STOCHASTIC EVALUATION OF FAIR SCHEDULING WITH APPLICATIONS TO QUALITY-OF-SERVICE IN BROADBAND WIRELESS ACCESS NETWORKS

by

Mohammed Hawa

B.S.E.E., University of Jordan, Amman, Jordan, 1997
M.S.TELECOM., University of London, London, UK, 1999

Presented to the Department of Electrical Engineering and Computer Science
and the Faculty of the Graduate School of the University of Kansas
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Committee:

Prof. David W. Petr, Chair

Prof. Victor Frost

Prof. Joseph Evans

Prof. Rongqing Hui

Prof. Tyrone Duncan

THE UNIVERSITY OF KANSAS

August 2003

STOCHASTIC EVALUATION OF FAIR SCHEDULING WITH APPLICATIONS TO QUALITY-OF-SERVICE IN BROADBAND WIRELESS ACCESS NETWORKS

Mohammed Hawa, Ph.D.
The University of Kansas, 2003

A prerequisite to providing multimedia services over wireless packet-switched networks is the provisioning of new mechanisms that allow such networks to treat packets differently according to their limitations and performance bounds. Those new mechanisms are collectively included within what is called a *Quality-of-Service* (QoS) architecture. A desirable feature of a QoS architecture for wireless networks is the ability to provide wireless end users with the same QoS guarantees that wired users currently enjoy.

To that end, we develop a new and efficient scheduling architecture to support bandwidth and delay QoS guarantees for packet-switched Broadband Wireless Access (BWA) networks. Our design goals are simplicity and improved network performance. The architecture we develop in this research effort supports various types of traffic including constant bit rate, variable bit rate (real-time and non-real-time) and best effort.

A key element of our proposed QoS architecture is known as *Fair scheduling* or *Fair Queueing* (FQ). Fair scheduling algorithms have received much attention in recent years because of their ability to provide a wide range of QoS guarantees to end users. In this dissertation, we concentrate our efforts on analyzing the stochastic performance of such a class of fair scheduling systems. We start by presenting a new analysis method that results in reasonably tight upper and lower bounds on mean packet delay and mean buffer occupancy experienced by fair scheduling algorithms under Poisson arrivals. We coin the new term M/G/FQ to describe this analysis method, and provide a range of simulation experiments to validate its results.

Using more simulations, we continue our study of fair scheduling algorithms by comparing the performance of three packet-based FQ policies to each other and to the reference scheduling policy called Generalized Processor Sharing (GPS) under random Poisson arrivals. Our experiments allow us to derive many useful insights into the operation of such packet-based FQ policies, which helps us better understand their specific characteristics and properties.

This work is dedicated to my loving parents.

Acknowledgments

My sincere thanks are due to Dr. David W. Petr for his advice and support in his role as the chairman for the committee of this dissertation. Dr. Petr's comments, stimulating discussions, suggestions and criticism have been of a great help for me in completing this work.

My extreme appreciation also goes to my committee members Drs. Victor Frost, Joseph Evans, Rongqing Hui and Tyrone Duncan for their time and effort and valuable suggestions that improved this research effort. I am especially thankful to Dr. Rongqing Hui for accepting to join the committee on a short notice.

I would also like to extend a special thanks to my friends, faculty and staff members at the University of Kansas for their continuous support and encouragement.

Last but not least, I am greatly indebted to my parents and family for their support, guidance and encouragement throughout my life, especially during the last four years. My parent's faith in me and their patience have been always my source of inspiration.

Partial funding for this project was provided by Sprint Corporation from August 2001 to August 2002.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

THE explosive growth that the Internet has experienced, its nearly universal coverage, flexibility, ease of use and the continuously declining cost of providing it, all make it the communication network of the future. People are turning to the Internet to check the latest news, buy and sell different products, send email, share their ideas, engage in heated debates and chats, and experiment with video conferencing. Music files and streaming video are being downloaded over the Internet in huge volumes every day.

In spite of all the attractive features that the Internet can provide as the network of the future, there still exist a number of technical challenges that need to be addressed. The Internet uses the *Internet Protocol* (IP), which was designed originally to provide a *same-for-all best-effort* packet delivery service. Such service is not the ideal solution for handling heterogeneous types of traffic that exhibit widely different service requirements. Instead, a more desirable solution would utilize a common infrastructure to provide multiple layers of service that suit the different requirements of different customers. This does not reflect the current status of the Internet.

Another shortcoming of the IP architecture is the lack of any network guarantees in terms of mean packet delay and/or packet loss rate. Such guarantees are particularly important if we were to effectively carry real-time traffic, such as voice-over-IP or streaming video, over the Internet.

In short, since the Internet was not designed with heterogeneous and real-time traffic in mind, new mechanisms should be put in place to support the diverse requirements of such traffic. Those new mechanisms are collectively included within what is called a *Quality-of-Service* (QoS) architecture for the Internet. In such an architecture, the Internet (a packet-switched network) will be provided with the tools to treat packets differently; for example, real-time packets will be given priority over non-real-time packets allowing them to traverse the network faster and arrive at the destination within their required delay bounds.

QoS in packet-switched networks can be characterized in terms of a specific set of carefully defined parameters including: *delay*, *delay jitter*, *bandwidth* and *loss* or *error rate*. The ability to maintain QoS in such networks requires sophisticated supervision and control mechanisms (known as *QoS management functions*) to ensure that the desired QoS parameters are attained and sustained.

A critical element of such QoS management functions is the *scheduling* of packets as they traverse different routers along their way through the network. Scheduling defines the order in which packets get transmitted at the output of each router, thus ensuring that packets from different applications meet their QoS constraints. An optimal scheduling mechanism will provide the necessary QoS guarantees required by heterogeneous classes of traffic while utilizing the resources as efficiently as possible.

The Internet, as we know it, has also expanded to the wireless realm, especially in recent years. Wireless technologies are gaining more and more popularity within today's agile and more mobile working environment. It is expected that, in the near future, mobile users will be able to transparently access communication networks from anywhere in the world at any time. The mobile user will use a powerful palmtop or laptop computer that is equipped with a wireless connection to the Internet to perform many activities including sending email, browsing web sites, making phone calls and even having a videoconference.

Such crossover to the wireless side leaves us with the desire to provide multimedia content to wireless users while extending the same QoS guarantees from the wired part of the network to the wireless part. The end user with a wireless terminal should be able to use all the applications that a user with a wired terminal enjoys with minimal service degradation.

Due to the inherent nature of the wireless environment being a *shared* medium, an extra level of complexity in the protocol stack is needed. This new level of complexity is usually represented by the wireless *Medium Access Control* (MAC) protocol. The wireless MAC layer is one of the key components in allowing QoS guarantees to be implemented in the wireless medium because it defines how the air interface in a wireless channel is shared among multiple users, thus having a significant impact on user performance, system capacity and remote terminal complexity.

The first part of this research effort focuses on designing an effective packet-based QoS architecture for the wireless MAC environment that can seamlessly integrate with the QoS architectures proposed for the wired Internet. The scheme we develop concentrates on the scheduling aspects of such a QoS mechanism. This is because scheduling represents a key element in maintaining the QoS guarantees required by end users.

Providing QoS within the wireless infrastructure poses several new technical challenges that did not exist in wired networks. Those challenges arise from the unique limitations of wireless channels, such as excessive amount of interference, higher error rates and lower bandwidth.

It is also interesting to note that the inherent characteristics of the air interface, being a shared medium, makes the development of a suitable scheduling mechanism for wireless MAC protocols quite challenging. If the operation of the MAC protocol is to be optimized, a complete synchronization between scheduling and the other MAC functions (including random access and data transmission) needs to be achieved. Hence, our scheduling architecture has to flawlessly cooperate with the wireless MAC protocol to make the most efficient use of the *shared* wireless resources.

A key element of our proposed QoS architecture is known as *Fair Scheduling* or *Fair Queueing* (FQ). Fair scheduling algorithms have received much attention in recent years because of their ability to provide a wide range of QoS guarantees to end users. Due to their importance to the operation of our QoS architecture, the second part of this research effort combines novel mathematical analysis with new discrete-event simulation experiments to extensively study the performance characteristics and properties of such scheduling policies. When dealing with performance evaluations, we particularly concentrate on what makes such policies fair and allows them to protect against misbehavior and congestion scenarios. This step of analytical study is a necessary first step for making correct decisions about traffic engineering and resource allocations in future networks.

1.1 IETF QoS Architectures for the Wired Internet

Part of the effort to provide QoS support for multimedia applications over the Internet led to the development of two primary QoS models by the *Internet Engineering Task Force* (IETF): The *Integrated Services* (Intserv) model [1] and the *Differentiated Services* (Diffserv) model [3].

The Intserv model resembles, in many aspects, the QoS architecture in *Asynchronous Transfer Mode* (ATM) networks. The fundamental assumption is that resources (e.g., bandwidth and buffer space) must be explicitly reserved for each real-time application to provide that application with a specific QoS level. A *Resource Reservation Protocol*

(RSVP) is defined to allow applications to reserve resources explicitly in each router at the call set-up time [2].

Call Admission Control (CAC) routines determine whether a request for resources can be granted or not based on the knowledge of total network resources and the flows that have already been setup. Unfortunately, the Intserv model does not scale up very easily. Hence, it can only be used in small networks or enterprise *Virtual Private Networks* (VPNs).

The other QoS model is the Diffserv model. Diffserv is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to specific *Behavior Aggregates* (BAs), with each BA being identified by a single *Diffserv Code Point* (DSCP).

Users request a specific performance level by marking the DSCP field of each packet they send with a specific value. Within the core of the network, packets are treated within aggregates according to their DSCP value. Scalability is the salient feature of the Diffserv framework, which allows it to be deployed in very large networks. This scalability is achieved by forcing much of the complexity out of the core of the network into boundary devices which process much smaller volumes of traffic, and by offering services for aggregated traffic rather than on a per flow basis.

The wireless QoS architecture we develop in this dissertation integrates well with both the Intsev and Diffserv architectures. This gives our architecture more flexibility and provides a more realistic solution to the QoS problem at hand.

1.2 Broadband Wireless Networks: An Overview

Associated with the growth of the Internet is the exponential growth of wireless technologies such as that of cellular communications, *Broadband Wireless Access* (BWA) systems, and *Wireless Local Area Networks* (WLANs).

In early 2003, a boom in the sales of IEEE 802.11b WLAN products, also known as Wi-Fi™, started to take place. The IEEE 802.11b WLAN standard (with its successors IEEE 802.11a and IEEE 802.11g)[1] is designed to emulate a wireless Ethernet. It uses the unlicensed industrial 2.4 GHz frequency band to enable multiple computers and portable devices to connect to one or more wireless hubs, thus gaining access to the Internet.

IEEE 802.11b allows for the wireless transmission of approximately 11 Mbps of raw data at indoor distances from several dozen to several hundred feet and outdoor

[1]IEEE 802.11g is a newer standard that is backwards compatible with IEEE 802.11b. Several related IEEE protocols address security, Quality of Service, and adaptive signal use, such as IEEE 802.11e, h, and i, among others.

distances of several to tens of miles. The IEEE 802.11a uses the 5 GHz band, and can handle 54 Mbps at typically shorter distances.

Wi-Fi^{TM}networks are now heavily deployed as public short-range wireless access networks, such as those found at airports, hotels, conference centers, coffee shops and restaurants. Several companies (such as Boingo, Surf and Sip, T-Mobile HotSpot, and Wayport) currently offer paid hourly, session-based, or unlimited monthly access via their deployed networks around the U.S. and internationally.

The growth in wireless technologies has also touched on broadband wireless access (BWA) networks, which are designed to deliver high-speed data and multimedia services to stationary wireless end users. Such systems are developed as a wireless competitor to broadband Cable and *Digital Subscriber Line* (DSL) access networks. Major component and equipment manufacturers such as Intel, Nokia, and Fujitsu have recently indicated they will support WiMAX (*Worldwide Interoperability for Microwave Access*), which promotes the IEEE 802.16a standard for BWA networks. The IEEE 802.16a standard has a range of up to about 30 miles with data transfer speeds of up to 70 Mbps.

Third generation (3G) cellular systems are also in the beginning processes of providing high data rates to cellular phone subscribers. The proliferation of camera phones that can exchange not only email and Web content but also multimedia messages is a clear evidence of this.

In short, the dream of Internet access (or access to information) anywhere anytime is becoming a reality, and the wireless technology is to be given the credit.

1.3 Research Motivation, Results and Applications

Our research is motivated by the fact that implementing QoS guarantees in packet switched networks is a necessary prerequisite for the efficient support of multimedia services over such networks. We believe that this topic requires extensive research on both the qualitative and the quantitative sides to achieve the most suitable QoS architecture.

We are also motivated by the fact that an ideal QoS architecture needs to seamlessly support the same QoS constraints across heterogeneous types of networks including those designed for the wired and the wireless environments. Our emphasis would be directed towards broadband wireless access networks in which stringent requirements are imposed on the QoS architecture due to the numerous limitations of the wireless channel. However, our research keeps in mind that such QoS provisioning needs to integrate well with corresponding QoS architectures for the wired part of the Internet.

We also believe that designing a proper scheduling mechanism for such networks while understanding its performance characteristics is a key first step to proper im-

plementation of the desired QoS architecture. We perform a combination of analytical and simulation studies to investigate the QoS performance aspects of our scheduling mechanism suitable for broadband wireless access (BWA) networks.

The results we obtained from this research effort are briefly summarized below:

1. We designed a new and efficient scheduling architecture to support bandwidth and delay QoS guarantees for the IEEE 802.16a broadband wireless access (BWA) standard. Our design objectives were simplicity and improved network performance. The architecture we developed supports various types of traffic including constant bit rate, variable bit rate (real-time and non-real-time) and best effort.

2. We introduced a new analysis method to statistically model the behavior of *fair scheduling* algorithms, which represent the main building block for our wireless QoS scheduling architecture. This analysis method allows us to derive upper and lower bounds on mean packet delay and mean buffer occupancy experienced by packets traveling through such scheduling policies.

3. We also developed extensive discrete-event simulation models and experiments to validate our analytical results and to further complement our analysis by providing more insights into the operation of fair scheduling algorithms. We derived some interesting results that otherwise cannot be obtained using analytical methods due to the involved complexity.

Our research finds direct application in BWA systems (which are mainly dependent on the IEEE 802.16a standard). Such systems offer attractive features over other last-mile access technologies, such as ease and speed of deployment, fast realization and revenues, and low infrastructure cost. Hence, these systems may find a large market share in the telecommunications sector.

In a similar way, our research can be extended to WLANs, such as those of IEEE 802.11b and IEEE 802.11a. Such networks are finding more and more acceptance in today's agile and more mobile working environment.

The results of our research are also expected to find application in third generation broadband wireless systems that aim to provide a wealth of multimedia content to its users on the move. A large portion of the wireless spectrum is already allocated to this technology, which promises huge benefit streams for wireless operators.

Other areas that can benefit from our proposed QoS architecture are Hybrid Fiber Coax (HFC) networks used by Cable System providers. Such last-mile systems exhibit many similarities to BWA networks and many of the concepts discussed here apply to HFC systems as well.

On another note, our performance investigation of fair scheduling algorithms, which represents the bulk of this dissertation, can also benefit the research on QoS architectures for the wired part of the Internet. To mention just one example, consider the efforts of the IETF to standardize Layer 2 transport over IP/MPLS backbone networks, in which users can lease *Multi-Protocol Label Switching* (MPLS) virtual connections and be guaranteed a certain level of QoS support [4].

In such a scenario, customers who formerly used to set up virtual private networks (VPNs) using Layer 2 point-to-point data link layer connectivity using ATM or *Frame Relay* virtual circuits can now lease MPLS virtual connections instead. Such MPLS Layer 2 virtual connections provide a basis for building VPNs that mainly use the IP protocol to accommodate Internet traffic.

Such setup is ideal for Internet Service Providers (ISPs), who are looking for a solution where a single IP-based network can provide both Layer 2 and Layer 3 services to end users. Using IP/MPLS networks will improve reliability and scalability, and offer users a range of services including VPNs based on leased MPLS links or a combination of MPLS and IP services.

For an MPLS backbone network to replace Frame Relay and ATM virtual connections in the next-generation Internet, QoS provisioning in IP/MPLS networks and the ability to manage a reliable network using MPLS Traffic Engineering should be the decisive factors for such networks to flourish.

It is expected that fair scheduling algorithms will constitute the main part of this QoS management procedure, and in such a scenario, our statistical study of fair scheduling algorithms gains even more importance, since it represents a first step to traffic engineering of service providers' backbone IP/MPLS networks.

1.4 Thesis Organization

The rest of this dissertation is organized as follows. Chapter 2 provides background on earlier work related to this research and explains some of the main wireless standards available nowadays. Chapter 3 describes our proposed QoS scheduling architecture for broadband wireless access (BWA) networks. Chapter 4 introduces the mathematical notation and assumptions used in the performance study of fair scheduling algorithms. In Chapter 5, we present a new analysis method that results in upper and lower bounds on mean waiting times experienced under fair scheduling algorithms, and in Chapter 6 we continue our performance study of such scheduling systems using simulation. We conclude in Chapter 7 by a quick summary of the contributions of this research effort and provide some pointers to possible future work.

1.5 Published Papers

Below is a list of papers that have been published or submitted for publication as part of this research:

- Mohammed Hawa and David W. Petr, *"Stochastic Evaluation of Fair Queueing Systems,"* Submitted to IEEE/ACM Transactions on Networking, August 2003.
- Mohammed Hawa and David W. Petr, *"M/G/FQ: Stochastic Analysis of Fair Queueing Systems,"* IEEE 2nd International Conference on Networking, pp. 368-381, 2002.
- Mohammed Hawa and David W. Petr, *"Quality of Service Scheduling in Cable and Broadband Wireless Access Systems,"* Tenth International Workshop on Quality of Service, pp. 247-255, 2002.

Chapter 2

Background

IN recent years, several last-mile high-speed technologies have been explored to provide Internet access and multimedia services to end users [5]. Most notable of those technologies are *Hybrid Fiber Coax* (HFC) cable networks, *Digital Subscriber Line* (DSL), Satellite Access, and fixed *Broadband Wireless Access* (BWA) systems. We opted for BWA systems to provide a case study for our research on extending QoS to the wireless part of the Internet.

The de facto standard for delivering broadband services over fixed BWA systems is a fairly recent protocol, called IEEE 802.16 (also known as WiMAX), which defines the wireless MAC protocol to be used in such systems. The IEEE 802.16 standard was developed as a consolidation of two proposals, one of which was based on the *Data Over Cable Service Interface Specifications* (DOCSIS), which defines the MAC protocol for HFC networks. It is worth mentioning at this point that the two standards IEEE 802.16 and DOCSIS hold many striking similarities. The reader will be able to verify this after examining Sections 2.1 and 2.2 below.

We start this chapter by presenting a quick survey of IEEE 802.16 and DOCSIS standards. The survey is intended to be short but comprehensive. It will be helpful in describing our wireless QoS architecture in the next chapter. We also provide some background on scheduling algorithms and explain what makes a scheduling algorithm *fair*. Such concepts are relevant to the performance analysis we develop in Chapters 4 and 5.

2.1 Data Over Cable Service Interface Specifications

HFC cable networks have been mainly used in the past to deliver broadcast-quality TV signals to homes. The wide availability of such systems and the extremely wide bandwidth they provide allows extending their functionality to deliver high-speed broadband data signals to end users [6]. To provide such support, the Data Over

Cable Service Interface Specifications (DOCSIS) protocol [7, 8] was developed by a group of major cable operators called Cable Labs. DOCSIS was later adopted by the ITU and is now supported by many vendors. Versions 1.0 and 1.1 of DOCSIS were completed by 1999, and version 2.0 was introduced in early 2002.

DOCSIS assumes an architecture in which a headend, called a *Cable Modem Termination System* (CMTS), controls the operations of many terminating *Cable Modems* (CMs) at subscriber locations. The medium between the CMTS and the different CMs is a two-way shared medium, in which downstream channels carry signals from the headend to users and upstream channels carry signals from users to the headend. Upstream and downstream channels in DOCSIS are separated using Frequency Division Duplex (FDD). DOCSIS defines both the physical layer and the MAC protocol to be used on these channels. DOCSIS supports a downstream data rate of up to 27 Mb/s shared among data cable users. In the upstream, data rates up to 10 Mb/s can be supported.

A CM normally tunes to one upstream channel and an associated downstream channel. Each upstream channel is inherently a shared medium, and the CMTS controls access of the CMs to such a medium in an orderly manner by means of the MAC protocol. The main access scheme in DOCSIS 1.0 and 1.1 is time division multiple-access (TDMA). DOCSIS 2.0 also allows frequency division multiple-access (FDMA) and synchronous code division multiple-access (S-CDMA) to complement the original TDMA access scheme. Each upstream channel is further divided into a stream of fixed-size time minislots.

The DOCSIS MAC protocol utilizes a request/grant mechanism to coordinate transmission between multiple CMs. If a CM needs to transmit anything on the upstream channel, it first *requests*, from the CMTS, an opportunity to transmit a certain amount of data. The CMTS is then responsible for allocating such a transmission opportunity (called a *data grant*) in the next upstream frame(s) if capacity is available. Periodically, the CMTS sends a *bandwidth allocation map* (MAP) message over the downstream channel to indicate to the CMs the specific time minislots allocated to them as their corresponding upstream transmission opportunities (see Figure 2.1). As a result of reserving bandwidth, the CMs are guaranteed a collision-free transmission. Besides indicating the transmission opportunities for the different CMs, the MAP message indicates in which time intervals the different CMs are allowed to send their requests for transmission. This reservation interval is a contention interval in which collisions may actually happen. A contention resolution algorithm is used to resolve such collisions. DOCSIS uses the simple *binary exponential backoff* algorithm for contention resolution. Requests for transmission can also be piggybacked on data packets transmitted by the CMs on the upstream channel.

