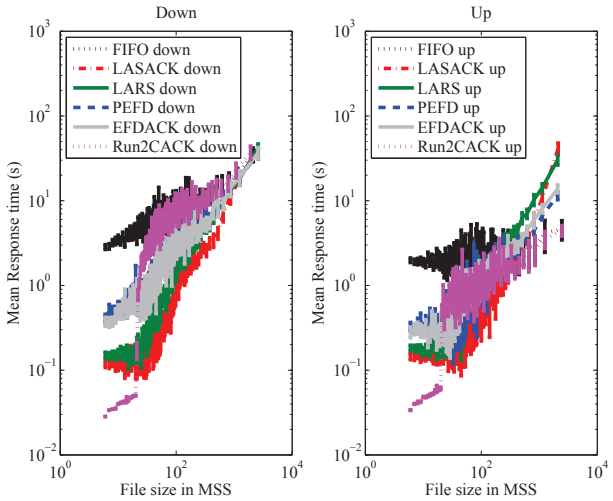


Figure A.2: Comparison of EFD variants for a symmetric workload: average response time – AP buffer of 30MSS, workload of 20Mbit/s

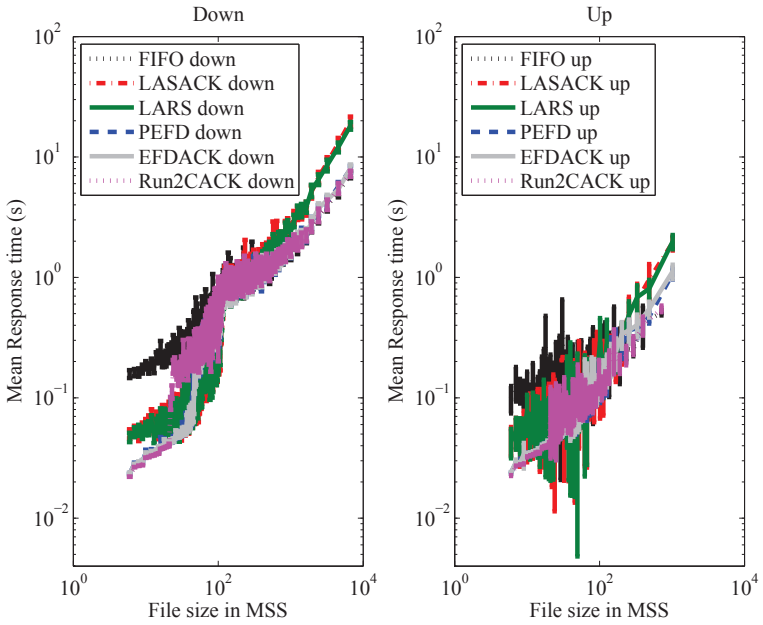


Figure A.3: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 30MSS, workload of 10Mbit/s