DOCSIS supports fragmentation and concatenation of data packets. Fragmentation happens when the CMTS provides a data grant to a CM that is smaller than the one

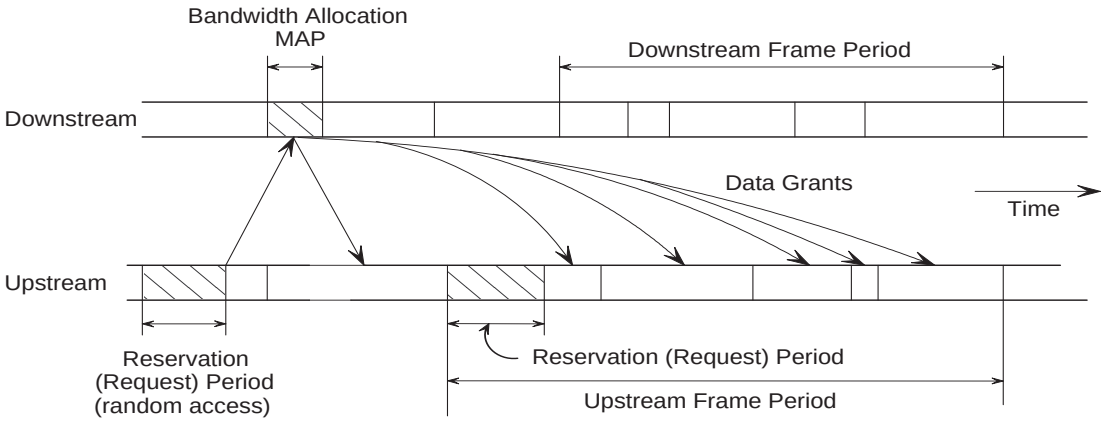

Figure 2.1: The request/grant mechanism used by the DOCSIS MAC protocol.

the CM actually requested. In such a case, the CM fills the partial grant it receives with the maximum amount of data possible, and sends the rest of the data payload in the next allocated data grant.

To support QoS, DOCSIS 1.1 introduces the concept of *service flows*. At least one service flow must be setup between any particular CM and the CMTS to carry best-effort traffic. However, to support other types of traffic, the CM may opt to set up multiple service flows to the CMTS with each flow having its own characteristics and traffic parameters.

An upstream service flow in DOCSIS 1.1 and DOCSIS 2.0 can be classified within one of the following Upstream Service Flow Types: *Unsolicited Grant Service* (UGS), *Real-Time Polling Service* (rtPS), *Non-Real-Time Polling Service* (nrtPS), *Best Effort* (BE) and *Unsolicited Grant Service with Activity Detection* (UGS/AD). The way DOCSIS treats each of those service flow types is explained in Section 2.3.

2.2 IEEE 802.16 Broadband Wireless Access

The IEEE 802.16 standard was developed for BWA systems to provide high-speed data access to subscribers [9, 10]. It was formally approved by the IEEE Standards Association in 2001.

IEEE 802.16 supports a metropolitan area network architecture. It assumes a point-to-multipoint topology, with a controlling base station (BS) that connects subscriber stations (SS) to various public networks linked to the BS. The BS and SSs are stationary and one SS typically serves one business or residential building.

The IEEE 802.16 standard defines a connection-oriented MAC protocol similar to that of DOCSIS. The MAC layer uses wide channels for the downstream and upstream channels, which are separated using either Frequency Division Duplex (FDD), as in

DOCSIS, or using Time Division Duplex (TDD). The access mode for the upstream channel is Time-Division Multiple Access (TDMA).

IEEE 802.16 utilizes contention and piggybacking, as in DOCSIS, to send requests to the BS for transmission opportunities on the upstream channel. The BS is the one responsible for assigning such transmission opportunities to different SSs and also for assigning a certain contention interval where such reservations can be made. IEEE 802.16 uses a *binary truncated exponential backoff* as its contention resolution protocol and maintains the concept of a bandwidth allocation MAP as in DOCSIS. Fragmentation and concatenation of data packets are also allowed.

To support QoS, IEEE 802.16 maintains the concept of a service flow. The Upstream Service Flow Types defined in IEEE 802.16 are, again: *Unsolicited Grant Service* (UGS), *Real-Time Polling Service* (rtPS), *Non-Real-Time Polling Service* (nrtPS) and *Best Effort* (BE).

An extra feature in IEEE 802.16, not available in DOCSIS, is that a SS is allowed to request transmission opportunities either as Grants per Connection (GPC), which is exactly the way DOCSIS works, or as Grants per Subscriber Station (GPSS), in which a SS requests transmission opportunities as a bundle for all the service flows it is maintaining. The SS then holds the responsibility for reassigning the received transmission opportunities between the different service flows. This allows hierarchical and distributed scheduling to be used and is not supported by DOCSIS.

2.3 QoS Service Flows in DOCSIS and IEEE 802.16

Both IEEE 802.16 and DOCSIS define a number of service flow types that should be treated appropriately by the MAC protocol scheduling process. Those service flow types are identical for both IEEE 802.16 and DOCSIS and are explained below.

2.3.1 Unsolicited Grant Service (UGS) Flows

UGS is designed to support real-time service flows that generate fixed size data packets on a periodic basis, such as Voice over IP. The service offers fixed size *unsolicited* data grants (transmission opportunities) on a periodic basis. This eliminates the overhead and latency of requiring the SS to send requests for transmission. In UGS, the SS is prohibited from using any contention requests and the BS does not provide any *unicast* request opportunities[1] for the SS. Piggyback requests are also prohibited in UGS.

[1] A unicast request opportunity is an interval of the upstream channel in which only one particular SS is allowed to send a bandwidth request to the BS. This is different from the contention interval in which many SSs contend to transmit their bandwidth requests.

The key service parameters for UGS service flows are: *Unsolicited Grant Size*, *Grants Per Interval*, *Nominal Grant Interval* and *Tolerated Grant Jitter*. The ideal schedule for enforcing such parameters (see Figure 2.2) is defined by a *Reference Time* t_0, with the desired transmission times being $t_i = t_0 + i * interval$, where *interval* is the Nominal Grant Interval. The actual grant times t'_i must be in the range $t_i \leq t'_i \leq t_i + jitter$, where *jitter* is the Tolerated Grant Jitter. When multiple grants per interval are requested, all grants must be within this jitter interval.

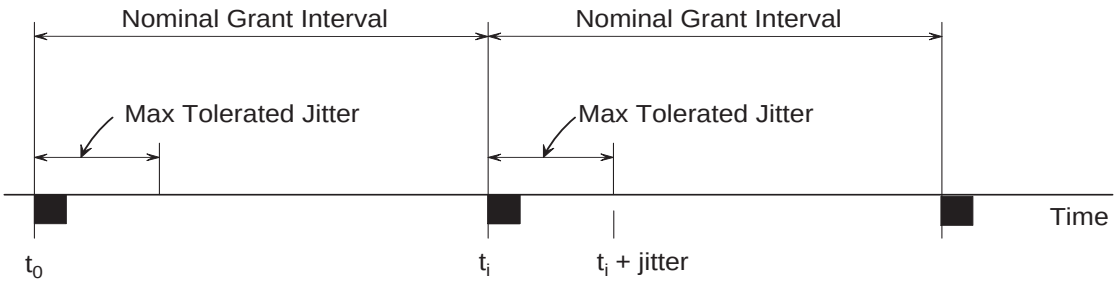

Figure 2.2: Data Grants for one UGS Service Flow.

2.3.2 Real-Time Polling Service (rtPS) Flows

rtPS is designed to support real-time service flows that generate variable size data packets on a periodic basis, such as MPEG video. The service offers periodic unicast request opportunities, which meet the flow's real-time needs and allow the SS to specify the size of the desired grants. The SS is prohibited from using any contention or piggyback requests. The key service parameters here are: *Nominal Polling Interval*, *Tolerated Poll Jitter* and *Minimum Reserved Traffic Rate*. The ideal schedule for enforcing such parameters is very similar to that for UGS service flows.

2.3.3 Non-Real-Time Polling Service (nrtPS) Flows

nrtPS is designed to support non-real-time service flows that require variable size data grants on a regular basis, such as high bandwidth FTP. The service offers unicast request opportunities (polls) on a periodic basis, but using more spaced intervals than rtPS. This ensures that the flow receives request opportunities even during network congestion. In addition, the SS is allowed to use contention and piggyback request opportunities. The key service parameters here are: *Nominal Polling Interval*, *Minimum Reserved Traffic Rate* and *Traffic Priority*.

2.3.4 Best Effort (BE) Service Flows

In BE service the SS is allowed to use contention and piggyback request opportunities, but neither periodic polls nor periodic data grants will be sent by the BS. The key service parameters for BE service flows are: *Minimum Reserved Traffic Rate* and *Traffic Priority*.

It is worth mentioning that for nrtPS and BE service flows, the standard specifies that the BS should use the Traffic Priority parameter when determining precedence in request service and grant generation. In addition, the BS must preferentially provide contention request opportunities based on priority.

2.3.5 Unsolicited Grant Service with Activity Detection (UGS/AD) Service Flows

UGS/AD is a service flow type that is supported by DOCSIS only. It is designed to support UGS flows that may become inactive for substantial portions of time (i.e., tens of milliseconds or more), such as Voice over IP with silence suppression. The service provides unsolicited grants when the flow is active and unicast polls when the flow is inactive. This combines the low overhead and low latency of UGS with the efficiency of rtPS. Though USG/AD combines UGS and rtPS, only one scheduling service should be active at a time.

It is essential that we mention at this point that even though both DOCSIS and IEEE 802.16 define the QoS service flow types that can be setup between the BS and the different SSs, they do not define or mandate a specific scheduling algorithm to be used at the BS to maintain the QoS constraints required by such service flows. This job is left entirely to the innovation of designers and implementation of service providers. Since scheduling represents an important component in supporting QoS over the wireless MAC protocol, and since the scheduler design has a great impact on the maximum utilization a MAC protocol can achieve, we will start our research (in Chapter 3) by developing a new and effective scheduling mechanism for IEEE 802.16 (and hence for DOCSIS as well) that can utilize the wireless resources as efficiently as possible while providing the necessary QoS guarantees to end users.

In addition, our scheduling algorithm will work closely with the request/grant mechanism employed by the wireless MAC protocol to reduce the possible collisions of request packets during the bandwidth reservation period. We optimize the resource reservation phase by efficiently distributing the channel among the competing nodes, thus reducing the collision probability (see Chapter 3).

2.4 Contention Minislot Allocation

Both IEEE 802.16 and DOCSIS require that each upstream frame interval include a proper contention (reservation) period. The standards, however, do not specify any method to determine the size of this reservation area and leaves that entirely to the discretion of the service provider. An appropriate choice of this contention area must reduce the number of possible collisions to shorten the contention resolution phase without *wasting* upstream bandwidth. Different strategies for allocating contention minislots will result in different performance for the entire MAC protocol, especially for the service flows that depend on contention to send their upstream requests to the BS (i.e., nrtPS and BE service flows).

No contention minislot allocation strategies have been proposed in the literature for IEEE 802.16. However, a few proposals have been made for Hybrid Fiber Coax (HFC) networks [26 – 29], although not all of them addressed the DOCSIS standard *per se*. In particular, two main approaches have been thoroughly investigated. They improve the performance of the MAC protocol by dynamically changing the number of contention minislots in successive frames based on the observed overall traffic. Such procedure typically offers improved performance as opposed to the straightforward solution of using a fixed number of contention minislots in each upstream frame.

The first of those two approaches utilizes a heuristic algorithm to compute the number of contention minislots. The algorithm is based on the observation that a simple contention resolution algorithm, such as the binary exponential back-off used by IEEE 802.16 and DOCSIS, has a maximum theoretical throughout efficiency of 33% [26]. This means that allocating approximately three request minislots for each data packet transmission expected in the upstream frame will theoretically force the random collisions in the contention area to drop to near zero. This will reduce the mean delay such requests incur to reach the BS and thus increase the throughout they observe (see also Section 3.2 for more details).

This approach is very intuitive and easy to implement, with the equations describing the algorithm being natural and straightforward. In addition, the results in [26] and [27] clearly show that such an algorithm (even with slightly different flavors) provides a noticeable improvement in the performance of the MAC protocol. The results in [26] show the improvement in throughout compared to assigning a fixed number of contention minislots in each frame, especially when the amount of contention-based traffic is significant. The improvement in throughput is shown to occur for both contention-based traffic and the combined contention-based and non-contention-based traffic. The results in [27] also show the improvement due to the dynamic algorithm in terms of mean delay required for successful transmission of bandwidth requests.

The two flavors suggested in [26] and [27] of the first approach to contention minislot allocation are mainly designed for ATM and IEEE 802.14 networks, and do not

completely fit the IEEE 802.16 or DOCSIS frameworks. This is because they do not address specific issues such as variable frame lengths and multi-priority contention traffic. In addition, one of the algorithms lacks a mechanism to properly combat congestion scenarios at the BS. Hence, we will introduce appropriate modifications to this approach to develop a more appropriate contention minislot allocation scheme for our proposed IEEE 802.16 QoS scheduling architecture (see Chapter 3 for more details).

The second approach to dynamic minislot allocation is slightly more involved. It started with the work in [28] where the authors suggest a dynamic allocation algorithm for the IEEE 802.14 standard (which uses the n-ary tree algorithm for contention resolution). This dynamic allocation scheme uses a specific formula based on empirical data to calculate the number of contention minislots in the initial frame, and then if collisions happen, it sets the number of contention minislots to *three* times the number of collisions in the last allocated minislot cluster. A slightly different approach with a similar two stage allocation system (initial number of contention minislots which is modified when collisions happen) has also been proposed in [28] for the DOCSIS binary exponential back-off contention resolution algorithm.

This approach has evolved in [29] into a more elaborate statistically optimized contention minislot allocation scheme. The new algorithm allocates contention minislots in two stages. However, in the first stage a time proportional scheme is used to estimate the necessary number of contention minislots. Once collisions start to happen, the new number of contention minislots is calculated by looking up a predetermined table of *most likely number of requests* (MLR). The simulation results in [29] demonstrate the improvement in performance achieved by such algorithm compared to using a *fixed* number of contention minislots. The improvement is shown as a drop in the mean access delay required for upstream requests to successfully reach the BS (including the ones that incur collisions) and as an increase in the throughput of the MAC protocol as observed by service flows that are dependent on contention.

Unfortunately, no comprehensive study explores the differences in performance between the different dynamic contention minislot allocation algorithms discussed above, mainly because of the different assumptions they make in terms of the infrastructure MAC protocol and the associated *contention resolution algorithm* (CRA). In addition, many of them do not adhere exactly to the DOCSIS MAC protocol and some of those algorithms require changes to the DOCSIS specifications to work correctly. This makes the comparison very difficult, if not impossible. For example, if a certain dynamic contention minislot allocation algorithm turns out to perform very well for the n-ary tree-based contention resolution algorithm, its mapping to the simpler binary exponential back-off algorithm might not be straightforward or might even result in a degraded performance.

Due to the intuitive nature of the first approach and the difficulty in implementing the second one, the scheme we develop in Chapter 3 will be based on the first approach with modifications to adapt to the minislot structure of IEEE 802.16, to account for piggybacking, to handle multi-priority traffic and to allow for variable frame lengths. The way we deal with different priorities within nrtPS and BE service flows is inspired by the work in [22] for dealing with priorities within the DOCSIS MAC protocol, although the algorithm we develop is slightly different. This is mainly because the algorithm in [22] requires modifications to the contention resolution algorithm mandated by DOCSIS, which is not an option we desire.

2.5 Fair Scheduling Algorithms

Fair scheduling or *Fair Queueing* (FQ) algorithms have received much attention in recent years because of their ability to provide a wide range of QoS guarantees to end users. Examples of well-known FQ algorithms include the Generalized Processor Sharing (GPS) policy [11], Weighted Fair Queuing (WFQ) [11, 12], Self-Clocked Fair Queuing (SCFQ) [13], Start-Time Fair Queuing (SFQ) [14], Starting Potential-based Fair Queuing (SPFQ) [15] and Weighted Round Robin (WRR) [16].

2.5.1 Classification of Fair Queueing Algorithms

Based on the way FQ algorithms are constructed, they can be classified into either *frame*-based or *sorted packet*-based algorithms. In frame-based algorithms, the time axis is divided into fixed or variable length frames, and each flow is allowed to transmit within a certain portion of this frame period corresponding to its bandwidth reservation. An example of a frame-based FQ algorithm is the WRR policy.

The sorted packet-based algorithms, on the other hand, are mainly derived from a packet-by-packet implementation of the GPS algorithm suggested in [11]. Such algorithms provide QoS guarantees to the supported flows by assigning certain timestamps to arriving packets and then serving those packets in increasing order of their timestamps. The timestamps (either *virtual finish times* or *virtual start times*) are assigned based on a system-wide function called the *virtual time*, denoted by $v(t)$, which tracks the progress of work in the scheduling system.

To elaborate more, let us denote by r_k the minimum *reserved service rate* (in bits/second) associated with each flow k in the set of all flows $\mathbf{K}$ supported by a sorted packet-based scheduler. Each arriving packet i of flow k, p_k^i, with arrival time a_k^i and length L_k^i is stamped by a *virtual start time* S_k^i and a *virtual finish time* F_k^i computed as follows,

$$F_k^i = \frac{L_k^i}{r_k} + S_k^i,\ S_k^i = \max(F_k^{i-1}, v(a_k^i)), \qquad i \geq 1,\ k \in \mathbf{K} \tag{2.1}$$

where $F_k^0 = 0$ and $v(a_k^i)$ is the value of $v(t)$ at the time of packet p_k^i arrival. Sorted packet-based FQ algorithms are divided into *Earliest Finish Time First* (EFTF) policies, in which packets are scheduled in the increasing order of their virtual finish times (e.g., WFQ, SCFQ and SPFQ), and *Earliest Start Time First* (ESTF) policies, in which packets are scheduled in the increasing order of their virtual start times (e.g. SFQ).

It is important to note here that even though the different FQ algorithms mentioned above were originally designed to emulate GPS behavior, the techniques they use to calculate the virtual time $v(t)$ are different. As a result, their implementation complexities and *fairness bounds* are also different [18].

For example, in WFQ the virtual time $v(t)$ is defined as a piece-wise linear function with a slope that changes whenever the set of backlogged flows $B(t_1, t_2)$ in an associated reference GPS system changes. Mathematically, such virtual time is computed as follow [11],

$$v(t_2) - v(t_1) = \frac{C}{\sum\limits_{k \in B(t_1, t_2)} r_k} \cdot (t_2 - t_1) \tag{2.2}$$

where C is the total output link capacity in bits/s and $[t_1, t_2]$ is an arbitrary subinterval of a busy period of the associated reference GPS system.

To reduce the complexity of WFQ, many approximations of $v(t)$ that are less computationally demanding were suggested. As an example, SCFQ uses the *virtual finish time* tag of the packet receiving service at any time t as an estimate of $v(t)$, i.e., $v(t) = F_k^i$ during the interval in which packet p_k^i is in service. On the other hand, SFQ uses the *virtual start time* tag to approximate virtual time, i.e., $v(t) = S_k^i$ during the interval in which packet p_k^i is in service.

More elaborate schemes, such as the SPFQ algorithm, uses a piecewise linear function of time to represent $v(t)$ as follows: $v(t_2) = v(t_1) + (t_2 - t_1)$. The algorithm re-calibrates $v(t)$ periodically (every time a packet finishes service) to a value equal to the minimum *virtual start time* currently in the queue. This makes sure the behavior of SPFQ approximates that of WFQ.

2.5.2 The Bounded Fairness Criterion of FQ Algorithms

Let us denote by $W_k(t_1, t_2)$ the aggregate service (in bits) received by flow k, $k \in \mathbf{K}$, during the time interval $[t_1, t_2]$. $W_k(t_1, t_2)/r_k$ is then the *normalized service* provided to flow k during that time interval.

At any time t, a flow maybe *backlogged* or otherwise *absent*. A flow is backlogged if its corresponding buffer contains one or more data packets that are waiting to be served or are in service. We denote by $B(t)$ the set of flows which are backlogged at time t and by $B(t_1,\, t_2)$ the set of flows which are backlogged during the entire interval $[t_1,\, t_2]$. We also denote by $A(t_1,\, t_2)$ the set of flows that are absent during the entire interval $[t_1,\, t_2]$.

A *scheduling algorithm is said to be fair if the difference in normalized services received by different backlogged flows in the scheduler is bounded* (by a *fairness bound* Ψ) *for all intervals of time* [13], where the value of Ψ is specific to the scheduling algorithm under consideration. In other words, a scheduling algorithm is fair if the following condition applies,

$$\left|\frac{W_k(t_1,t_2)}{r_k} - \frac{W_j(t_1,t_2)}{r_j}\right| \leq \Psi, \qquad j,k \in B(t_1,t_2) \tag{2.3}$$

where $B(t_1,t_2)$ is the set of flows which are backlogged[2] during the entire time interval $[t_1,t_2]$. For example, the fairness bound for SCFQ and SFQ is given by $\Psi = L_k^{\max}/r_k + L_j^{\max}/r_j$, where $L_k^{\max}$ and $L_j^{\max}$ are the maximum packet lengths of flows k and j, respectively.

In sorted packet-based schedulers, an alternative definition of fairness is also possible. To arrive at such a definition we notice that such FQ algorithms maintain, in addition to the virtual time (also called *system potential*), a *connection potential* $v_k(t)$ associated with each flow $k \in \mathbf{K}$. The connection potential keeps track of the amount of normalized service *received* by that connection, and is mathematically defined as follows,

$$v_k(t_2) = \begin{cases} v_k(t_1) + W_k(t_1,t_2)/r_k\,, & k \in B(t_1,t_2) \\ v(t_2), & k \in A(t_1,t_2) \end{cases} \tag{2.4}$$

Connection potentials can be used to generate timestamps for the packets queued in a FQ system. In an EFTF FQ system, for example, the Nth packet in queue k has the timestamp (virtual finish time) of [20],

$$v_k(t) + \sum_{n=1}^{N} \frac{L_k^n}{r_k} \tag{2.5}$$

[2]It is worth mentioning that in some FQ algorithms, such as WFQ, the condition $k \in B(t_1,t_2)$ in (2.3) refers to flows being backlogged in the reference system maintained by such FQ algorithms rather than the actual packet-by-packet system.

where L_k^n is the length of the nth packet queued in buffer k at time t. We define the *service lag* of a flow k, denoted by $\delta_k(t)$, as the difference between the system potential $v(t)$ and the connection potential $v_k(t)$ of flow k at any time t, i.e.,

$$\delta_k(t) = v(t) - v_k(t), \qquad k \in \mathbf{K} \tag{2.6}$$

The fairness bound Ψ for sorted packet-based FQ algorithms can be translated into an equivalent fairness bound in terms of the service lag characterized by (2.6). Such a fairness bound can be written as follows (see Appendix A),

$$0 \leq \delta_k(t) \leq \psi_k(t), \qquad k \in \mathbf{K} \tag{2.7}$$

where $\psi_k(t)$ is a fairness bound specific to the scheduling algorithm under consideration[3]. In SCFQ, for example, the bound on the service lag of flow k is given by $\psi_{k,SCFQ}(t) = L_k(t)/r_k$, where $L_k(t)$ is the length of the packet in queue k that finishes service after time t [13].

It is easy to see that the mean of the service lag $\delta_k(t)$, denoted by $\overline{\delta_k}$, has the following bound,

$$0 \leq \overline{\delta_k} \leq \overline{\psi_k}, \qquad k \in \mathbf{K} \tag{2.8}$$

where $\overline{\psi_k}$ is the mean of $\psi_k(t)$. Another important parameter that we will use in our analysis is the lag between the connection potentials of two different flows, defined as follows,

$$\begin{aligned} &\delta_{kj}(t) = v_k(t) - v_j(t) \\ &= [v(t) - v_j(t)] - [v(t) - v_k(t)] = \delta_j(t) - \delta_k(t), \quad k, j \in \mathbf{K} \end{aligned} \tag{2.9}$$

The $\delta_{kj}(t)$ parameter is a random variable with a mean that is bounded in a FQ system by (cf. (2.8) and (2.9)),

$$-\overline{\psi_k} \leq \overline{\delta_{kj}} \leq \overline{\psi_j}, \qquad k, j \in \mathbf{K} \tag{2.10}$$

It is worth mentioning that if we introduce some reasonable, although not mathematically rigorous, assumptions (or approximations) we can dramatically enhance the bounds on $\overline{\delta_k}$ and $\overline{\delta_{kj}}$ in (2.8) and (2.10), respectively. We introduce the assumptions here, and defer the discussion on why they are reasonable until Chapter 5.

[3]The results derived in this dissertation work just as well for FQ algorithms that have a fairness bound of the form $-a_k \leq \delta_k \leq b_k$. We only need to define a new equivalent fairness bound given by $0 \leq \delta'_k \leq a_k + b_k = \psi'_k$.

The first approximation allows us to derive a tighter bound on $\overline{\delta_k}$, and requires a more careful consideration of how FQ systems work. Notice from the definition of a connection potential in (2.4) that when flow k is absent, its connection potential becomes equal to the system potential (i.e., the service lag becomes zero) and remains that way until the flow is backlogged again. In other words, we can write,

$$0 \leq \overline{\delta_k} \leq \overline{\psi_k} \cdot \Pr\left[k \in B\right], \qquad k \in \mathbf{K} \tag{2.11}$$

where $\Pr\left[k \in B\right]$ is the probability of flow k being backlogged. Although we cannot find the exact value of such a probability, we can find an upper bound for it, which would still be useful in (2.11).

To proceed, we remember that in a general G/G/1 queuing system, the probability of the queuing system being backlogged is equal to the probability of the server being busy[4], which in turn is equal to the utilization factor $\rho = \lambda\overline{X} = \lambda\overline{L}/C$. Using the same argument in our FQ system while *assuming* that flow k is guaranteed an *equivalent* server of *minimum* capacity of r_k (see reasoning in Chapter 5), we get,

$$\Pr\left[k \in B\right] \leq \lambda_k \frac{\overline{L_k}}{r_k}$$

where $\overline{L_k}$ is the mean length of flow k data packets, and λ_k is the mean arrival rate at flow k. We will use ρ'_k to represent the quantity $\lambda_k\overline{L_k}/r_k$ to express the notion of a new utilization factor of flow k under an equivalent server of capacity r_k. Hence, the mean service lag of flow k has the upper and lower bounds of,

$$0 \leq \overline{\delta_k} \leq \rho'_k\overline{\psi_k} = \lambda_k \frac{\overline{L_k}}{r_k}\overline{\psi_k}, \qquad k \in \mathbf{K} \tag{2.12}$$

We can make another approximation by noticing that the service lag $\delta_k(t)$ is a random variable that has an unknown distribution. Although such a distribution will, most likely, be dependent on how the specific FQ algorithm works, we can assume that the distributions of $\delta_k(t)$ and $\delta_j(t)$, $k, j \in \mathbf{K}$, are *similar*. We refer to two distributions as similar if they have the forms $f_\delta(\delta)$ and $(1/\tau)\, f_\delta(\delta/\tau)$, where τ is a constant. For example, if it turns out that $\delta_k(t)$ has a uniform distribution extending between 0 and $\psi_k(t)$, it is reasonable to assume another uniform distribution for $\delta_j(t)$ but this time extending between 0 and $\psi_j(t)$. This is justified by the fact that different flows in the FQ system are treated in the exact same way except for the amount of reserved bandwidth they receive.

[4] $\Pr[\text{backlog}] = 1 - p_0 = \rho = \lambda\overline{X}$

If such an assumption holds then the means of $\delta_k(t)$ and $\delta_j(t)$ will also be related (see Appendix B). This relation is expressed as follows: If $\overline{\delta_k} = \alpha\,\rho'_k\overline{\psi_k}$, where $0 \leq \alpha \leq 1$, then $\overline{\delta_j} = \alpha\,\rho'_j\overline{\psi_j}$. Hence, we can say that $\overline{\delta_{kj}}$ is given by,

$$\overline{\delta_{kj}} = \overline{\delta_j} - \overline{\delta_k} = \alpha\left(\rho'_j\overline{\psi_j} - \rho'_k\overline{\psi_k}\right), \qquad k, j \in \mathbf{K} \tag{2.13}$$

And a new tighter upper bound on $\overline{\delta_{kj}}$ is then derived by setting $\alpha = 1$,

$$\overline{\delta_{kj}} \leq \rho'_j\overline{\psi_j} - \rho'_k\overline{\psi_k}, \qquad k, j \in \mathbf{K} \tag{2.14}$$

Chapter 3

Quality of Service Scheduling Architecture

IN this chapter, we describe a new and efficient scheduling architecture to support bandwidth and delay QoS guarantees for both IEEE 802.16 and DOCSIS standards. Our design goals are simplicity and improved network performance. The architecture we develop here supports various types of traffic including constant bit rate, variable bit rate (real-time and non-real-time) and best effort.

It is worth mentioning that a few proposals have already been devised to support QoS in HFC networks [22 – 25]. However, most of those proposals do not specifically address the QoS requirements of DOCSIS (or IEEE 802.16). For example, in [22] the authors propose a multi-tiered priority-based HFC scheduler, which supports contention-based traffic. The proposed scheduler, however, has no provision for delay-sensitive traffic such as UGS and rtPS service flows. To the best of our knowledge, only the work in [25] addresses the DOCSIS 1.1 standard *per se*, but it also falls short of supporting all the service flow types defined in DOCSIS (it deals only with UGS and BE services). Its treatment of UGS service is also problematic since it does not provide any guarantees in terms of Tolerated Jitter for such UGS service flows.

The scheduling architecture we present here is the first one to be proposed for the new IEEE 802.16 standard and is also the first that truly addresses the QoS needs of DOCSIS. Although our scheduler supports both IEEE 802.16 and DOCSIS, our discussion will be directed more toward the IEEE 802.16 standard.

3.1 The Scheduling Architecture

Scheduling within the IEEE 802.16 BS can be essentially reduced to the process of building the bandwidth allocation MAP describing which data grants (packets) get transmitted in the next frame period. Our suggested upstream scheduling architecture for IEEE 802.16 is shown in Figure 3.1. In such architecture, requests for transmission from the different SSs are received by the BS through contention, unicast request opportunities and piggybacking (see Chapter 2). Those requests are first translated into suitable upstream transmission opportunities (data grants), which then get scheduled on a frame-by-frame basis by building a corresponding allocation MAP message that describes the usage of each frame interval.

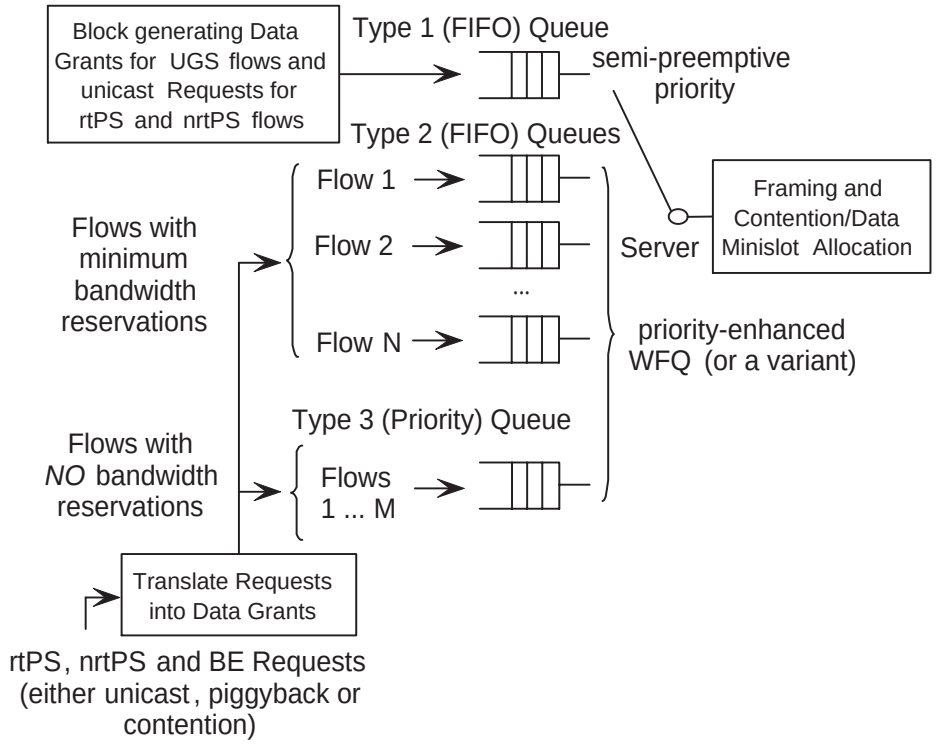

Figure 3.1: Architecture of the proposed upstream scheduler for both IEEE 802.16 and DOCSIS.

To allow for multiple QoS requirements, the scheduler keeps the data grants to be scheduled in three types of queues, which we will refer to as Type 1, Type 2 and Type 3 queues. We represent the hardware block responsible for scheduling the data grants (or creating the MAP message) in Figure 3.1 by a *server* that continuously schedules

different data grants (and unicast request opportunities) on the upstream channel. In such a representation, each data grant is treated as a *packet* that needs to find its way through the server (scheduler). When such a packet finishes service (i.e., when a data grant gets scheduled), a corresponding entry is logged in the MAP message for the next frame period. The actual transmission of the corresponding data packet does not take place, however, until the next frame starts.

In developing an understanding of our scheduling architecture, the reader should keep in mind the difference between the time axis used by the scheduler trying to build the allocation MAP and the actual transmission of data packets on the upstream channel. The scheduler in Figure 3.1 advances within a faster virtual time frame to identify the proper instants for scheduling data grants, while the upstream channel operates within the normal time frame to transmit the corresponding data packets (see Figure 3.2).

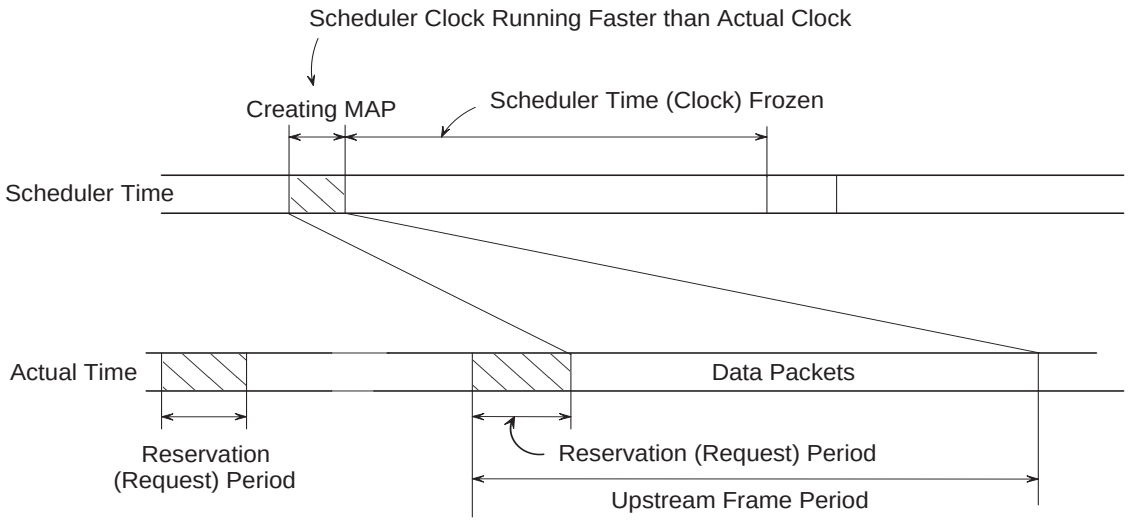

Figure 3.2: Difference between scheduler time and actual time.

Maintaining a different time axis for the scheduler allows us to use the queueing model in Figure 3.1 as if the *server* corresponds to the upstream channel. We will refer to the new time axis used by the scheduler as the *scheduler time*. More discussion on the properties of our scheduler and how it treats the different service flow types in IEEE 802.16 and DOCSIS is presented below.

3.1.1 Unsolicited Grant Service (UGS) Flows

UGS packets cannot tolerate excessive delays in their transmission. Hence, the processing of UGS flows should be decoupled as much as possible from all other flows in the scheduler. To achieve this goal a separate hardware block in our scheduler keeps track of all admitted UGS service flows by maintaining a table similar to the one shown in Figure 3.3. This table is updated whenever the *Connection Admission Control* (CAC) algorithm admits or releases a new UGS flow. The separate hardware block then uses the information in this table to periodically generate (in scheduler time) data grants that feed the Type 1 queue in the scheduler (see Figure 3.1).

Active UGS Service Flows						
Flow SID	Reference Time, t_0	Nominal Grant Interval	Tolerated Grant Jitter	Grant Size	Grants Per Interval	Other Data
1	X	X	X	X	X	X
2	X	X	X	X	X	X
3	X	X	X	X	X	X

Figure 3.3: The table maintained by UGS dedicated hardware block.

One data grant (or more) is generated per nominal interval for each active UGS service flow. The generation time of each data grant is given by $t_i = t_0 + i * interval$, where *interval* is the Nominal Grant Interval for that service flow. Each generated data grant is also marked with a delivery deadline equal to $t_i + jitter$, where *jitter* is the Tolerated Grant Jitter for such a flow. The scheduling algorithm makes sure that data grants fed to the Type 1 queue are served (scheduled) before their corresponding deadlines by providing priority to such data grants. It is important to mention that all the above times are in scheduler time discussed earlier and not the actual time.

The server provides a strict *semi-preemptive* priority to data grants in the Type 1 queue, whereby a grant undergoing service is *sometimes* allowed to complete service without interruption even if a grant of higher priority (a Type 1 grant) arrives in the meantime. This happens *only when* the newly arriving Type 1 grant can still be delivered within its deadline without the need to preempt the grant undergoing service. However, service of a grant must be interrupted (preempted) when a Type 1 grant arrives with a deadline that is earlier than the completion time of the grant in service. In such a case, the newly arriving Type 1 grant is served first and the remainder of the preempted grant is served afterwards. In IEEE 802.16, this results in the lower priority data grant being fragmented. Of course, when the server becomes free while the Type 1 queue is nonempty, Type 1 grants are always the ones that enter service first. A question here is whether preemption (when needed) should be done just before the Type 1 grant deadline or at an earlier time given that the server

knows it needs to perform preemption. Since preemption means fragmentation of a particular data grant, we suggest to preempt at a point convenient for fragmentation (e.g., at a fragment size equal to a power of 2) if possible.

For such a scheduling algorithm to work, fragmentation should be enabled for all *non*-UGS service flows in the network. Otherwise, if fragmentation must be avoided, a simpler architecture can be used in which data grants of all non-UGS service flows are limited by management functions[1] to a certain size that is always smaller than the minimum Tolerated Jitter of all UGS service flows. In this case, no UGS data grant will ever miss its deadline due to a grant being served under a simple strict *non-preemptive priority* queuing discipline. In fact, in such a design scenario, we can stop attaching deadlines to Type 1 data grants. The SS will be the one responsible for limiting the packet sizes corresponding to non-UGS flows to fit the new data grants with limited sizes.

3.1.2 Real-Time Polling Service (rtPS) Flows

There are two portions of rtPS traffic that need to be handled by the scheduler. First, there are the periodic upstream unicast request opportunities (periodic polls) to be provided for each rtPS service flow, and second, there are the actual data grants (transmission opportunities) to be allocated to such a flow.

Our scheduler treats rtPS upstream unicast request opportunities in exactly the same way as UGS data grants: A dedicated hardware block generates periodic unicast requests (polls) based on information stored in an internal table about the rtPS flows, and feeds those requests to the Type 1 queue in the scheduler. The table structure is the same as that used for UGS traffic (see Figure 3.3), but with replacing entries corresponding to data grants by entries corresponding to unicast requests (e.g., replacing Nominal Grant Interval by Nominal Polling Interval and Tolerated Grant Jitter by Tolerated Polling Jitter).

For the other portion of rtPS traffic, which is the transmission of actual data grants, we notice that a fundamental difference between UGS traffic and rtPS traffic is that UGS reserves a fixed portion of the upstream bandwidth that can only be used by that UGS service. In rtPS, however, if a service flow is inactive for a short period of time, the excess reserved capacity can be reused by other rtPS (or nrtPS and BE) flows[2]. Hence, when the scheduler is generating data grants, it should treat rtPS traffic in a different way than UGS traffic. Also, each rtPS service flow may or may not make a

[1] IEEE 802.16 allows the BS to impose a specific limit on data grant sizes of rtPS, nrtPS and BE service flows. This is done using the Maximum Traffic Burst management parameter (which has a default value of 1522 bytes).

[2] This requires using a queueing discipline that supports this feature. A straightforward solution is to incorporate a *fair queueing* algorithm in our scheduling architecture to provide this behavior.

minimum bandwidth reservation request at connection setup. The scheduler should also treat various rtPS flows differently based on the amount of bandwidth reservation they make.

Based on the above observations, after a rtPS request for transmission is received on the upstream channel, a corresponding data grant is generated and is fed to either a Type 2 or a Type 3 queue based on whether the corresponding service flow has made a minimum bandwidth reservation or not. The data grant is fed to a Type 3 queue if its corresponding service flow has no bandwidth reservation or is fed to a Type 2 queue if its flow has made such a bandwidth reservation (see Figure 3.1). The Type 3 queue in the scheduler is shared by all service flows with no explicit bandwidth reservations, while the scheduler provides a dedicated Type 2 queue for each service flow that has already requested some bandwidth guarantees from the BS.

We suggest using a Weighted Fair Queuing (WFQ) [11, 12] discipline or a simpler variant of it such as Self-Clocked Fair Queuing (SCFQ) [13] or Start-Time Fair Queuing (SFQ) [14] to handle rtPS flows fed to Type 2 and Type 3 queues. A WFQ rate (or weight) is assigned to each Type 2 queue based on the minimum bandwidth reserved for the corresponding service flow. The WFQ rate for Type 3 queue is calculated by subtracting all the reserved rates for Type 2 queues from the aggregate output link capacity (of course, after subtracting the bandwidth reserved for UGS traffic and contention minislots).

It is fair to assume that the number of flows set up with minimum bandwidth reservations will be much smaller than those with no reservations (because reserving bandwidth will require the customer to pay a higher fee). This is why aggregating all flows with no bandwidth reservations in one Type 3 queue will reduce the complexity of the underlying WFQ algorithm considerably. The choice of per service flow scheduling for Type 2 traffic, on the other hand, is adopted to provide hard bandwidth guarantees for the corresponding service flows that wish to have such guarantees.

We envision that pricing will depend mainly on the amount of minimum bandwidth reserved for a certain service flow. As explained in Chapter 2, nrtPS and BE service flows can be assigned different priority levels in the range of $0 - 7$, with higher values indicating higher priority. We envision that for nrtPS and BE service flows, the Traffic Priority parameter will be a second-level pricing criterion. In other words, priority levels can provide finer-grained pricing to be combined with the coarser-grained pricing for the amount of minimum bandwidth reservation a user makes.

Although the IEEE 802.16 standard does not assign any priority levels to rtPS service flows, we believe that since users are expected to pay more for rtPS than nrtPS and BE services, rtPS should have an implicit priority level of 8.

3.1.3 Non-Real-Time Polling Service (nrtPS) Flows

There are two differences between nrtPS and rtPS services. First, nrtPS does not depend solely on unicast requests allocated to it by the BS but also utilizes contention and piggybacking to send requests to the scheduler. Second, nrtPS flows can be assigned different priority levels while rtPS has only one implicit priority level. In all other aspects, nrtPS and rtPS service flows are identical.

Hence, we handle the periodic upstream unicast requests (polls) for nrtPS in exactly the same manner as we handled rtPS (i.e., using a dedicated hardware block feeding requests to the Type 1 queue). In addition, after the nrtPS requests are received at the scheduler, the generated data grants are also fed to a Type 2 or Type 3 queue based on whether they have a minimum bandwidth reservation or not, respectively.