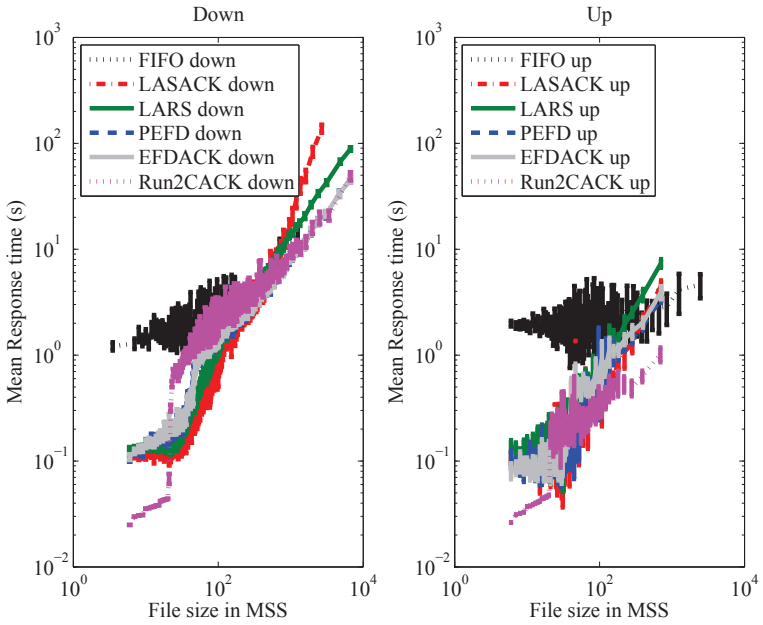


Figure A.4: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 30MSS, workload of 20Mbit/s

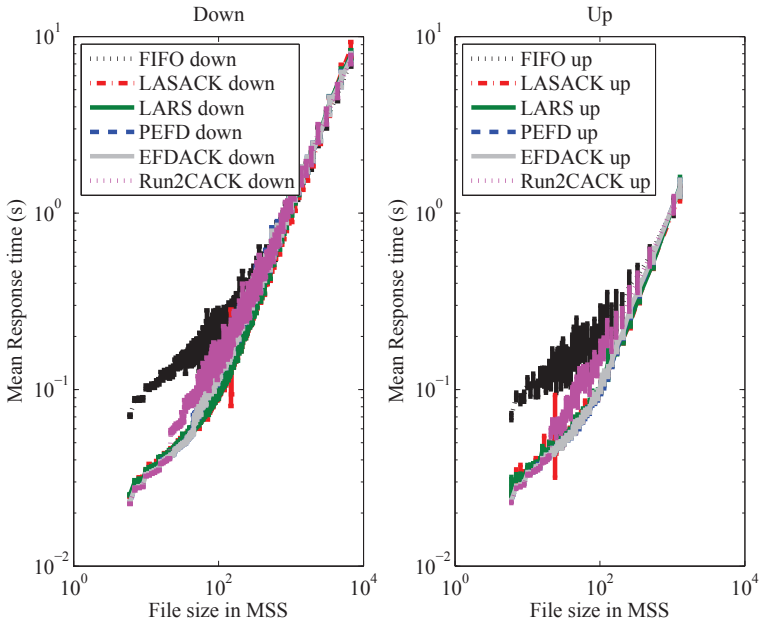


Figure A.5: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 300MSS, workload of 10Mbit/s

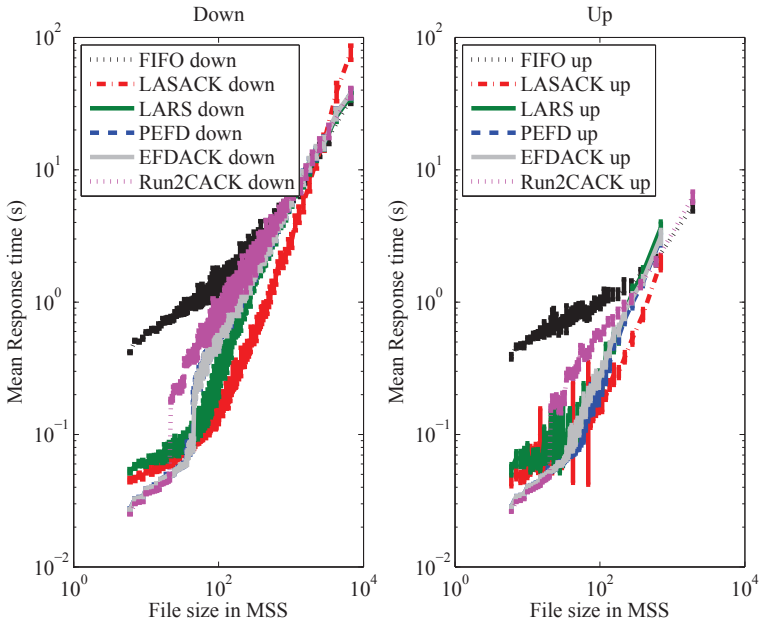


Figure A.6: Comparison of EFD variants for an asymmetric workload: average response time – AP buffer of 300MSS, workload of 20Mbit/s