Since the standard requires that higher priority service flows be given lower delay and higher buffering preference, given that they are identical in all other QoS parameters besides priority, we propose the following modification of WFQ to produce a *priority-enhanced* WFQ. If two data grants (from two different queues, whether Type 2 or Type 3 queues) have identical[3] WFQ *virtual finish times*, then the first grant to be served is not selected randomly but is chosen based on its priority level, with higher priority grants being served first. This makes sure that higher-priority grants always incur less delay.

In addition, since all service flows fed to the Type 3 queue in the scheduler have zero bandwidth reservation, priority can be further invoked by adopting a strict *non-preemptive* priority discipline in serving data grants from Type 3 queue before being handed to the WFQ global server (see Figure 3.1). Thus, Type 3 grants pass through a non-preemptive priority server first, then pass through a WFQ server in which priority may again be invoked against grants from Type 2 queues.

3.1.4 Best Effort Service (BE) Flows

BE traffic is treated exactly in the same way as nrtPS traffic except for the fact that no periodic unicast requests are scheduled for any BE service flows *unless* they are needed to satisfy the minimum reserved bandwidth for that service.

In Section 3.2 we will discuss how nrtPS and BE flows can use contention minislots to send their requests to the BS. Our only challenge at this point is that nrtPS and BE flows with minimum reserved bandwidth may not be able to send enough requests to the BS to occupy such allocated bandwidth because of possible collisions in the

[3]The probability of two packets having identical virtual finish times in our scheduler is higher than that in a general variable-length data packet infrastructure. This is because the size of any data grant in IEEE 802.16, although variable, is always a multiple of the IEEE 802.16 minislot size.

contention region (especially at high loads). To avoid this problem, we allocate extra upstream unicast request opportunities at the start of each frame period to all nrtPS and BE flows *with* minimum bandwidth reservations to allow them to at least request such a minimum bandwidth.

The reason for placing such extra unicast requests at the start of each frame period, even before the contention minislots, is an attempt to relieve the contention area by forcing some SSs to use the unicast requests and thus avoid the need for further contention. This should reduce the number of collisions in the contention area.

3.1.5 Unsolicited Grant Service with Activity Detection (UGS/AD)

To handle a UGS/AD service flow, a certain portion of the upstream channel bandwidth should be reserved for that flow. This reservation is made fixed when the service flow is active by creating a temporary entry in the UGS table, and treating that flow as if it was a UGS flow. When the flow becomes inactive, the entry in the table is temporarily blanked and instead the service flow is considered a rtPS one with its Minimum Reserved Traffic Rate parameter set to the original UGS traffic rate. This allows any excess bandwidth not used by the service flow to be utilized by other users, but still guarantees the minimum required bandwidth by the service.

3.2 Contention Minislot Allocation

In IEEE 802.16 and DOCSIS, nrtPS and BE service flows use contention to send their requests to the BS. We need to allocate an appropriate number of contention request minislots in each frame period to reduce the number of possible collisions and to shorten the contention resolution process. If done properly, this will improve the performance of the MAC protocol under varying traffic load conditions.

3.2.1 Frame Structure

In our scheduler we use a variable length upstream frame structure in order to achieve maximum scheduling flexibility and minimum transmission latency. We opt for the frame structure shown in Figure 3.4, where contention minislots are all clustered adjacently at the beginning of each upstream frame interval. This configuration allows easier implementation at both the BS and the SS because both devices have to switch to the contention mode only once at the start of each frame period. Also in such a configuration, the feedback MAP message from the BS corresponding to a cluster of minislots can be received prior to the beginning of the next frame period. This reduces latency in receiving request acknowledgments and in contention resolution, which is of great concern in the contention process.

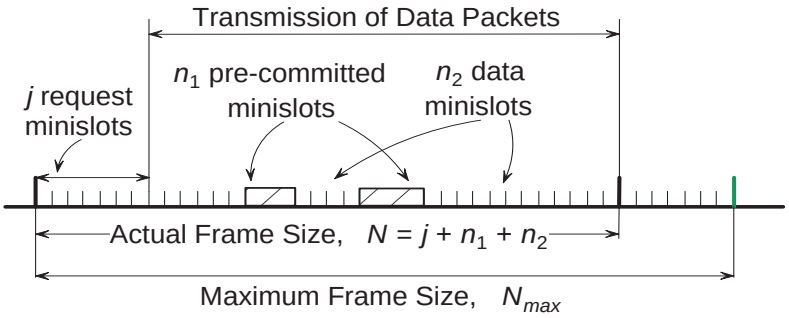

Figure 3.4: Upstream frame structure.

3.2.2 Contention Minislot Allocation

Different contention minislot allocation schemes have been suggested in the literature [22, 26 – 29]. The scheme we propose here is an extension of the mechanisms suggested in [26] and [22] with some modifications to adapt to the minislot structure of IEEE 802.16, to account for piggybacking, and to allow for variable frame lengths.

We assume that the frame length in Figure 3.4 (in units of minislots) is variable with a certain maximum limit, N_{max}. The actual frame length depends on the number of data grants pending transmission on the upstream channel. Figure 3.4 shows the case where there are only n_2 worth of nrtPS and BE data grants pending transmission. UGS grants, rtPS polls/grants and other periodic traffic that do not require contention occupy n_1 minislots of the frame interval. Such n_1 minislots are considered as pre-committed minislots in our algorithm and do not count as part of the frame. A variable number, j, of contention minislots are used in each upstream frame as request minislots. We will develop an algorithm to dynamically calculate the value of j for each frame based on different MAC loading conditions.

The algorithm can be explained as follows. Just enough contention minislots need to be created so that the average throughput (per frame) of the contention request minislots closely matches the number of new data packets that can be transmitted in a maximum frame period. We remember that using a simple contention resolution algorithm such as the random binary exponential backoff, mandated by IEEE 802.16, gives a throughput efficiency of 33% for contention minislots [26], which means that approximately *three* request minislots are needed per request (or data packet) to achieve 100% throughput efficiency. Hence, the number of contention minislots in a frame should be adjusted to satisfy the following requirement: the number of requests that can be transmitted within j contention opportunities should be equal to *three* times the number of nrtPS and BE data packets that can be transmitted in a frame with maximum length.

To transform the above statement to units of minislots, we note that a data packet may need, on average, l_d minislots to be transmitted. Also a contention request may

need, on average, l_c contention minislots to be transmitted[4]. Hence we can say that the ratio $j/l_c : (N_{\max} - j - n_1)/l_d$ should be at most 3:1. The parameters l_d and l_c can either be fixed *a priori* or measured on periodic basis during the scheduler operation.

Our algorithm should also be able to dynamically reduce the number of required contention minislots per frame as the load on the system increases. This is possible because as the traffic from certain flows become heavier; those flows can utilize piggybacking more often, and hence reduce the load on contention. Taking this into account allows us to reduce the ratio of 3:1 we originally needed for contention minislots to a smaller ratio. To quantify this effect we note that if a multi-packet batch arrives at a certain SS buffer, only the first packet in that buffer generates a contention request. However, future requests for the rest of the multi-packet batch can be sent using piggybacking. Now, assume that the average length (in units of packets) of multi-packet batches arriving at different SSs is k, then we can say that k data packets are utilized for each request that makes it through the minislot contention process. Again, the value of k can be measured during normal operation. In summary, we need to have,

$$j = \frac{3 * (N_{\max} - j - n_1) * l_c}{k * l_d}$$

3.2.3 Algorithm

We compute the number of required contention minislots in each frame period based on an estimate of the maximum number of data packets that can be transmitted in such a frame. This estimate is mainly derived from the traffic observed in the previous frame. The following algorithm dynamically calculates the number of contention minislots j_i for each frame i:

Frame 0: Set $j_0 = j_{\min}$ (Initialization)

Frame i: Let $j_i = \max\left\{\frac{3*(N_{\max} - j_{i-1} - n_{1,i-1})}{(k*l_d/l_c)},\ j_{\min}\right\}$

If $Q \geq \alpha * (N_{\max} - j_{i-1} - n_{1,i-1})$, Set $j_i = j_{\min}$

Q in the above expression is the total number of data minislots requested but not yet allocated by the BS (i.e., the aggregate length of pending data grants at the scheduler), and α is a design parameter set to 2.5 or 3.5. The condition including Q means that when there are so many outstanding minislot requests that cannot be handled within the next two frames or so, the BS should prevent SSs from sending further contention requests. This will only happen in overload (congestion) situations,

[4]A typical value of l_c is 1 minislot. It is also reasonable to assume that $l_d >> l_c$.

and will prevent the buffers in the BS from overflowing unnecessarily. It also makes sense to deny piggybacked requests in those overload situations.

3.2.4 Dealing with Priorities

Now that we know the number of contention minislots to allocate per frame, we need to divide this capacity of contention minislots between the different service flow priorities. The IEEE 802.16 standard requires a preferential treatment of higher priority traffic allowing it to have a better chance of sending requests through contention. This can be achieved by introducing a set of multiplication factors, a_d, $0 < a_d < 1$ for $d = 0, \ldots, 7$ that allow dividing the available amount of contention minislots between the different priorities based on a preference criterion decided by the service provider and, of course, related to pricing. The a_d factors should satisfy the requirement $\sum_{d=0}^{7} a_d = 1$. Hence, after calculating the total number of contention minislots per frame, j, we calculate the number of minislots, j^d, to allocate to each priority d as follows: $j^d = \lfloor j * a_d \rfloor$.

One scenario that might happen in the above minislot allocation scheme is that a group of flows with a certain priority may get more minislots allocated than their actual need. This will happen if the aggregate load of a particular priority is smaller than anticipated. To overcome this situation and to improve the operation of our allocation scheme, we incorporate another factor related to the observed traffic load of each priority in the system. More specifically, we start by computing a moving average of the observed number of contention minislots used per frame for each priority level. We denote these values by r_d for $d = 0, \ldots, 7$. These values will represent estimates of how many minislots each priority should be expected to use in the next frame period. Notice that these estimates are updated periodically.

The idea is that we want to utilize any excess amount of contention minislots for use by other priorities (preferably higher priorities). To do that, after calculating the number of minislots allocated to one priority, j^d, we compare this value to the number of minislots we should expect in the next frame for that priority, r_d. If the value of j^d is much larger than r_d, we borrow a few of the j^d minislots, say δ, and redistribute them among the contention minislots allocated to higher priorities. Such an algorithm is recursive and is illustrated below, where β and δ are design parameters.

Start: $d = 0$

While $d < 7$ do{ (we stop at $d = 7 - 1$)

If $j^d > \beta\, r_d$, Set $j^d = j^d - \delta$

And set $j^e = j^e + \delta * a_e \Big/ \sum_{f=d+1}^{7} a_f$ for all $e = d + 1, ..., 7$

}End While

Another option for distributing the contention minislots between the different priorities is to use a nested priority scheme, in which higher priority flows are allowed to use the whole contention area, while low priority flows are only allowed to use part of such a contention region. The reason we avoid this scheme is that it does not allow complete separation of the different priorities and hence cannot prevent misbehaving high priority traffic from causing undue collisions in the whole contention area, including the contention interval for low priority traffic. In our scheme, on the other hand, the region given to low priority traffic is pre-determined by the service provider using the a_d parameter (which is mainly determined by pricing) and high priority traffic cannot receive more than its allocated share of the contention interval unless the low priority traffic load is smaller than anticipated.

3.3 Buffer Management

This section deals with the problem of allocating buffer space to the different queue Types (Type 1, 2 and 3) of the scheduler to achieve minimum losses of data grants during scheduling. It is important to note at this point that losing a data grant at the scheduler due to buffer overflow does not necessarily mean the loss of the corresponding data packet itself. This is because after the BS receives a request for a data grant from the SS, it sends a signal back to the SS in the bandwidth allocation MAP indicating a pending data grant. When this data grant is lost due to buffer overflow at the scheduler, the SS will eventually timeout and will retransmit another request. This will certainly cause the SS buffer to grow monotonically during the timeout period but will not necessarily result in the loss of information. Hence, mapping data grant losses in the scheduler to actual data packet losses in the SS is not an easy task to achieve and is heavily dependent on the buffer space at the different SSs along with the utilized timeout mechanisms.

Now, returning to the scheduler architecture shown earlier in Figure 3.1, we see that data grants are treated as generic packets that are placed in different queue Types before being served by the scheduler. An important distinction we need to draw here is between the *virtual* and the *actual* meaning of each queue buffer space in such a scheduler. Virtually, the system works as if it is scheduling packets with variable lengths passing through a continuous-time server. Actually, however, when a data grant is generated, the only information that needs to be stored about such a data grant is its length (in units of minislots) and optionally its deadline (in the case of Type 1 data grants). Such information is the only information needed to construct a bandwidth allocation MAP at the end of each processing period to describe the usage of the upstream channel. The size of this data grant *information* is of fixed length whether it corresponds to a 1 KB data grant or a 4 KB data grant. This means

that the buffer space allocated to each queue type needs to be measured in units of fixed-size data grant information units rather than units of bytes.

Now, to distribute the total amount of available buffer space locations between the different queue types, we start with the single Type 1 queue. Calculating the maximum number of data grants that can accumulate in such a queue at any moment of time can be done easily because of the periodic nature of UGS traffic feeding this queue. We always allocate such a maximum number of buffer space locations to the Type 1 queue. Remember that we cannot afford to lose any data grants from such a queue since there is no other mechanism for UGS service flows to request new data grants if the scheduler loses them. For Type 2 queues, we can allocate buffer space based on a pricing criterion. Since the price paid for setting up a service flow increases with the amount of minimum bandwidth reserved for that flow and the priority level assigned to it, a service flow that reserves more bandwidth and has a higher priority level should receive a larger buffer space. The remaining buffer space after deciding on Type 1 and Type 2 queue sizes is then used for the Type 3 queue.

For buffer management of Type 2 and Type 3 queues, we suggest using the Random Early Detection (RED) and multi-priority RED schemes, respectively. RED [30] is a buffer management scheme that avoids congestion by randomly (probabilistically) dropping packets when the buffer occupancy reaches a certain limit. Multi-priority RED is just an extension of RED to support multi-priority flows sharing the same buffer, as is the case for the Type 3 queue.

The reason we suggest using RED for buffer management in our scheduler is that RED was designed to work hand-in-hand with the TCP congestion control algorithm, and hence is best suited for Internet traffic. Since IEEE 802.16 and DOCSIS are mainly Internet oriented, RED would be the best candidate for buffer management in our scheduler.

3.4 Properties and Advantages of the New Architecture

The architecture we described in this chapter supports diverse QoS guarantees for various service flow types suggested by both IEEE 802.16 and DOCSIS standards. More specifically, it supports tight delay guarantees for UGS traffic and minimum bandwidth reservations for rtPS, nrtPS and BE flows. It is worth mentioning that vendor-specific QoS parameters can also be used in IEEE 802.16 and DOCSIS. This means that users can also request QoS delay bounds for their rtPS and nrtPS service flows. Because we are using a fair queueing algorithm in our scheduler, providing such guarantees is feasible and can be implemented easily given that the service flows feeding the scheduler are properly policed (either at the BS or at the SS level).

Our architecture employs a dynamic contention minislot allocation scheme that

should improve performance under varying load conditions. Such algorithm speeds the contention phase by providing extra bandwidth for contending packets. Another main advantage of our proposed architecture is the ease in which it can be implemented in hardware. This is because the architecture is built around simple queueing blocks and very well established queueing strategies, and also because the architecture schedules packets using a *single-pass* approach, in which periodic UGS traffic and non-periodic traffic are scheduled alongside while time is progressing in one direction. This is different than software scheduler implementations, where UGS traffic is first coded into the MAP message in one pass, and then other types of traffic are scheduled in the next pass. The performance advantage of building the scheduling architecture in hardware rather than software-based alternatives is significant.

Other features of our architecture include the fact that the scheduler takes advantage of the Tolerated Jitter parameter for UGS to fit as many packets as possible in the upstream frame between UGS data grants thus avoiding fragmentation and being more efficient. The scheduler also lends itself (by virtue of design) to easier and straightforward performance analysis via classical queuing theory techniques (see Chapters 4 and 5).

Another attractive property of the proposed scheduling architecture is the ability to integrate it with both *Internet Engineering Task Force* (IETF) QoS models: Intsev and Diffserv (see Chapter 1). Our architecture is based on a *per-flow* QoS model, in which QoS parameters are requested and maintained using single flow granularity. This is exactly the same QoS model employed by Intserv, which means that the integration between our architecture and Intserv is straightforward. For example, a possible mapping between Intserv QoS traffic classes (see [1]) and IEEE 802.16 QoS service flows is explained below:

- Intserv *Guaranteed Service* can be mapped into UGS service flows.
- Intserv *Controlled Load Service* can be mapped into rtPS and/or nrtPS service flows.
- Intserv *Best Effort* can be mapped into BE service flows.

Integration with the Diffserv QoS architecture is also possible if the IEEE 802.16 BS is equipped with a mechanism to aggregate multiple incoming flows with similar QoS parameters into one single flow. The aggregation should be consistent with the different Behavior Aggregates (BAs) defined by the service provider using Diffserv in the core network. We have designed our scheduling mechanism to be compatible with the service classes defined by the Diffserv QoS model (see [3]). A possible mapping between Diffserv and IEEE 802.16 service classes is given below:

- Diffserv *Expedited Forwarding (EF)* flows can be mapped into UGS service flows. This guarantees the required level of isolation and protection necessary for EF Diffserv traffic.
- Diffserv *Assured Forwarding (AF)* flows can be mapped into rtPS and/or nrtPS service flows. Different AF subclasses can also be mapped to different priority levels within the nrtPS service type.
- Diffserv *Best Effort* flows can be mapped into BE service flows.

3.5 Scheduler Performance Aspects

We have been involved in a joint project to conduct simulation experiments that demonstrate the performance and efficiency of the wireless QoS scheduler proposed here. The experiments were set up using the OPNET simulation package. Even though OPNET does not include of-the-shelf modules that implement the wireless framework defined by the IEEE 802.16 standard, it does include an add-on module that implements the DOCSIS MAC protocol (for HFC networks), which represents a very similar framework to that of IEEE 802.16. We decided to use the DOCSIS module because many of the service flow types defined by IEEE 802.16 are already implemented in DOCSIS (see Chapter 2 for a discussion on the similarities between DOCSIS and IEEE 802.16).

Unfortunately, however, the simulation process was substantially hindered by the need to receive new updates to fix critical software bugs in the OPNET DOCSIS module. These bugs were discovered during the initial simulation experiments. Because there were no immediate feasible alternatives, we were forced to leave the process of verifying the performance of our QoS scheduling architecture (within a comprehensive IEEE 802.16 environment) for future work. The results of any such study will be provided in future publications.

In addition, a more elaborate study of the behavior of the dynamic contention minislot allocation scheme proposed for our scheduling architecture is also being planned. Specifically, we plan to study the effects of congestion on the effective throughput and delay performance of such contention minislot allocation algorithm as observed by the incoming request packets. A comparison of the performance of this mechanism with other contention minislot allocation schemes (see Section 2.4) is also in order. We will make sure that any such comparison implements the same *contention resolution algorithm*, which is the binary exponential backoff scheme in our case. This is because each contention resolution algorithm imposes its own limitation and properties on whatever contention minislot allocation scheme one can propose.

Having deferred the above assignments for future work, we concentrate our efforts on a more detailed study of the main component of our QoS scheduling architecture,

which is the *fair queueing* (FQ) algorithm that is responsible for guaranteeing the *Minimum Reserved Traffic Rate* for rtPS and nrtPS service flows. The remainder of this dissertation is dedicated to a better understanding of the properties and performance characteristics of a general class of such fair queueing algorithms. We specifically concentrate on the traffic engineering aspect of implementing the fair queueing algorithm within our proposed QoS architecture. The main concern of such traffic engineering problem is to determine, under normal operating conditions, the required resources (including the output link capacity and memory space requirements) to provide a certain *expected delay* for the supported end users. This question might also be slightly modified to consider appropriate selections of the *Minimum Reserved Traffic Rate* parameter that users should specify to maintain the *mean* (or *expected*) delay QoS requirements for the applications they are running.

In other words, the performance study of fair queueing algorithms that we perform in the following chapters will provide a better understanding of the relationship between the mean packet delay and the associated bandwidth reservations users can make under a particular fair queueing algorithm. This and more insights into the operation of fair queueing systems are the topic of the next three chapters.

Chapter 4

M/G/FQ Mean Packet Delay Analysis

THE remainder of this dissertation is dedicated to a more detailed investigation of the properties of a general class of fair scheduling, or fair queueing (FQ), policies. We use both mathematical analysis and simulation experiments to obtain useful results that describe the performance behavior and related characteristics of such queueing systems.

A significant volume of work in the literature [11 – 20] has been concerned with evaluating the *deterministic* worst-case delay guarantees that FQ algorithms can provide when the burstiness of the traffic feeding them is bounded (for example, shaped by a leaky bucket). Little work, though, has been reported on analyzing the delay characteristics of such policies under a general probabilistic traffic model, which is more relevant to the traffic engineering problem we introduced earlier in Section 3.5 (see also Section 4.1 for more details). This has been mainly due to the difficulty in statistically modeling the complex behavior of FQ algorithms.

Indeed an important advantage of statistical modeling of FQ systems as compared to worst-case deterministic analysis is that stochastic analysis takes into account the actual dynamics of the packet arrival process, thus being more accurate in predicting the system status under normal *steady-state* operating conditions.

In this dissertation, we carry out a stochastic performance evaluation of a general class of FQ algorithms. We first derive new upper and lower bounds on mean packet delay and mean buffer occupancy experienced by FQ systems when fed by Poisson arrivals, and then perform a multitude of simulation experiments to compare the performance of a number of FQ policies. This will allow us to derive some very interesting (and not completely intuitive) behavioral characteristics of fair scheduling algorithms.

We derive the analytical delay bounds in Chapter 5 and then describe the simulation experiments in Chapter 6. This chapter serves two purposes: First it introduces the underlying mathematical notation adopted throughout this dissertation, and second it allows us to conduct a quick analysis run-through to illustrate the level of difficulty one faces when trying to characterize the exact statistical behavior of FQ systems. This will provide a very good introduction when we derive the bounds on mean packet delay and mean buffer occupancy in the next chapter.

4.1 Motivation

Different end-to-end applications require different QoS guarantees from the network infrastructure. This is usually represented by different QoS parameters that such applications will ask the network to sustain.

For example, voice-over-IP and video conferencing applications have stringent requirements on *delay* and *delay jitter*. Delays above 500 ms impair human interaction and excessive delay jitter imposes extra limitations on the buffer space at both receiving ends. Other applications, on the other hand, might require stringent *bandwidth* and *error rate* QoS parameters instead of stringent delay and delay jitter. Business transactions over the Internet represent a good example of this behavior.

The Fair Queueing (FQ) component of our wireless QoS architecture (see Chapter 3) has only one QoS parameter (which is the *Minimum Reserved Traffic Rate*) that can be requested by rtPS and nrtPS service flows. It is necessary in such scenario to provide guidelines to translate between such *bandwidth* reservations one can make and the expected performance guarantees in terms of mean *delay* and *delay jitter* that the fair scheduling algorithm will provide. This is especially important in the case where the end user applications using rtPS and nrtPS service flows define their required QoS parameters in terms of delay and delay jitter.

As we mentioned at the beginning of this chapter, there have been various efforts to quantify the absolute maximum packet delay (worst-case scenario) that packets can incur for a specific bandwidth reservation under fair scheduling policies. Unfortunately, such analysis, by definition, neglects the true nature and main advantage of packet-switched networks in which substantial improvement in performance is achieved through statistical multiplexing of randomly arriving traffic. In other words, such analysis does not provide any clues about the actual *statistical* performance users are most likely to encounter under normal steady-state operating conditions.

Researchers have avoided stochastic analysis of fair scheduling algorithms (which results in mean packet delay for a certain bandwidth allocation) because of the involved complexity in trying to model fair scheduling policies under a probabilistic arrival model (see also Section 4.3). Therefore, to complement the research in this

area, we devote the remainder of this dissertation toward quantifying the relationship between the reservations users make under a general class of FQ systems and the performance they should expect in terms of mean packet delay and mean buffer occupancy.

We will not only address this performance aspect of fair scheduling systems, but we will also further investigate the similarities and differences that exist between different types of scheduling policies within our stochastic analysis framework.

4.2 M/G/FQ Notation and Assumptions

The problem we focus on in this dissertation consists of a single-server *Earliest Finish Time First* (EFTF) FQ scheduler served by an access link with a total capacity of C bits/second. The scheduler is work conserving, which means that the server must be busy if there are packets waiting in the system. We denote by $\mathbf{K} = \{1, 2, 3, ..., K\}$ the set of flows supported by such a scheduler, and by r_k, $k \in \mathbf{K}$, the minimum reserved service rate (in bits/second) associated with each flow k.

The FQ system is fed by multiple Poisson streams with arrival rates λ_1, λ_2, ...,λ_K as shown in Figure 4.1. The buffers corresponding to different flows are *infinite* in length and the packets in each one of those buffers are served in the order they arrive.

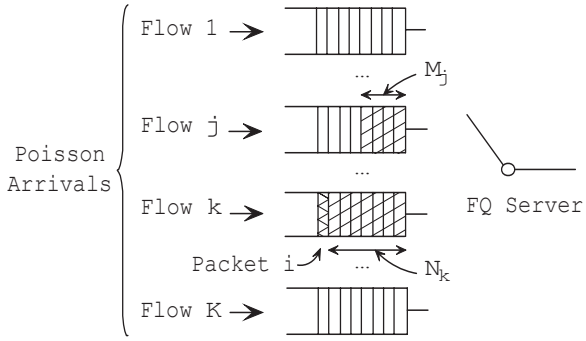

Figure 4.1: Fair Queueing system under study.

We use L_k^i to denote the length (in bits) of the ith data packet arrival at the kth buffer, $k \in \mathbf{K}$. Similarly, we use $X_k^i = L_k^i/C$ to denote the service time (in seconds) of the ith data packet arrival at the kth buffer. We assume that the random variables X_k^i from the multiple Poisson streams are identically distributed, mutually independent, and independent of the arrival times. Such variables X_k^i can assume any general distribution. We denote the mean service time of arriving data packets by $\overline{X} = E[X_k^i] = 1/\mu$, where μ is the mean service rate. The second moment of the service time is denoted by $\overline{X^2}$. For convenience, we will refer to the total arrival rate at the FQ system by $\lambda = \sum_{k=1}^{K} \lambda_k$.

The utilization of each flow k is denoted by $\rho_k = \lambda_k \overline{X} = \lambda_k/\mu$, while the utilization of the output link is given by $\rho = \lambda \overline{X}$. In this study we maintain $\rho < 1$, which keeps the system from being overloaded in an average sense.

We denote the *mean* packet waiting time in queue k by W_k, and the *expected* number of packets in such a queue (*not* including any packet that may be in service) by N_k. We assume *ergodicity* of the queueing system (which is true provided that $\rho < 1$) and note that, in our system, the values of N_k and W_k seen by an outside observer at a random time are the same as seen by an arriving customer. This is due to the Poisson character of the arrival process, which implies that the occupancy distribution upon arrival is typical [31].

Let us *tag* the ith data packet arrival at the kth queue of the FQ system. This tagged packet must wait in queue for a mean residual time R until the end of the current packet transmission and must also wait for the transmission of the mean number of packets N_k currently in the kth queue ahead of it. In addition, the tagged packet must also wait for the transmission of all packets in the system (not in queue k) with timestamps that are smaller than the timestamp assigned to the tagged packet. Some of these packets may even arrive after the tagged packet. The mean number of such packets in each queue j in the system is denoted by M_j (see Figure 4.1).

Thus, the mean waiting time (queuing delay) in queue k for the tagged packet is given by,

$$W_k = R + \frac{1}{\mu}\left(N_k + \sum_{j \in \mathbf{J}} M_j\right) \tag{4.1}$$

where we used $\mathbf{J}$ to represent the set of all flows supported by the scheduler *except* flow k, i.e., $\mathbf{J} = \{j \in \mathbf{K} : j \neq k\}$. We can evaluate the mean residual time R by a graphical argument as in [31] to obtain,

$$R = \frac{1}{2}\sum_{k=1}^{K} \lambda_k \overline{X_k^2} = \frac{\lambda \overline{X^2}}{2} \tag{4.2}$$

Applying Little's theorem, $N_k = \lambda_k W_k$, to (4.1) and substituting the value of R in (4.2), we obtain,

$$W_k = \frac{1}{2}\overline{X^2}\lambda + \rho_k W_k + \frac{1}{\mu}\sum_{j \in \mathbf{J}} M_j \tag{4.3}$$

4.3 Exact Mean Packet Delay Analysis

The only unknown quantities in (4.3) now are the values of M_j. Let us attempt to find such values. We first notice that when the tagged packet i arrives at queue k, it will find n_k packets in front of it with probability $P[q_k = n_k]$. We already know that $\sum_{n_k=0}^{\infty} n_k \cdot P[q_k = n_k] = N_k$, which is the mean number of packets in queue k. In a similar way, we can evaluate M_j for $j \in \mathbf{J}$ as follows,

$$M_j = \sum_{n_k=0}^{\infty} E[M_j \,|q_k = n_k] \cdot P[q_k = n_k], \quad j \in \mathbf{J} \tag{4.4}$$

where $E[M_j \,|q_k = n_k]$ is the expected value of the random variable[1] M_j given that the arriving tagged packet finds n_k packets in queue k. Hence, to evaluate the mean value of M_j we need to evaluate $E[M_j \,|q_k = n_k]$ for each value of n_k in addition to finding the probability distribution $P[q_k = n_k]$ of the buffer k occupancy.

If we *assume* that the *connection potentials* for the different incoming flows are equal on *average*[2], then for each n_k packets from queue k, we can serve up to $\left\lfloor (n_k+1)\,\frac{r_j}{r_k} \right\rfloor$ packets from queue j (*if* such number of packets exists). This is due to the way timestamps are assigned in an EFTF FQ system. In other words,

$$\begin{aligned} E[M_j \,|q_k = n_k] = & \sum_{m_j=0}^{\left\lfloor (n_k+1)\frac{r_j}{r_k} \right\rfloor - 1} m_j \cdot P[(q_j + a_j) = m_j] \\ + & \sum_{m_j=\left\lfloor (n_k+1)\frac{r_j}{r_k} \right\rfloor}^{\infty} \left\lfloor (n_k+1)\,\frac{r_j}{r_k} \right\rfloor \cdot P[(q_j + a_j) = m_j] \end{aligned} \tag{4.5}$$

where $P[(q_j + a_j) = m_j]$ is the probability that the sum of both (a) the packets already queued in buffer j when the tagged packet i arrives; plus (b) any new packet arrivals at buffer j while the tagged packet i is waiting in buffer is equal to m_j. This is a joint distribution of both random variables q_j and a_j.

Notice that we removed the conditioning on $q_k = n_k$ in the probability $P[(q_j + a_j) = m_j]$ for convenience. A more comprehensive notation would have been $P[(q_j + a_j) = m_j \,|q_k = n_k]$. To simplify the equations in the remainder of this section, we will remove any conditioning on $q_k = n_k$ and implicitly assume its existence. In addition, we will simplify the notation further by defining a new quantity m'_j as follows,

[1] Notice that we use the symbol M_j to denote both (a) the number of packets served from queue j before the tagged packet is served (which is a *random variable*) and also (b) the *mean* of such random variable. This helps us avoid a rather awkward notation. We make sure, however, that the context is clear about the intended meaning of M_j.

[2] We use this assumption to simplify the mathematical analysis in this section. We will incorporate the differences in connection potentials in the final analysis in the next chapter.

$$m'_j = \left\lfloor (n_k + 1)\, \frac{r_j}{r_k} \right\rfloor \tag{4.6}$$

Using (4.6), we can write (4.5) as follows,

$$\begin{aligned} E\left[M_j \,|q_k = n_k\right] &= \sum_{m_j=0}^{m'_j-1} m_j \cdot P[(q_j + a_j) = m_j] \\ &+ m'_j \cdot \sum_{m_j=m'_j}^{\infty} P[(q_j + a_j) = m_j] \end{aligned} \tag{4.7}$$

By the rules of conditional probability, we can write,

$$P[(q_j + a_j) = m_j] = \sum_{l_j=0}^{m_j} P\left[a_j = m_j - l_j \,|q_j = l_j\right] \cdot P[q_j = l_j] \tag{4.8}$$

where the probability $P[q_j = l_j]$ corresponds to the buffer occupancy distribution of queue j.

The main purpose of proceeding with this line of analysis is to show how cumbersome the equations are going to be for the mean packet delay of a FQ system when attempting to follow through with the above *exact* equations. To clarify this further, let us consider the very simple case of a FQ system supporting two flows only, i.e., $\mathbf{K} = \{k, j\} = \{1, 2\}$, and let us assume that all packets have the same fixed packet length[3].

Finding the probability $P\left[a_j = m_j - l_j \,|q_j = l_j\right]$ is possible in this case. Remembering that the arrivals are Poisson in nature and are independent of service times, we can evaluate $P\left[a_j = m_j - l_j \,|q_j = l_j\right]$ from knowledge of the waiting time of the tagged packet that just arrived. This waiting time is the sum of three components: (a) the residual time of the packet currently in service, (b) the time to serve n_k packets ahead of the tagged packet in queue k, and finally (c) the time to serve a maximum of m_j or m'_j packets from queue j. Such waiting time for the tagged packet is thus given by,

$$X' = \begin{cases} R + \frac{1}{\mu}\left(n_k + m_j\right), & m_j < m'_j \\ R + \frac{1}{\mu}\left(n_k + m'_j\right), & m_j \geq m'_j \end{cases} \tag{4.9}$$

where we used the not so rigorous approximation that the actual residual time for the packet in service is equal to the mean residual time R always. Proceeding with the analysis, this means that,

[3]This assumption applies only for this section. In the analysis of Chapter 5, we go back to the assumption of generally distributed packet lengths.

$$P[a_j = m_j - l_j \,|q_j = l_j\,] = \frac{(\lambda_j X')^{m_j - l_j}}{(m_j - l_j)\,!} e^{-\lambda_j X'} \tag{4.10}$$

Substituting (4.10) into (4.8) gives,

$$P[(q_j + a_j) = m_j] = \sum_{l_j=0}^{m_j} \frac{(\lambda_j X')^{m_j - l_j}}{(m_j - l_j)\,!} e^{-\lambda_j X'} \cdot P[q_j = l_j] \tag{4.11}$$

and substituting (4.11) into (4.7) and expanding X' we get,

$$\begin{aligned} &E\,[M_j \,|q_k = n_k\,] = \\ &\sum_{m_j=0}^{m'_j-1} \left(m_j \cdot \sum_{l_j=0}^{m_j} \frac{(\lambda_j R + \rho_j (n_k + m_j))^{m_j - l_j}}{(m_j - l_j)\,!} \cdot e^{-\lambda_j R - \rho_j (n_k + m_j)} \cdot P[q_j = l_j] \right) \\ &+ \sum_{m_j=m'_j}^{\infty} \left(m'_j \cdot \sum_{l_j=0}^{m_j} \frac{\left(\lambda_j R + \rho_j \left(n_k + m'_j\right)\right)^{m_j - l_j}}{(m_j - l_j)\,!} \cdot e^{-\lambda_j R - \rho_j \left(n_k + m'_j\right)} \cdot P[q_j = l_j] \right) \end{aligned} \tag{4.12}$$

Let us now concentrate on the second term on the right hand side of (4.12). Notice that if we change the order of the sums in that term, we end up with,

$$m'_j \cdot \sum_{l_j=0}^{\infty} \left(\sum_{m_j=\max\left(l_j, m'_j\right)}^{\infty} \frac{\left(\lambda_j R + \rho_j \left(n_k + m'_j\right)\right)^{m_j - l_j}}{(m_j - l_j)\,!} \cdot e^{-\lambda_j R - \rho_j \left(n_k + m'_j\right)} \right) \cdot P[q_j = l_j] \tag{4.13}$$

If we introduce the intermediate variable $i = m_j - l_j$, the above expression becomes,

$$\begin{aligned} &m'_j \cdot \sum_{l_j=0}^{m'_j-1} \left(\sum_{m_j=m'_j}^{\infty} \frac{\left(\lambda_j R + \rho_j \left(n_k + m'_j\right)\right)^{m_j - l_j}}{(m_j - l_j)\,!} \cdot e^{-\lambda_j R - \rho_j \left(n_k + m'_j\right)} \right) \cdot P[q_j = l_j] \\ &+ m'_j \cdot \sum_{l_j=m'_j}^{\infty} \left(\sum_{i=0}^{\infty} \frac{\left(\lambda_j R + \rho_j \left(n_k + m'_j\right)\right)^{i}}{i\,!} \cdot e^{-\lambda_j R - \rho_j \left(n_k + m'_j\right)} \right) \cdot P[q_j = l_j] \end{aligned} \tag{4.14}$$

But the sum over i is the sum of a discrete Poisson probability distribution, which will add up to unity. This transforms (4.14) into,

$$\begin{aligned} &m'_j \cdot \sum_{l_j=0}^{m'_j-1} \left(\sum_{m_j=m'_j}^{\infty} \frac{\left(\lambda_j R + \rho_j \left(n_k + m'_j\right)\right)^{m_j - l_j}}{(m_j - l_j)\,!} \cdot e^{-\lambda_j R - \rho_j \left(n_k + m'_j\right)} \right) \cdot P[q_j = l_j] \\ &+ m'_j \cdot \sum_{l_j=m'_j}^{\infty} P[q_j = l_j] \end{aligned} \tag{4.15}$$

Now, back to the first term on the right hand side of (4.12), switching the order of the sums, we get,

$$\sum_{l_j=0}^{m'_j-1}\left(\sum_{m_j=l_j}^{m'_j-1} m_j\cdot\frac{\left(\lambda_j R+\rho_j\left(n_k+m_j\right)\right)^{m_j-l_j}}{\left(m_j-l_j\right)!}\cdot e^{-\lambda_j R-\rho_j(n_k+m_j)}\right)\cdot P\left[q_j=l_j\right] \quad (4.16)$$

Substituting (4.15) and (4.16) into (4.12) gives,

$$\begin{aligned}E\left[M_j\,|q_k=n_k\right] &= m'_j\cdot\sum_{l_j=m'_j}^{\infty}P\left[q_j=l_j\right]\\ &+\sum_{l_j=0}^{m'_j-1}\left(\begin{array}{l}\sum\limits_{m_j=m'_j}^{\infty} m'_j\cdot\frac{\left(\lambda_j R+\rho_j\left(n_k+m'_j\right)\right)^{m_j-l_j}}{\left(m_j-l_j\right)!}\cdot e^{-\lambda_j R-\rho_j\left(n_k+m'_j\right)}\\ +\sum\limits_{m_j=l_j}^{m'_j-1} m_j\cdot\frac{\left(\lambda_j R+\rho_j(n_k+m_j)\right)^{m_j-l_j}}{\left(m_j-l_j\right)!}\cdot e^{-\lambda_j R-\rho_j(n_k+m_j)}\end{array}\right)\cdot P\left[q_j=l_j\right]\end{aligned} \quad (4.17)$$

The three terms in (4.17) clearly make sense as the basic components for the mean value of M_j given $q_k=n_k$. The first component corresponds to the case of serving a maximum of m'_j packets *if* they already exist in the jth buffer at the time of the tagged packet arrival. If a smaller number of packets exist in the buffer, then we serve anywhere between zero packets up to a maximum of m'_j packets from queue j depending on the number of new arrivals at the jth buffer while the tagged packet is waiting in queue to be served. This represents the other two components in (4.17). To proceed, (4.17) can be written as follows,

$$\begin{aligned}E\left[M_j\,|q_k=n_k\right] &= m'_j\cdot\left(1-\sum_{l_j=0}^{m'_j-1}P\left[q_j=l_j\right]\right)\\ &+m'_j\cdot\sum_{l_j=0}^{m'_j-1}\left(1-\sum_{m_j=l_j}^{m'_j-1}\frac{\left(\lambda_j R+\rho_j\left(n_k+m'_j\right)\right)^{m_j-l_j}}{\left(m_j-l_j\right)!}\cdot e^{-\lambda_j R-\rho_j\left(n_k+m'_j\right)}\right)\cdot P\left[q_j=l_j\right]\\ &+\sum_{l_j=0}^{m'_j-1}\left(\sum_{m_j=l_j}^{m'_j-1} m_j\cdot\frac{\left(\lambda_j R+\rho_j(n_k+m_j)\right)^{m_j-l_j}}{\left(m_j-l_j\right)!}\cdot e^{-\lambda_j R-\rho_j(n_k+m_j)}\right)\cdot P\left[q_j=l_j\right]\end{aligned} \quad (4.18)$$

After some algebra,

$$\begin{aligned}E\left[M_j\,|q_k=n_k\right] &= m'_j\\ &-m'_j\cdot\sum_{l_j=0}^{m'_j-1}\left(\sum_{m_j=l_j}^{m'_j-1}\frac{\left(\lambda_j R+\rho_j\left(n_k+m'_j\right)\right)^{m_j-l_j}}{\left(m_j-l_j\right)!}\cdot e^{-\lambda_j R-\rho_j\left(n_k+m'_j\right)}\right)\cdot P\left[q_j=l_j\right]\\ &+\sum_{l_j=0}^{m'_j-1}\left(\sum_{m_j=l_j}^{m'_j-1} m_j\cdot\frac{\left(\lambda_j R+\rho_j(n_k+m_j)\right)^{m_j-l_j}}{\left(m_j-l_j\right)!}\cdot e^{-\lambda_j R-\rho_j(n_k+m_j)}\right)\cdot P\left[q_j=l_j\right]\end{aligned} \quad (4.19)$$

A shorter notation for (4.19) is the following,

$$
\begin{aligned}
&E\left[M_j \,|q_k = n_k\right] = \\
&\sum_{l_j=0}^{m'_j-1} \left(\sum_{m_j=l_j}^{m'_j-1} m_j \cdot P_{m_j-l_j}\left(\lambda_j R + \rho_j\left(n_k + m_j\right)\right) \right) \cdot P\left[q_j = l_j\right] \\
&+m'_j \cdot \left(1 - \sum_{l_j=0}^{m'_j-1} \left(\sum_{m_j=l_j}^{m'_j-1} P_{m_j-l_j}\left(\lambda_j R + \rho_j\left(n_k + m'_j\right)\right) \right) \cdot P\left[q_j = l_j\right] \right)
\end{aligned}
\tag{4.20}
$$

where $P_n(\lambda t) = e^{-\lambda t}(\lambda t)^n / n!$ is the well-documented Poisson distribution. The expression in (4.20) has a finite computation time since all the sums are finite. A nice feature of this expression is that the Poisson probabilities can be evaluated once and reused if necessary.

The final step in the computation process is to substitute (4.20) for each value of n_k in the following equation,

$$
M_j = \sum_{n_k=0}^{\infty} E\left[M_j \,|q_k = n_k\right] \cdot P\left[q_k = n_k\right] \tag{4.21}
$$

It is obvious from (4.3), (4.20) and (4.21) that the exact equation for the mean packet delay is a cumbersome one, and even if we can find it as a closed expression (given that we can evaluate the buffer occupancy probability distribution of both buffers k and j in a recursive way), such an expression would be hardly manageable. This leads us to the conclusion that an approximate solution or some sort of bounds on the mean packet delay experienced by a FQ algorithm might be a very good replacement for the exact expression.

Chapter 5

Mean Delay Bounds for Fair Queueing Systems

TO the best of our knowledge, the only work on statistical modeling of fair queuing (FQ) algorithms is that of [21], in which the author derives stochastic bounds on the delay distribution of GPS-related FQ algorithms fed by a Switched Bernoulli Batch process. The analysis in [21] is quite complex and does not result in explicit analytical equations, thus limiting its usefulness for back-of-the-envelope calculations and comparisons. The analysis also makes some limiting assumptions such as the use of fixed packet lengths and the need to set all flows other than the tagged one to be greedy all the time.