A.2 Tables showing the mean and the confidence interval

Table A.1: Performance Statistics - symmetric - 30MSS - 10Mbit/s

			Download		Upload	
			short flows	long flows	short flows	long flows
Mean size (MSS)			21	1063	21	1055
Response time (seconds)	mean	FIFO	0.432	2.404	0.301	1.202
		LASACK	0.081	2.547	0.082	2.158
		LARS	0.082	2.418	0.083	2.039
		PEFD	0.071	1.965	0.048	1.197
		EFDACK	0.069	1.930	0.050	1.191
		Run2CACK	0.136	2.483	0.066	1.135
	95%-CI	FIFO	[0.416,0.448]	[2.172,2.637]	[0.286,0.317]	[1.025,1.380]
		LASACK	[0.078,0.085]	[2.276,2.818]	[0.078,0.086]	[1.900,2.415]
		LARS	[0.078,0.086]	[2.183,2.653]	[0.079,0.087]	[1.824,2.253]
		PEFD	[0.069,0.073]	[1.797,2.134]	[0.047,0.049]	[1.102,1.292]
		EFDACK	[0.068,0.071]	[1.770,2.091]	[0.049,0.051]	[1.097,1.285]
		Run2CACK	[0.130,0.141]	[2.256,2.710]	[0.063,0.068]	[1.041,1.228]

Table A.2: Performance Statistics - symmetric - 30MSS - 20Mbit/s

			Download		Upload	
			short flows	long flows	short flows	long flows
Mean size (MSS)			21	1112	21	1124
Response time (seconds)	mean	FIFO	4.034	21.514	1.974	3.276
		LASACK	0.189	21.929	0.188	17.395
		LARS	0.282	17.349	0.286	14.644
		PEFD	0.825	16.913	0.423	6.750
		EFDACK	0.746	16.707	0.398	8.070
		Run2CACK	1.088	22.950	0.249	2.991
	95%-CI	FIFO	[3.940,4.129]	[19.572,23.455]	[1.926,2.022]	[2.914,3.638]
		LASACK	[0.184,0.194]	[19.039,24.818]	[0.182,0.194]	[15.130,19.661]
		LARS	[0.276,0.288]	[15.881,18.818]	[0.278,0.293]	[13.393,15.894]
		PEFD	[0.796,0.853]	[15.443,18.382]	[0.406,0.439]	[6.262,7.238]
		EFDACK	[0.720,0.773]	[15.185,18.230]	[0.383,0.413]	[7.449,8.692]
		Run2CACK	[1.049,1.127]	[21.007,24.892]	[0.238,0.259]	[2.729,3.254]

Table A.3: Performance Statistics - asymmetric - 30MSS - 10Mbit/s

			Download		Upload	
			short flows	long flows	short flows	long flows
Mean size (MSS)			21	1065	21	1064
Response time (seconds)	mean	FIFO	0.231	1.827	0.132	0.801
		LASACK	0.092	3.195	0.066	2.157
		LARS	0.089	3.031	0.067	1.992
		PEFD	0.076	1.852	0.044	1.045
		EFDACK	0.075	1.865	0.046	1.101
		Run2CACK	0.097	1.837	0.046	0.887
	95%-CI	FIFO	[0.227,0.235]	[1.765,1.889]	[0.121,0.142]	[0.699,0.904]
		LASACK	[0.090,0.095]	[3.039,3.351]	[0.061,0.072]	[1.815,2.500]
		LARS	[0.087,0.091]	[2.894,3.167]	[0.062,0.073]	[1.709,2.276]
		PEFD	[0.075,0.077]	[1.789,1.914]	[0.043,0.045]	[0.914,1.176]
		EFDACK	[0.074,0.076]	[1.802,1.929]	[0.045,0.048]	[0.960,1.242]
		Run2CACK	[0.095,0.098]	[1.777,1.897]	[0.045,0.048]	[0.766,1.007]