The analysis we introduce here, on the other hand, is much simpler than that in [21] and results in upper and lower bounds on mean waiting time and mean buffer occupancy experienced by a FQ algorithm fed by Poisson arrivals. Our analysis follows closely the well-known M/G/1 queueing analysis, thus the name M/G/FQ, and results in well-contained equations that provide significant theoretical value and great insight into the operation of FQ systems.

Our analysis fits a broad range of scheduling policies that exhibit fairness bounds. Such a class of FQ algorithms is similar to the one studied in [17] and many scheduling policies belong to this group including WFQ, SCFQ, SFQ and SPFQ. The key to our analysis is to utilize the bounded fairness criterion of FQ systems (see Chapter 2) to derive the desired bounds on mean waiting time and mean buffer occupancy.

5.1 M/G/FQ Stochastic Analysis

In Chapter 4 we found that the mean waiting time (queuing delay) for a tagged flow k in an EFTF FQ system is given by (4.3), which we repeat here for convenience,

$$W_k = \frac{1}{2}\overline{X^2}\lambda + \rho_k W_k + \frac{1}{\mu}\sum_{j\in\mathbf{J}} M_j \tag{5.1}$$

where $\mathbf{J} = \{j \in \mathbf{K} : j \neq k\}$ is the set of all flows $\mathbf{K}$ supported by the scheduler *except* flow k.

We showed in Chapter 4 how difficult it is to find the exact values of the M_j quantities, which are certainly dependent on both the arrival rates at the different queues and the corresponding reservation rates. We can, however, find upper and lower bounds on the mean waiting time W_k if we can find upper and lower bounds on the mean values M_j.

Let us consider a single queue $j \in \mathbf{J}$ in the system. Assume that this queue has a connection potential $v_j(a_k^i)$ at the time a_k^i of the tagged packet i arrival at queue k. Assume also that the connection potential of queue k was $v_k(a_k^i)$ at that same instant. Using the result for the connection potential in (2.5) and noting that the M_jth packet in the jth queue should have a timestamp that is smaller than the ith packet (in queue k) so that the FQ scheduler can serve it first, we get,

$$E\left[v_j(a_k^i)\right] + M_j\frac{\overline{L_j}}{r_j} \leq E\left[v_k(a_k^i)\right] + (N_k+1)\frac{\overline{L_k}}{r_k} \tag{5.2}$$

Also, packet M_j+1 in the jth queue should have a timestamp that is larger than the ith packet in queue k because packet i gets served first. Hence, we can write,

$$E\left[v_j(a_k^i)\right] + (M_j+1)\frac{\overline{L_j}}{r_j} \geq E\left[v_k(a_k^i)\right] + (N_k+1)\frac{\overline{L_k}}{r_k} \tag{5.3}$$

Rearranging (5.2), we get the following upper bound on the mean value M_j,

$$M_j \leq (N_k+1)\frac{r_j}{r_k} + \overline{\delta_{kj}}\frac{r_j}{\overline{X}C}, \quad j \in \mathbf{J} \tag{5.4}$$

where $\delta_{kj} = v_k - v_j$ is the difference in the connection potentials of flows k and j, respectively, as defined in (2.9). Notice that we have used $\overline{L_k} = \overline{L_j}$ in deriving (5.4) which is based on our earlier assumption of identically distributed packet lengths from all the supported flows (see Section 4.2).

Similarly, rearranging (5.3) results in a lower bound on the mean value M_j,

$$M_j \geq (N_k + 1)\,\frac{r_j}{r_k} + \overline{\delta_{kj}}\frac{r_j}{\overline{X}C} - 1, \quad j \in \mathbf{J} \tag{5.5}$$

Notice that the above upper and lower bounds on M_j are dependent only on the value of N_k, which is a desirable feature if we want to reduce (5.1) into a manageable equation (remember that N_k and W_k are related using Little's law, $N_k = \lambda_k W_k$). However, achieving this simplification means that we have lost any information about the effect of mean arrival rate of flow j on the actual M_j value. This might affect how tight the upper and lower bounds on M_j are going to be. Tighter bounds are expected when the arrival rate at the j flow is high enough that we always have packets to send from such buffer.

However, we can still include *some* information about the mean arrival rate at the jth flow as follows: Notice that we can serve *no less* than zero packets and *no more* than $N_j + \lambda_j W_k$ packets from queue j before we have to serve the tagged packet i in queue k, where N_j is the mean number of packets in queue j as observed by the arriving tagged packet and $\lambda_j W_k$ is the mean number of packets that arrive at buffer j while the tagged packet is waiting to be served.

This means that improved upper and lower bounds on M_j can be derived from (5.4) and (5.5), respectively, as follows,

$$M_j \leq \min\left((N_k + 1)\,\frac{r_j}{r_k} + \overline{\delta_{kj}}\frac{r_j}{\overline{X}C},\ N_j + \lambda_j W_k\right), \quad j \in \mathbf{J} \tag{5.6}$$

$$M_j \geq \min\left(\max\left((N_k + 1)\,\frac{r_j}{r_k} + \overline{\delta_{kj}}\frac{r_j}{\overline{X}C} - 1,\ 0\right),\ N_j + \lambda_j W_k\right), \quad j \in \mathbf{J} \tag{5.7}$$

It is interesting to note that if $M_j = N_j + \lambda_j W_k$, the situation becomes very similar to that of a *strict non-preemptive priority* queueing where queue k has a lower priority than queue j.

5.2 The Upper Bound on Mean Waiting Time

The challenge we face in trying to solve for the upper bound on mean waiting time is that we need to choose the minimum of two quantities in (5.6) for each flow $j \in \mathbf{J}$ before being able to substitute it into (5.1). Such a decision cannot be made without prior knowledge of the actual N_j (or W_j) values, $j \in \mathbf{J}$, which are the unknowns we are seeking to find.

To avoid such a problem we notice that as far as the upper bound on mean waiting time is concerned, using the first expression on the right hand side of (5.6) instead of

the minimum does not actually affect the correctness of the upper bound on M_j, although it might slightly weaken its tightness. Since such an expression is not dependent on the value of N_j, we can drastically simplify the derivation process. Substituting this expression into (5.1) and using Little's theorem, we get,

$$W_k \leq \frac{\frac{1}{2}\overline{X^2}\lambda + \sum\limits_{j\in\mathbf{J}} \left[\frac{1}{\mu}\frac{r_j}{r_k} + \overline{\delta_{kj}}\frac{r_j}{C}\right]}{1 - \rho_k \frac{\sum\limits_{j\in\mathbf{K}} r_j}{r_k}} \tag{5.8}$$

We have already derived an upper bound on the quantity $\overline{\delta_{kj}}$ for *Fair Queueing* (FQ) systems in Chapter 2 (cf. (2.10) and (2.14)). Substituting (2.10) into (5.8) we get,

$$W_k \leq \frac{\frac{1}{2}\overline{X^2}\lambda + \sum\limits_{j\in\mathbf{J}} \left[\frac{1}{\mu}\frac{r_j}{r_k} + \overline{\psi_j}\frac{r_j}{C}\right]}{1 - \rho_k \frac{\sum\limits_{j\in\mathbf{K}} r_j}{r_k}} \tag{5.9}$$

On the other hand, using the improved upper bound on $\overline{\delta_{kj}}$ from (2.14) transforms (5.8) into,

$$W_k \leq \frac{\frac{1}{2}\overline{X^2}\lambda + \sum\limits_{j\in\mathbf{J}} \left[\frac{1}{\mu}\frac{r_j}{r_k} + \left(\rho'_j\overline{\psi_j} - \rho'_k\overline{\psi_k}\right)\frac{r_j}{C}\right]}{1 - \rho_k \frac{\sum\limits_{j\in\mathbf{K}} r_j}{r_k}} \tag{5.10}$$

In Section 5.5, we follow a more elaborate approach to derive an upper bound on mean waiting time using all the information about M_j provided by (5.6).

5.3 The Lower Bound on Mean Waiting Time

Finding the lower bound on the mean waiting time W_k requires a similar approach to that of finding the upper bound. For the lower bound, however, we cannot just substitute the first expression in (5.7) instead of the required minimum since this is mathematically incorrect. However, we can still derive a simple equation for the lower bound similar to that of (5.10) by setting the minimum in (5.7) to zero all the time, which gives the following simple lower bound on mean waiting time,

$$W_k \geq \frac{\frac{1}{2}\overline{X^2}\lambda}{1 - \rho_k} \tag{5.11}$$

Obviously, this is not the best possible lower bound on mean waiting time. However, as will be apparent in Section 5.6, this lower bound is reasonably tight in almost all practical cases one might encounter. Section 5.5 also explains a more elaborate scheme for finding the lower bound on mean waiting time using the exact formula in (5.7).

5.4 Properties of the Delay Bounds

An interesting observation we can make about the M/G/FQ delay bounds in (5.9), (5.10) and (5.11) is that both the upper and lower bounds increase in inverse proportion to $1 - \rho_k$ (or $1 - \rho_k\,(C/r_k)$). This means that the mean waiting time in our system is expected to dramatically increase as the utilization factor increases, or at least that would be the behavior of the delay bounds in such a condition. Comparing this to an M/G/1 queueing system, we notice the same exact behavior for the mean waiting time versus utilization. This behavior is actually a general characteristic of almost any queueing system one might encounter.

Now let us consider the conditions that would result in tighter bounds on mean waiting time. We can see from (5.10) and (5.11) that the difference between the upper and lower bounds is mainly dependent on a summation factor including the fairness bound $\overline{\psi_j}$ for $j \in \mathbf{J}$. This means that tighter bounds are expected in the following situations: (1) when the number of flows K (and hence J) supported by the scheduler is smaller, (2) when the fairness bound of the FQ system $\overline{\psi_j}$ is tighter, and finally (3) because of the dependence on the inverse of $(1 - \rho_k)$, tighter delay bounds are expected when the load on queue k, measured by ρ_k, is smaller.

It is also interesting to notice that the upper and lower bounds in (5.10) and (5.11) are applicable not only to one specific FQ algorithm, but rather to the whole class of FQ policies that exhibit a specific fairness bound. This illustrates the flexibility and generality of our analysis method, which requires only partial information (the fairness bound) about the FQ algorithm itself to be able to produce bounding criterion for its mean waiting time. Such ability is quite important in many situations where the complexity of the FQ algorithm under consideration may prohibit mathematical tractability of its exact properties.

On the other side of the coin, deriving delay bounds based on partial information (the fairness bound) of FQ algorithms means that the obtained bounds need to be as wide apart as possible to accommodate all scheduling policies with the same fairness bound irrespective of their internal structure. The power of our analysis method is that we can further tailor its delay bounds to a specific scheduling algorithm by deriving tighter bounds on the service lag distributions of such algorithms. Of course, this requires more information about the specific scheduling policy under consideration to be known ahead of time to carry out such an analysis.

As a final note, we want to elaborate on our earlier assumption (approximation) in Chapter 2 (Section 2.5.2) that allowed us to derive a tighter upper bound on $\overline{\delta_k}$ and $\overline{\delta_{kj}}$, which we then used to derive a tighter upper bound on mean waiting time in (5.10). We assumed that the tagged flow k is guaranteed an equivalent server of minimum capacity of r_k. This allowed us to derive an upper bound on $\Pr[k \in B]$ (the probability of the tagged flow being backlogged), which then allowed us to derive a tighter bound on $\overline{\delta_k}$ and $\overline{\delta_{kj}}$.

As we mentioned earlier, this assumption is not mathematically rigorous (see also the results of Chapter 6 to verify that). However, the assumption still provides valid upper bounds as we will show in the simulation experiments in Section 5.6 below. The reason this assumption works is that when the load on the system is low, which is the only time when the assumption is *not* totally accurate, the *actual* mean value of M_j is so small (virtually close to zero). This is due to the fact that the system is empty for a reasonable amount of time making N_k and the time stamp of the tagged packet (and hence M_j) very small. A slight error in calculating a positive upper bound on the M_j value would not be significant at all in such a scenario (again see the results of Section 5.6 for further validation of this argument).

We conclude this section by reminding the reader that the upper and lower bounds on mean waiting time can also be used to derive a corresponding upper and lower bounds on mean buffer occupancy using Little's law, which states that $N_k = \lambda_k W_k$. Such bounds on buffer occupancy can be very useful in allocating buffer space in routers and other switching devices that support QoS.

5.5 Improved Delay Bounds

Including the information in (5.6) and (5.7) about the arrival rate at the non-tagged flows in deriving the upper and lower bounds on mean waiting time is still possible. This, however, requires a more involved analysis. In this section, we explain two possible approaches to include the extra information: one that involves an additional mathematical equation and the other involves an iterative algorithm rather than a well-contained mathematical formula.

Let us start with the simple approach. Notice that another mathematically *valid* upper bound on the M_j values can be calculated using only the second expression on the right hand side of (5.6) instead of the minimum. Substituting this expression into (5.1), we get,

$$W_k \leq \frac{1}{2}\overline{X^2}\lambda + \rho_k W_k + \frac{1}{\mu}\sum_{j \in \mathbf{J}} (N_j + \lambda_j W_k) \tag{5.12}$$

By Little's Law, this becomes,

$$W_k \leq \frac{1}{2}\overline{X^2}\lambda + \rho_k W_k + \sum_{j\in\mathbf{J}} \rho_j W_j + \sum_{j\in\mathbf{J}} \rho_j W_k \tag{5.13}$$

Rearranging, we get,

$$W_k \leq \frac{\frac{1}{2}\overline{X^2}\lambda + \sum\limits_{j\in\mathbf{J}} \rho_j W_j}{1-\rho} \tag{5.14}$$

where $\rho = \sum_{k=1}^{K} \rho_k$ is the utilization of the output link. To solve for the upper bound on mean waiting time W_k, we need to calculate the sum involving the W_j values. To do that, first notice that in this scenario, where $M_j = N_j + \lambda_j W_k,\ j \in \mathbf{J}$, we have a very similar situation to a *strict non-preemptive priority* queueing system[1] where queue k has the lowest priority among *all* the other flows $j \in \mathbf{J}$ in the system.

It is apparent why this represents a valid upper bound on the mean waiting time of flow k. Packets corresponding to flow k incur the worst possible waiting time when they have the lowest priority. We expect this upper bound on mean waiting time to be tight (i.e., close to the actual mean waiting time of flow k) if the mean arrival rate at the non-tagged flows is small. This is because in such case, packets in the lowest priority buffer do not have to wait for many non-tagged flow packets to depart before they get a chance to depart, too. However, this same upper bound will tend to be loose when the arrival rate at the non-tagged flows is high (i.e., when the non-tagged flows are greedy). This should not be alarming, however, because the upper bound on mean waiting time that we derived in (5.10) will be tight in such greedy sessions scenario.

To calculate the new upper bound on mean waiting time, we notice that if flow k has the lowest priority compared to all the other flows, then the arrivals at queue k do not affect the mean waiting time W_j of any other flow $j \in \mathbf{J}$. Since the server is work conserving, this means that for the purposes of computing the mean waiting times $W_j,\ j \in \mathbf{J}$, we can neglect the existence of flow k (except for calculating the residual time R). This is very similar to the line of analysis in priority M/G/1 queueing systems. In other words, for calculating the W_j values, we consider a new M/G/1 queueing system with only $\mathbf{J}$ flows (instead of $\mathbf{K}$ flows) and a link capacity of C bits/s. In such system, the aggregate Poisson arrival rate is given by $\sum_{j\in\mathbf{J}} \lambda_j = \lambda - \lambda_k$, which means that the server utilization[2] is now $\rho - \rho_k$.

[1]Notice that we cannot just use the simple M/G/1 non-preemptive priority queueing result here because there is no priority order among the *other* flows $j \in \mathbf{J}$ in the system.

[2]This applies only for the purposes of computing the W_j values, $j \in \mathbf{J}$.

Remember that the conservation law of M/G/1 systems [31] states that the sum of mean waiting times of individual flows weighted by their corresponding utilization is equal to the aggregate mean waiting time weighted by the output link utilization. For our case, this means that,

$$\sum_{j \in \mathbf{J}} \rho_j W_j = (\rho - \rho_k) \cdot \frac{R}{1 - (\rho - \rho_k)} = \frac{\frac{1}{2}\overline{X^2}\lambda\,(\rho - \rho_k)}{1 - (\rho - \rho_k)} \tag{5.15}$$

where $R = \lambda\overline{X^2}\Big/2$ is the mean residual time for the whole system. Substituting (5.15) into (5.14), we get,

$$W_k \leq \frac{\frac{1}{2}\overline{X^2}\lambda\left[1 + \frac{(\rho - \rho_k)}{1 - (\rho - \rho_k)}\right]}{1 - \rho} \tag{5.16}$$

Rearranging, the new upper bound on mean waiting time W_k becomes,

$$W_k \leq \frac{\frac{1}{2}\overline{X^2}\lambda}{(1 - \rho)\,(1 - \rho + \rho_k)} \tag{5.17}$$

Hence, an improved upper bound on mean waiting time W_k can be found by combining (5.10) and (5.17) together, which results in,

$$W_k \leq \min\left(\frac{\frac{1}{2}\overline{X^2}\lambda + \sum_{j \in \mathbf{J}}\left[\frac{1}{\mu}\frac{r_j}{r_k} + \left(\rho_j'\overline{\psi_j} - \rho_k'\overline{\psi_k}\right)\frac{r_j}{C}\right]}{1 - \rho_k\frac{\sum_{j \in \mathbf{K}} r_j}{r_k}},\ \frac{\frac{1}{2}\overline{X^2}\lambda}{(1 - \rho)\,(1 - \rho + \rho_k)}\right) \tag{5.18}$$

As we explained earlier, the first expression on the right hand side of (5.18) is expected to be more useful when the non-tagged flows are greedy (i.e., exhibiting a high mean arrival rate), while the second expression on the right hand side of (5.18) is expected to be more useful when the non-tagged flows are *not* greedy. We illustrate this idea further in the test results of Section 5.6.

A different approach to utilizing the extra information in (5.6) and (5.7) in deriving the upper and lower bounds on mean waiting time is through an iterative algorithm, which we explain below. Since the result for the M_j values in (5.6) and (5.7) are dependent on the actual value for the mean waiting time W_j (and W_k), which are the unknowns we are trying to find by computing M_j's, the iterative algorithm starts with an initial guess for the M_j values and iterates as we adjust the M_j values based on the calculated upper or lower bounds on mean waiting time W_k (and W_j).

Let us start by explaining the iterative algorithm for the upper bound on mean waiting time. For each flow supported by the scheduler $k \in \mathbf{K}$, we build an array M with $K-1$ elements. Each element of the array M represents the corresponding value of M_j, $j \in \mathbf{J}$. We fill the M array by the initial guess for the upper bound on the M_j values computed from the first expression on the right hand side of (5.6). We denote such initial guess by M_j^0, which is given by,

$$M_j^0 = (N_k + 1)\,\frac{r_j}{r_k} + \left(\rho_j'\overline{\psi_j} - \rho_k'\overline{\psi_k}\right)\frac{r_j}{\overline{X}C}, \quad j \in \mathbf{J} \tag{5.19}$$

which means that the initial guess for the upper bound on mean waiting time W_k is given by,

$$W_k^0 = \frac{1}{2}\,\overline{X^2}\lambda + \rho_k W_k + \frac{1}{\mu}\sum_{j\in\mathbf{J}} M_j^0 = \frac{\frac{1}{2}\overline{X^2}\lambda + \sum\limits_{j\in\mathbf{J}}\left[\frac{1}{\mu}\frac{r_j}{r_k} + \left(\rho_j'\overline{\psi_j} - \rho_k'\overline{\psi_k}\right)\frac{r_j}{C}\right]}{1 - \rho_k\frac{\sum\limits_{j\in\mathbf{K}} r_j}{r_k}} \tag{5.20}$$

After calculating the upper bound on waiting time W_k^0 for all the flows $k \in \mathbf{K}$, we compute the next iterative value for the upper bounds on M_j, denoted by M_j^1. This is done using the following equation,

$$\begin{aligned} M_j^1 &= \min\left(M_j^0, N_j^0 + \lambda_j W_k^0\right) \\ &= \min\left(\left(N_k^0 + 1\right)\frac{r_j}{r_k} + \left(\rho_j'\overline{\psi_j} - \rho_k'\overline{\psi_k}\right)\frac{r_j}{\overline{X}C},\ \lambda_j\left(W_j^0 + W_k^0\right)\right), \qquad j \in \mathbf{J} \end{aligned} \tag{5.21}$$

where N_k^0 is calculated from W_k^0 by Little's Law. This is done for all the elements of the array M corresponding to each flow $k \in \mathbf{K}$. Once the next iterative value of the upper bound on M_j is found, the next guess for the upper bound on mean waiting time is evaluated as follows,

$$W_k^1 = \frac{1}{2}\,\overline{X^2}\lambda + \rho_k W_k + \frac{1}{\mu}\sum_{j\in\mathbf{J}} M_j^1 = \frac{\frac{1}{2}\,\overline{X^2}\lambda + \frac{1}{\mu}\sum\limits_{j\in\mathbf{J}} M_j^1}{1 - \rho_k} \tag{5.22}$$

This process is repeated until the guesses on the M_j values are no longer modified, at which time we read the final value on the upper bound on mean packet waiting times.

Notice that the iterative process can only reduce the values of M_j^i, $i = 0, 1, 2, \ldots$, because of the min(.) function in (5.21), which in turn can only reduce the values of W_k^i, $i = 0, 1, 2, \ldots$. This leads us to the conclusion that even if all the upper

bounds on the M_j values happen to change, which is highly unlikely, this algorithm still converges after a finite number of steps. We leave any study on the speed of convergence of such an iterative algorithm for future work.

The algorithm for the lower bound on mean waiting time is very similar in nature, except that we make sure that the initial guess on M_j values are greater than or equal to zero, and then we proceed in the exact same way as we did for the upper bound on mean packet waiting time.

5.6 Experimental Results

We perform several simulation experiments to validate the results of the M/G/FQ analysis presented in this chapter. A word of caution is essential at this point. Because it is quite difficult to find many FQ algorithms with the *exact* same fairness bound, we decided to study three EFTF FQ algorithms (namely SCFQ, WFQ and SPFQ) using the fairness bound $\overline{\psi_k} = \overline{L_k}/r_k$ even though this bound is exact only for SCFQ. These three algorithms have close enough fairness bounds that we can safely use only one of them to illustrate the points we are trying to make.

5.6.1 Experiment: Bounding the Delay of Multiple FQ Algorithms

In this experiment, a FQ server with a total output link capacity of 10 Mb/s supports four incoming Poisson streams under SCFQ, WFQ or SPFQ. The reserved rates for the different connections are: $r_1 = r_2 = r_3 = r_4 = 2.5$ Mb/s. The first source is tagged and its mean arrival rate is varied between 0.25 and 2 Mb/s, while all the other three sources transmit at a fixed mean rate of 2.5 Mb/s, 2.75 Mb/s and 2.75 Mb/s, respectively. The packet length distribution is uniform and ranges between 4000 bits and 12000 bits per packet (with an average of 8000 bits, or 1000 bytes, per packet).

The results of the experiment are shown in Figure 5.1(a), in which the simulation-generated mean waiting time for the tagged flow is displayed versus load under the different FQ policies. We also show in this figure the analytical upper and lower bounds derived for this case based on (5.18) and (5.11), respectively.

We notice from the results that the mean packet waiting times generated by SCFQ, WFQ and SPFQ are all bounded by the upper and lower bounds of the M/G/FQ analysis irrespective of the incoming load value. This emphasizes the fact that the delay bounds derived here actually accommodate all FQ algorithms that exhibit the same (or nearly the same) fairness bound irrespective of their internal operations, and as such the delay bounds in (5.18) and (5.11) are reasonably tight.

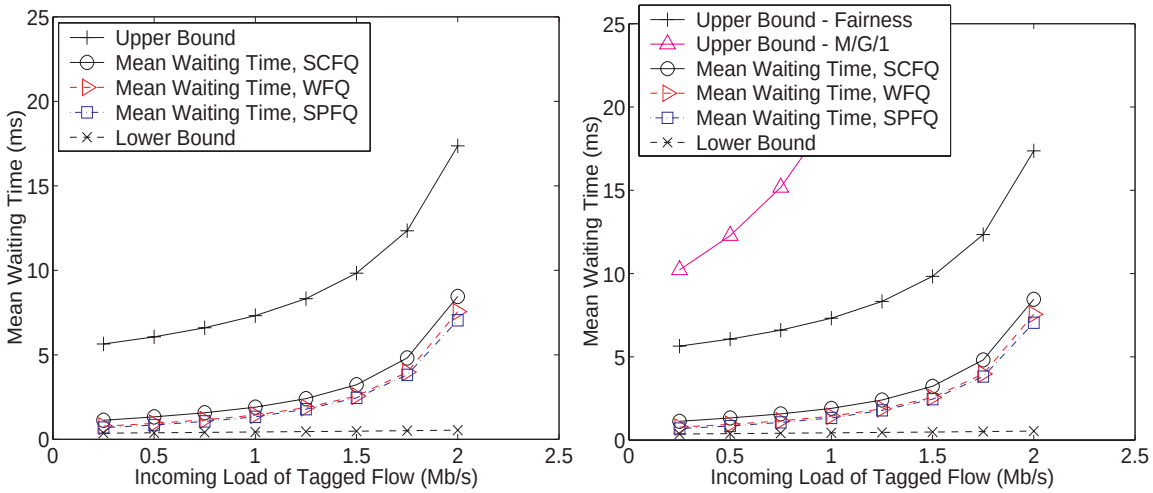

Figure 5.1: Mean Waiting Time versus Incoming Load for four Poisson streams under a heavily loaded system (a) only the improved upper bound in (5.18) is shown; and (b) the two components (5.10) and (5.17) of the improved upper bound are shown.

It is interesting to illustrate in this situation the difference in behavior between the two components that make up the improved upper bound in (5.18). Those two components are those in (5.10) and (5.17), both of which are shown separately for this experiment in Figure 5.1(b). The results in Figure 5.1(b) confirm our earlier argument that the bound in (5.10) is reasonably tight when the non-tagged sessions are greedy all the time, which is the case here. However, in such scenario the bound in (5.17) derived using the M/G/1 conservation law is not tight enough. Since we select the minimum of those two bounds in (5.18), we end up with a tight upper bound as a final result. We show in the next section that the situation is different when the non-tagged sessions are not heavily loaded. In such circumstances, the upper bound derived using the M/G/1 conservation law provides a better bound than that in (5.10).

Notice that the upper bound on mean waiting time in Figure 5.1 was calculated using the improved upper bound on fairness as per (5.10) not (5.9). As we can see, it still bounds the mean waiting time for all tested FQ algorithms. This strengthens the validity of the assumption we made earlier in Section 2.5.2 to improve the upper bound on $\overline{\delta_{kj}}$.

The lower bound in Figure 5.1(a) is also reasonably tight for low to moderate incoming load values. However, it does not show the same dramatic behavior as the incoming load approaches its limits when compared to the actual mean waiting time curves. This is the price we have to pay to obtain a reasonable mathematical formula as that in (5.11) for the lower bound on mean waiting time.

As a final observation, let us compare our stochastic upper bound shown in Figure 5.1 to the corresponding deterministic worst-case delay bound that the current literature provides. A word of caution is necessary at this point. The deterministic upper bound is derived assuming a regulated traffic scenario (e.g., one that adheres to a leaky bucket), while our stochastic upper bound is derived assuming purely random (*unregulated*) Poisson arrivals. Since those two types of incoming traffic have different mathematical descriptions, we need to introduce some sort of approximation to be able to compare the two upper bounds to each other.

The approximation we use here is to slightly modify the Poisson distribution so it can be described by a leaky bucket for the purpose of calculating the deterministic upper bound. In other words, as far as the deterministic upper bound is concerned, we use an approximate *regulated* Poisson arrival process rather than the actual purely random Poisson arrivals. We can do that simply by neglecting the tail of the Poisson probability distribution (for example neglect the arrivals after the 99% quantile of the distribution). Of course, we make sure to maintain the same mean arrival rate for both the original Poisson distribution (used for our stochastic bound) and the approximate regulated Poisson distribution (used for the deterministic bound).

Notice that the above approximation means that we neglect the probability of infinite arrivals in an arbitrary period of time (which is not a Poisson distribution in the true mathematical sense). However, the probability of infinite arrivals is very small, especially when running a simulation experiment for a finite period of time. In addition, this is the only way we can provide a meaningful comparison between the two upper bounds. Also notice that this comparison should work to the advantage of the deterministic upper bound because a small portion of the arrivals are dropped when calculating such bound.

The result of the comparison between the stochastic and deterministic upper bounds for the parameters of this experiment is shown in Figure 5.2. In this figure, we display the deterministic upper bound for WFQ, which is the smallest among all the three FQ algorithms tested in this experiment. It is clear from the comparison that the stochastic upper bound provides a much better estimation and tighter bounding of the *mean* waiting time of fair scheduling systems under steady-state normal operating conditions.

5.6.2 Experiment: Reducing the Load on the FQ System

In the previous experiment, we saw the dramatic increase of the mean packet waiting time as the utilization factor ρ_k of the tagged flow starts to increase (see Figure 5.1(a)). This was the behavior we would expect by looking at (5.10), which describes the upper bound on mean waiting time.

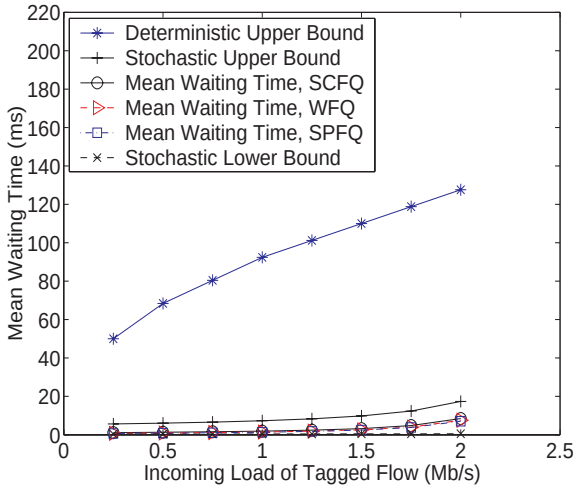

Figure 5.2: A comparison between the Stochastic and corresponding Deterministic upper bounds on Mean Waiting Time.

However, a key part contributing to this behavior is the fact that all flows other than the tagged one are transmitting at or above their nominal reserved rate, i.e., they are greedy. The aim of this experiment is to investigate the effects of reducing the load of the non-tagged flows on the results of the FQ system.

For this experiment, we use the exact same parameters as in Section 5.6.1 with the exception that all flows other than the tagged one are set to transmit at a fixed mean rate of 2 Mb/s instead of 2.5 Mb/s or 2.75 Mb/s. The results are shown in Figures 5.3(a) and (b), the scale of which are set to match Figures 5.1(a) and (b). Figure 5.3(b) shows the two upper bounds (5.10) derived suing the fairness bound and (5.17) derived using the M/G/1 conservation law, while Figure 5.3(a) shows the final improved upper bound in (5.18).

In this experiment the load on the FQ system is smaller than that in Section 5.6.1. Under such circumstances, all three studied FQ policies give comparable performance because the FQ server is free for a considerable amount of time. The lower bound is reasonably tight. The upper bound in (5.10), on the other hand, cannot completely adjust at higher input load values. The reason for this is that when the load on the non-tagged flows is reduced, the mean packet count $N_j, j \in \mathbf{J}$, drops dramatically. This means that neglecting the $N_j + \lambda_j W_k$ terms in (5.6) to derive (5.10) in such a case results in a weaker upper bound. However, in such a situation, the upper bound in (5.17) provides a tighter upper bound on the mean waiting time because it takes into consideration the arrival rates at the non-tagged flows. The final result is an improved upper bound (5.18) in this experiment too.

Although the more elaborate iterative algorithm described in Section 5.5 managed to converge in both experiments we conducted in Sections 5.6.1 and 5.6.2, unfortunately

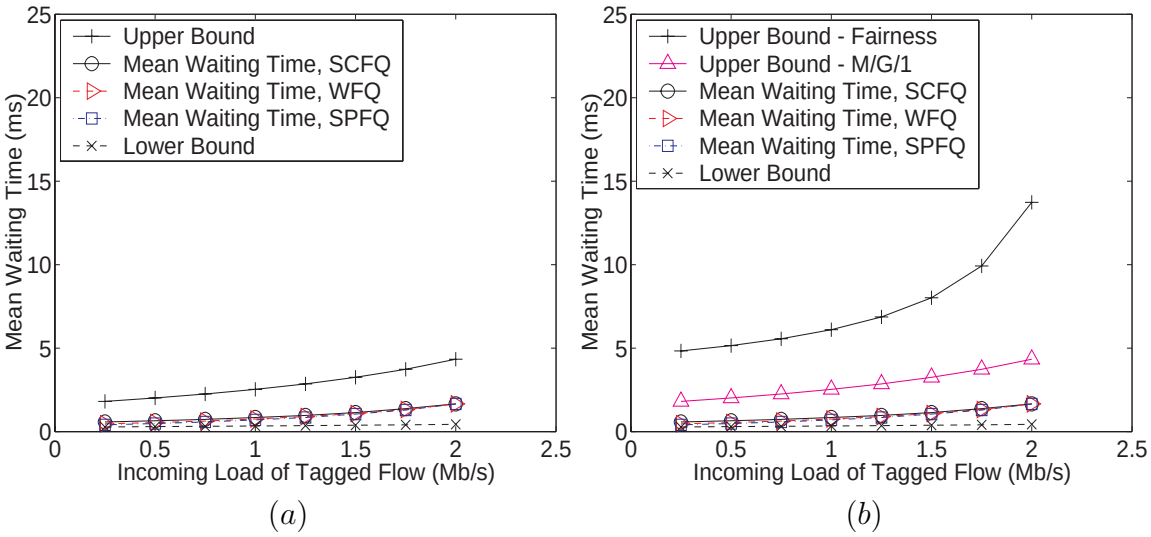

Figure 5.3: Mean Waiting Time versus Incoming Load for four Poisson streams under a partially loaded system (a) only the improved upper bound in (5.18) is shown; and (b) the two components (5.10) and (5.17) of the improved upper bound are shown.

it did not provide any enhancements on the obtained upper bounds. This is mainly because the upper bounds on mean delay for the non-tagged flows were not small enough to cause a change in the min(.) function in (5.21) required for the iterative process to converge to a new smaller upper bound. However, this does not preclude the possibility that in other situations the iterative algorithms might be useful. We were not motivated to create a special case where the iterative algorithm would actually improve performance, since simpler results can be obtained by using the lower bound as a good quick estimate for the mean waiting time in such scenario.

5.6.3 Experiment: Different Reservations and Length Distributions

In this experiment, we test whether changing flow reservations or packet length distributions affects our upper and lower bounds on mean waiting time. We set up a FQ server with an output link capacity of 10 Mb/s that supports two incoming Poisson streams under SCFQ, WFQ or SPFQ. The reserved rates for the two flows are: $r_1 = 4$ Mb/s and $r_2 = 6$ Mb/s. The tagged flow is the first one, and its mean arrival rate varies between 0.5 and 3.5 Mb/s, while the other flow transmits at a fixed mean rate of 6 Mb/s. The packet length distribution is first set to a uniform distribution that ranges between 4000 bits and 12000 bits per packet, then to a Pareto distributions with parameters $\alpha = 2.5$ and $\beta = 4800$ (so that the mean packet length is 8000 bits per packet) and then set to an exponential distribution with an average packet length of 8000 bits per packet.

The results of the experiment are shown in Figures 5.4(a), (b) and (c) for the uniform, Pareto and exponential packet length distributions, respectively. The results clearly confirm that the delay bounds are working correctly for all three cases.

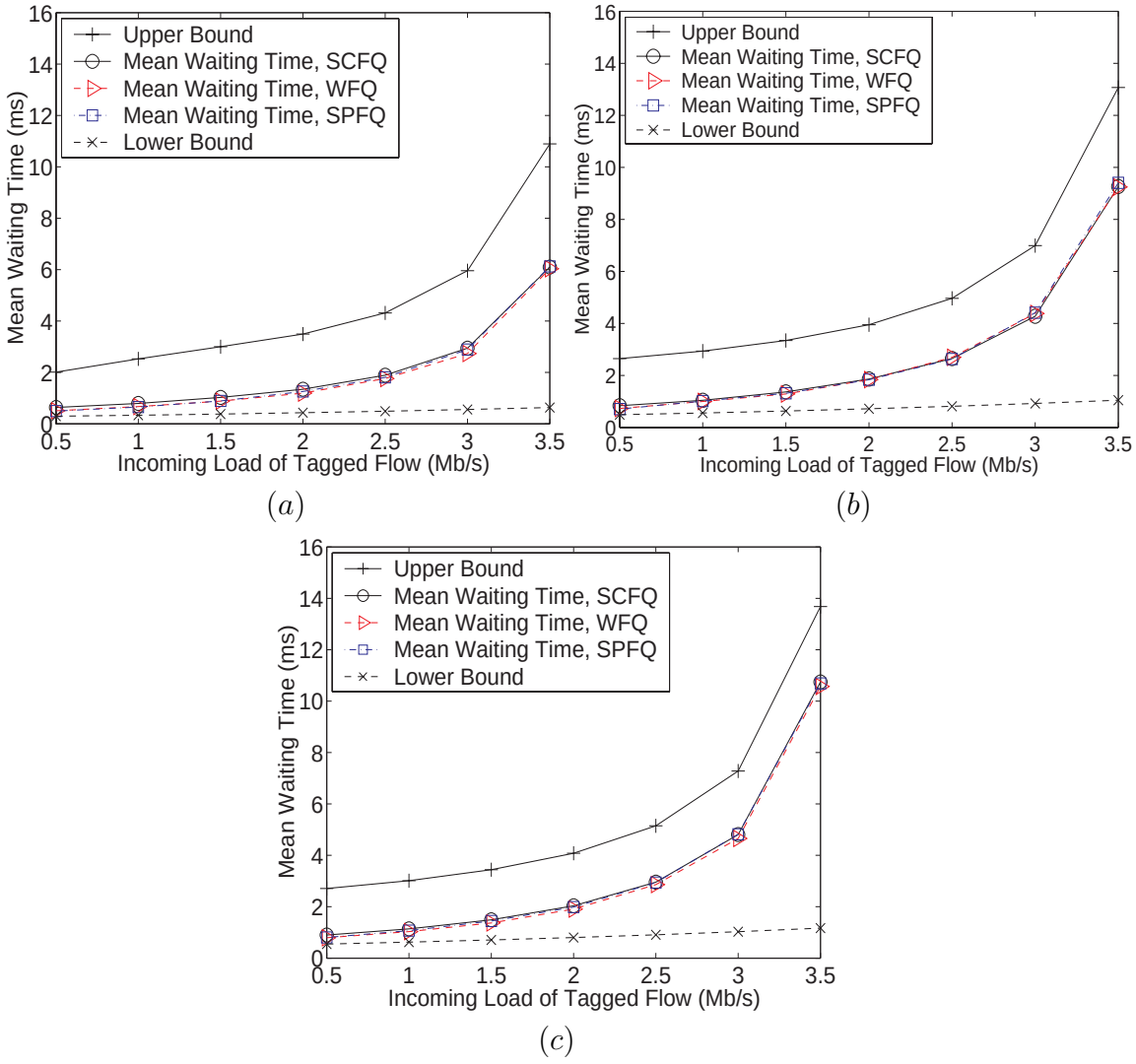

Figure 5.4: Mean Waiting Time versus Incoming Load for two Poisson streams with (a) uniform, (b) Pareto and (c) exponential packet length distributions.

Chapter 6

Performance Comparison of FQ Policies

IN the previous chapter we derived a mathematical result that describes the behavior of a general class of Fair Queueing (FQ) systems. Now we move on to study the difference in statistical performance between different FQ systems within this class.

We start by comparing the performance of three FQ policies (namely *Weighted Fair Queuing* (WFQ), *Self-Clocked Fair Queuing* (SCFQ) and *Starting Potential-based Fair Queuing* (SPFQ)) under different flow setup scenarios. We try to answer the question: how differently will those algorithms perform when faced with random (unregulated) Poisson arrivals? Our study shows that *if none* of the incoming flows exceeds its allocated capacity, all the above three algorithms exhibit virtually the same mean waiting time. This applies even though the three algorithms have different deterministic delay bounds under regulated traffic scenarios.

This leads us to the conclusion that under normal operating conditions, many FQ algorithms (at least the three studied here) perform quite the same when faced with random arrivals. This has serious implications for the applications of such scheduling policies, since performance (on *average sense*) is not the main differentiating factor between the different FQ algorithms. Rather, fairness and complexity of implementation become more significant criteria.

We also compare the performance of the above packet-based FQ algorithms to that of the reference Generalized Processor Sharing (GPS) algorithm, which is considered the origin of all fair scheduling policies. We illustrate some of the similarities and differences between such scheduling policies under Poisson arrivals.

Finally, we introduce the concept of *virtual capacity allocation*, which we use to derive some useful insights into the operation of packet-based FQ policies. We use this concept to show that FQ algorithms exhibit some sort of *bias* towards flows

with higher reservations. This is an important observation because FQ algorithms were originally designed to protect against misbehaving flows (i.e., flows exceeding their allocated capacity) during congestion periods, not to discriminate between flows under normal operating conditions.

6.1 Comparison between Three Packet-based FQ Policies

In this section, we perform different simulation experiments to compare the performance of three FQ algorithms: WFQ, SCFQ and SPFQ. We carefully select appropriate experiments with a wide range of parameters and flow setup scenarios to illustrate the specific characteristics of such FQ policies in a comprehensive manner. We will investigate how these three scheduling algorithms behave as we vary the number of incoming flows, arrival rates, flow reservations and packet length distributions.

We show that under various flow setup and reservation scenarios, the performance of the three algorithms in terms of *mean waiting time* is virtually the same. This is the case *only* when none of the supported flows is misbehaving (i.e., when none of the flows is exceeding its allocated capacity). We also experiment with cases where one or more of the supported flows are misbehaving and observe the differences in performance that arise between the three FQ policies in such case.

We carry out our simulations using Extend^{TM} simulation package. To perform the desired experiments we developed (and validated) *new* Extend^{TM} modules that implement the operations of the three FQ algorithms: WFQ, SCFQ and SPFQ. We then used the pseudo-random number generator (PRNG) provided by Extend^{TM} to generate the incoming packets. We ran the experiments for extended periods of time to minimize the statistical error in our results, a fact manifested by the absence of rugged curves in the resulting graphs (see Figures 6.1 – 6.10 below).

To keep our study realistic, we maintained a link capacity of 10 Mb/s and an average packet length of 1000 bytes (8000 bits) throughout our experiments. Such a scenario corresponds to an IP protocol running on top of an Ethernet link or a wireless MAC protocol.

6.1.1 Experiment: Two Flows, Different Reservations

In the first experiment, we start with the minimum possible number of incoming flows, and assume that they have different reservations. Hence, we consider a FQ server that supports two incoming Poisson streams under SCFQ, WFQ or SPFQ. The reserved rates for the different flows are set to $r_1 = 7$ Mb/s and $r_2 = 3$ Mb/s. The *tagged* flow is flow 1 and its mean arrival rate is varied between 0.5 and 8 Mb/s, while the second flow transmits at a fixed mean rate of 2 Mb/s. Notice that we

wanted to test a case where one of the flows (the tagged flow) transmits well below its reservation and then starts misbehaving (i.e., starts transmitting at a rate higher than its reservation). Notice, however, that even though the tagged flow exceeds its reservation, the stability of the queueing system is maintained because the total arrival rate is smaller than 10 Mb/s. We opt for a simple packet length distribution for this experiment, which is a uniform distribution that ranges between 500 and 1500 bytes (4000 – 12000 bits) for each packet.