Table A.4: Performance Statistics - asymmetric - 30MSS - 20Mbit/s

			Download		Upload	
			short flows	long flows	short flows	long flows
Mean size (MSS)			21	1042	21	1015
Response time (seconds)	mean	FIFO	1.658	11.919	1.973	3.189
		LASACK	0.188	27.104	0.152	16.385
		LARS	0.238	13.944	0.201	10.961
		PEFD	0.274	8.200	0.147	4.752
		EFDACK	0.278	8.133	0.162	5.492
		Run2CACK	0.337	9.170	0.093	1.258
	95%-CI	FIFO	[1.603,1.714]	[10.621,13.217]	[1.925,2.021]	[2.848,3.530]
		LASACK	[0.185,0.191]	[25.355,28.854]	[0.144,0.161]	[13.366,19.404]
		LARS	[0.235,0.241]	[13.482,14.407]	[0.192,0.211]	[9.692,12.229]
		PEFD	[0.270,0.277]	[7.938,8.462]	[0.139,0.156]	[4.220,5.284]
		EFDACK	[0.275,0.281]	[7.874,8.393]	[0.155,0.169]	[4.844,6.139]
		Run2CACK	[0.331,0.343]	[8.818,9.522]	[0.089,0.096]	[1.150,1.367]

Table A.5: Performance Statistics - asymmetric - 300MSS - 10Mbit/s

			Download		Upload	
			short flows	long flows	short flows	long flows
Mean size (MSS)			21	1079	21	1108
Response time (seconds)	mean	FIFO	0.112	1.210	0.114	1.247
		LASACK	0.042	1.179	0.043	1.219
		LARS	0.042	1.199	0.044	1.246
		PEFD	0.042	1.245	0.040	1.244
		EFDACK	0.042	1.241	0.041	1.248
		Run2CACK	0.047	1.241	0.048	1.259
	95%-CI	FIFO	[0.111,0.113]	[1.155,1.266]	[0.112,0.116]	[1.083,1.410]
		LASACK	[0.041,0.042]	[1.112,1.246]	[0.042,0.044]	[1.009,1.428]
		LARS	[0.041,0.042]	[1.138,1.260]	[0.043,0.044]	[1.046,1.446]
		PEFD	[0.042,0.043]	[1.186,1.304]	[0.040,0.041]	[1.054,1.435]
		EFDACK	[0.042,0.042]	[1.182,1.300]	[0.040,0.041]	[1.058,1.438]
		Run2CACK	[0.047,0.047]	[1.184,1.298]	[0.047,0.049]	[1.092,1.426]

Table A.6: Performance Statistics - asymmetric - 300MSS - 20Mbit/s

			Download		Upload	
			short flows	long flows	short flows	long flows
Mean size (MSS)			21	1041	21	1034
Response time (seconds)	mean	FIFO	0.668	6.463	0.571	3.200
		LASACK	0.065	6.187	0.072	5.932
		LARS	0.086	5.720	0.090	5.385
		PEFD	0.087	6.106	0.060	4.307
		EFDACK	0.086	6.096	0.067	4.368
		Run2CACK	0.138	6.597	0.108	3.442
	95%-CI	FIFO	[0.664,0.671]	[6.221,6.705]	[0.564,0.578]	[2.931,3.469]
		LASACK	[0.064,0.065]	[5.668,6.706]	[0.070,0.075]	[4.387,7.478]
		LARS	[0.085,0.087]	[5.496,5.943]	[0.087,0.093]	[4.709,6.060]
		PEFD	[0.086,0.088]	[5.860,6.353]	[0.059,0.061]	[3.776,4.838]
		EFDACK	[0.085,0.086]	[5.850,6.342]	[0.065,0.068]	[3.844,4.892]
		Run2CACK	[0.137,0.140]	[6.341,6.854]	[0.106,0.110]	[3.107,3.778]