Figure 6.1 shows the measured *mean waiting time* for both the tagged and the non-tagged flows under the three different FQ policies. The graph shows that the three algorithms perform practically the same when the tagged flow is transmitting below its reservation of 7 Mb/s. Once the tagged flow starts misbehaving (i.e., transmits at a rate higher than 7 Mb/s), however, the performance of the different FQ algorithms starts to differ. We make two interesting observations in this graph. First, when one of the flows starts to misbehave, the difference in performance between the three algorithms manifests itself as a difference in the mean waiting time incurred by the *non-misbehaving* flow. For the misbehaving flow, on the other hand, all FQ algorithms provide a dramatic increase in the mean waiting time, which will eventually approach infinity as shown in Figure 6.1. The second observation we make is that only WFQ performance starts diverging from the other two FQ policies at the onset of flow misbehavior. In Section 6.1.3, we show that this is not necessarily the case for all experiments.

It is worth mentioning at this point that *not all* FQ policies perform the same under M/G/FQ assumptions. To prove this, we show in Figure 6.2 the results of running the same above experiment under SFQ (an ESTF policy). Comparing this to the results of SCFQ, the difference in mean waiting time for the non-tagged flow is too obvious to neglect, even when all flows are transmitting well below their reserved capacities. This result should not come as a surprise because there are some fundamental differences between the operations of EFTF (e.g., WFQ, SCFQ and SPFQ) and ESTF (e.g., SFQ) algorithms. In ESTF policies, flows with smaller reservations are usually overprotected (i.e., experience a smaller mean waiting time in the queue) by the inherent design of the algorithm itself [14].

6.1.2 Experiment: Two Flows, Other Reservation Scenarios

In this experiment, we test the effect of changing the flow reservations on our mean waiting time results. Hence, we repeat the experiment in Section 6.1.1 with two incoming flows but now with different reservation scenarios. The cases we consider are summarized in Table 6.1 below. They cover all the three possibilities of $r_1 > r_2$, $r_1 = r_2$ and $r_1 < r_2$. The tagged flow is flow 1.

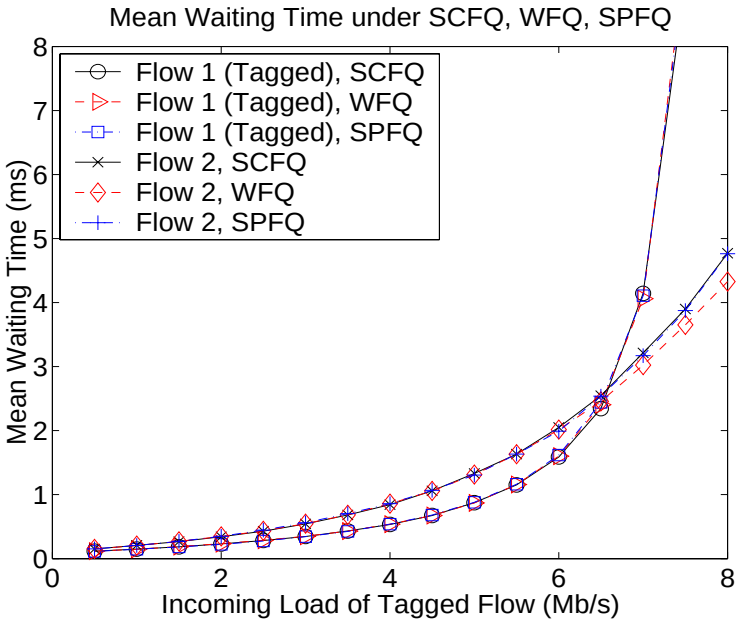

Figure 6.1: Mean waiting time for the tagged and non-tagged flows under SCFQ, WFQ and SPFQ scheduling policies.

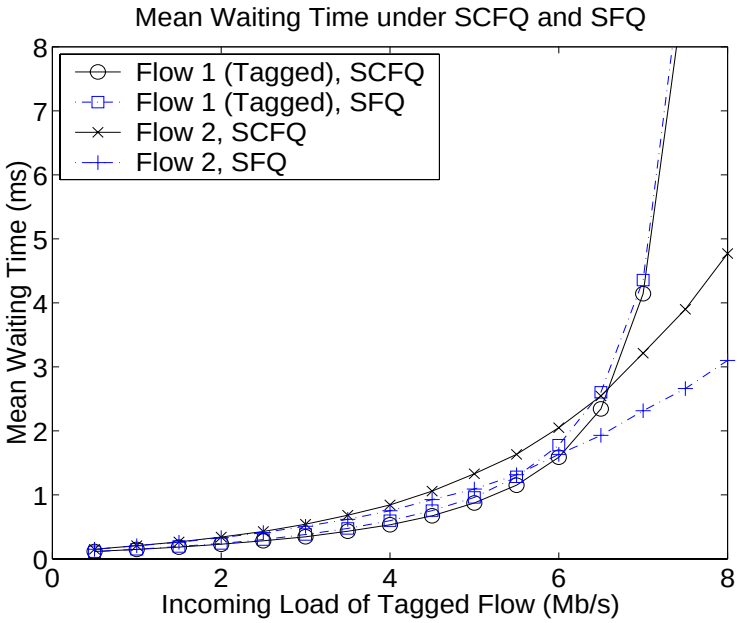

Figure 6.2: Mean waiting time for the tagged and non-tagged flows under SFQ and SCFQ scheduling policies.

Table 6.1: Reservation Scenarios for Experiment 6.1.2 (Values in Mb/s).

Case	r_1	r_2	$\lambda_1\bar{L}$	$\lambda_2\bar{L}$
A	7	3	0.5 – 8	2
B	5	5	0.5 – 6	4
C	3	7	0.5 – 4	6

Figure 6.3 shows the mean waiting time for the tagged flow (part (a)) and the non-tagged flow (part (b)) for all three FQ policies under the different scenarios A, B and C described in Table 6.1. Figure 6.3 clearly illustrates that irrespective of the reservations made by the different flows, as long as none of the flows is misbehaving, all three FQ algorithms (WFQ, SCFQ and SPFQ) perform virtually the same under random arrivals.

Again, notice in Figure 6.3 that the mean waiting time of the tagged flow dramatically increases under all FQ policies as the tagged flow misbehaves, while such misbehavior effects the mean waiting time of the other non-misbehaving flows such that WFQ performance starts to diverge from the other FQ policies.

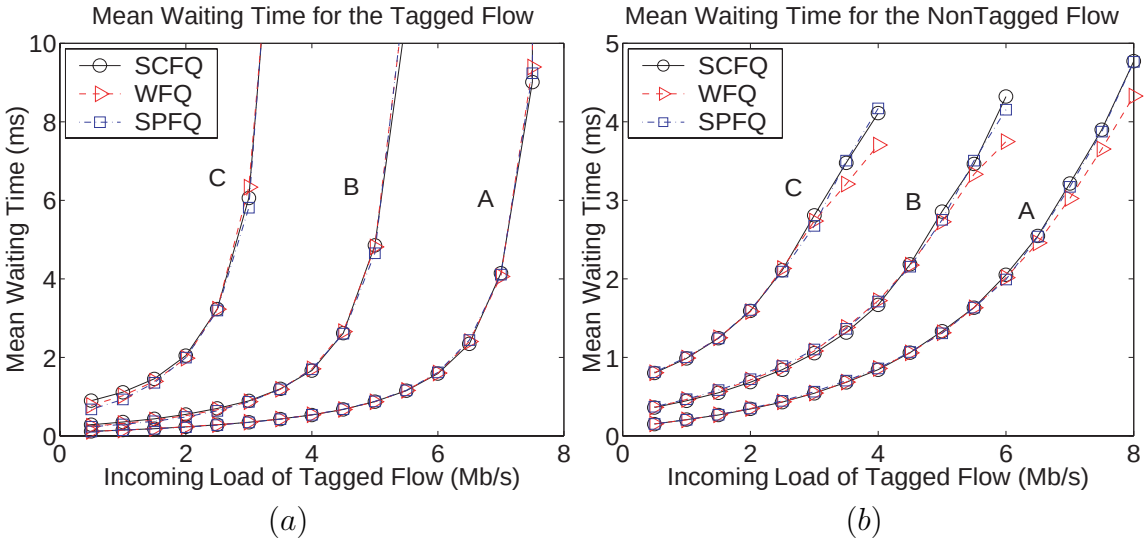

Figure 6.3: Mean waiting time for the (a) tagged flow and (b) non-tagged flow under the different reservation scenarios in Table 6.1.

6.1.3 Experiment: Three Flows, Log Normal Packet Length Distribution

For the next experiment, we investigate the effects of changing both the number of incoming flows to the scheduler and the packet length distribution. We set up a FQ server that supports *three* incoming Poisson streams under SCFQ, WFQ or

SPFQ. The packet length distribution is set to a *Log Normal* distribution with a mean packet length of 1000 bytes (8000 bits) and a standard deviation of twice the mean (i.e., 16000 bits). We pick a case where the tagged flow makes a reservation that is nearly equal to the reservations made by the other non-tagged flows. A different scenario will be tested in the next experiment. The reservations we opted for are r_1 = 3 Mb/s, r_2 = 2 Mb/s and r_3 = 5 Mb/s. The *tagged* flow is flow 1, and the different flow arrival rates we consider are summarized in Table 6.2 below. Notice that the first two scenarios A and B in Table 6.2 represent the case when the tagged flow is the one that misbehaves in part of the experiment, while in the next two scenarios C and D, the *non-tagged* flows are the ones that misbehave all the time.

Table 6.2: Flow Arrival Rates for Experiment 6.1.3 (Values in Mb/s).

Reservations	3	2	5
Case	$\lambda_1 L$	$\lambda_2 L$	$\lambda_3 L$
A	0.5 – 6	1	3
B	0.5 – 4	1.5	4.5
C	0.5 – 2	2.5	5.5
D	0.5 – 1	2.5	6.5

Figure 6.4 shows the mean waiting time of the *tagged* flow for the different scenarios of this experiment (the non-tagged flow curves are omitted to avoid cluttering the graph). Part (a) of the figure shows the first two cases A and B, while part (b), on the other hand, zooms into the last two cases C and D. The results clearly show that under random arrivals, the tagged flow exhibits nearly the same performance under the different FQ policies so long as *none* of the other (non-tagged flows) is misbehaving. This seems to be the case irrespective of the packet length distribution one might encounter.

It is also interesting to notice here that SCFQ is the FQ algorithm that has different performance than the other two when the non-tagged flows are misbehaving. This is different than what happened in Section 6.1.1, where WFQ was the FQ algorithm that had different performance.

6.1.4 Experiment: Four Flows, Exponential Packet Length Distribution

To complete our deduction process, we again increase the number of incoming flows and change the packet length distribution. We perform an experiment where the FQ server supports four incoming Poisson streams, with packet lengths that are *exponentially* distributed. We maintain a mean packet length of 1000 bytes (8000 bits) and tag flow 1. In this experiment, we chose the tagged flow to be the dominant flow in terms of reservation. Hence, we set the flow reservations to r_1 = 6 Mb/s,

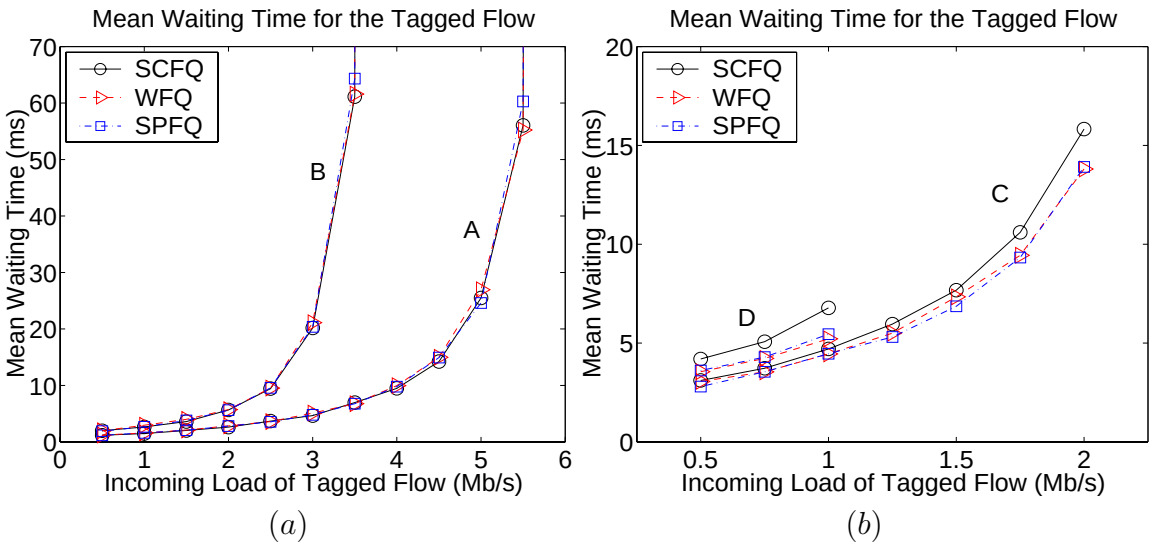

Figure 6.4: Mean waiting time for the tagged flow under the different flow setup scenarios in Table 6.2. Part (a) shows scenarios A and B, while part (b) shows scenarios C and D.

$r_2 = 1$ Mb/s, $r_3 = 2$ Mb/s and $r_4 = 1$ Mb/s. Table 6.3 shows the different flow arrival rates for this experiment, where in the first two scenarios A and B, the tagged flow heavily utilizes its large reservation, while in the second two cases C and D, the tagged flow underutilizes its reserved capacity, at which time the non-tagged flows are misbehaving (exceeding their reservations).

Table 6.3: Flow Arrival Rates for Experiment 6.1.4 (Values in Mb/s).

Reservations	6	1	2	1
Case	$\lambda_1 L$	$\lambda_2 L$	$\lambda_3 L$	$\lambda_4 L$
A	0.5 – 9	0.25	0.5	0.25
B	0.5 – 7	0.75	1.5	0.75
C	0.5 – 5	1.25	2.5	1.25
D	0.5 – 3	1.75	3.5	1.75

The results of this experiment are shown in Figures 6.5(a) and (b), which again strengthen our argument that the above three FQ algorithms produce nearly the same mean waiting times irrespective of the numbers of supported flow and irrespective of packet length distributions, *except* when one or more of the supported flows is misbehaving.

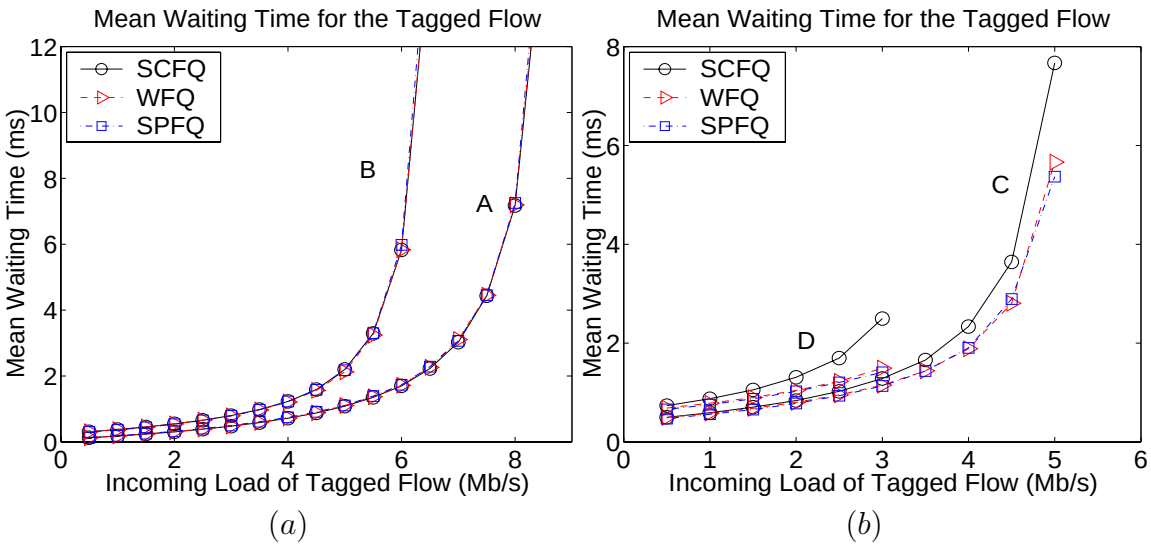

Figure 6.5: Mean waiting time for the tagged flow under the different flow setup scenarios in Table 6.3. Part (a) shows scenarios A and B, while part (b) shows scenarios C and D.

6.2 Comparison with the GPS Policy

One might ask the question: If the performance of the above three FQ algorithms that are trying to emulate GPS actually converge to one behavior, is that behavior the same as that of GPS? To answer this question, we redo the experiment in Section 6.1.1, but now under an idealized GPS policy. We compare the performance of GPS to that of SCFQ in Figure 6.6.

The main conclusion we can derive from Figure 6.6 (combined with Figure 6.1) is that although WFQ, SCFQ and SPFQ are attempting to emulate GPS, they fail to emulate its exact behavior under random arrivals. This is because GPS refers to an *idealized* solution, while the other algorithms are designed for a more practical *packet-based* one. In other words, even though the performance of many EFTF packet-based FQ policies ended up converging to a single behavior, this behavior is different from that of the GPS algorithm.

Another interesting observation we can make in Figure 6.6 is that the GPS system produces a smaller mean waiting time for low reservation flows. The reason for this behavior is that in GPS, the *residual service time* seen by an arriving packet is dependent only on the mean arrival rate for that particular flow. For packet-based fair scheduling policies, on the other hand, there is an extra residual time added due to packets arriving from the other flows supported by the scheduler (cf. (4.1) and (4.2)). Hence, flows that transmit at low rates (corresponding to low reservation flows) incur smaller waiting times under the GPS policy.

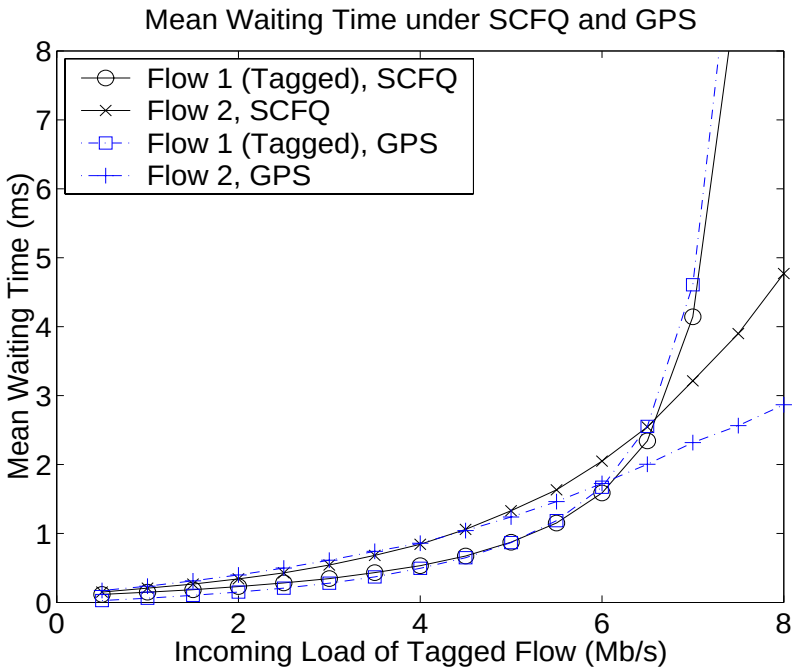

Figure 6.6: Mean waiting time under SCFQ (a packet-based scheduling policy) and GPS (an idealized scheduling policy).

This is an interesting observation because comparing Figures 6.2 and 6.6, we notice some similarity between GPS and SFQ behavior in that they both protect low reservation flows. Since our main focus in this research effort is investigating EFTF FQ algorithms (rather than ESTF algorithms), we leave for future wok investigating whether or not this means that ESTF algorithms are more capable of emulating GPS behavior under random arrivals.

6.3 Virtual Capacity in Queueing Systems

In this section, we introduce a new concept in queueing systems, which we call *virtual capacity allocation*. As we explain shortly, this new concept allows us to derive some useful insights into the operation of packet-based FQ policies by comparing them to other well-known queueing systems.

The virtual capacity concept allows us to look at the mean waiting time produced by an M/G/FQ system as if it was a result of an equivalent M/G/1 system (more details in Section 6.3.1). The interesting thing about virtual capacity is that it is a very sensitive metric to variations in mean waiting times, which is a desirable property when comparing the performance of different queueing disciplines.

6.3.1 The Concept of Virtual Capacity

Earlier in Chapters 4 and 5, we found that the mean waiting time for a tagged flow k in a FQ system is given by,

$$W_k = R + \frac{1}{\mu}\left(N_k + \sum_{j \in \mathbf{J}} M_j\right) \tag{6.1}$$

Let us assume in (6.1) that $M_j = f_{jk} \cdot N_k$, where f_{jk} is a factor specific to the pair of flows j and k, $j, k \in \mathbf{K}$. We assume such factors f_{jk} to be unknown for now and, of course, are dependent on the loading conditions of the FQ system. We can thus write (6.1) as follows,

$$W_k = R + \frac{1}{\mu}\left(1 + \sum_{j \in \mathbf{J}} f_{jk}\right) N_k = R + \frac{1}{\mu'_k} N_k \tag{6.2}$$

where,

$$\mu'_k = \frac{\mu}{1 + \sum\limits_{j \in \mathbf{J}} f_{jk}} \tag{6.3}$$

is the *effective* service rate that packets from queue k observe. We can now define the *virtual capacity* c_k (in bits/second) seen by flow k as follows,

$$c_k = \mu'_k \cdot \overline{L_k} \tag{6.4}$$

where $\overline{L_k}$ is the mean length of data packets arriving at queue k, $k \in \mathbf{K}$. Notice that c_k represents the share of the capacity that flow k sees if it was in a *virtual* M/G/1 system with its own separate buffer and its own server. Notice that in general $\sum_{k=1}^{K} c_k \neq C$ because we are describing a totally different (virtual) system. We will show, however, that the sum of virtual capacities actually add up to the link capacity only in one special case, when the *first-in first-out* (FIFO) scheduling algorithm is being used.

The concept of virtual capacity is useful in visualizing the operations of FQ systems, because it converts the result for waiting time in M/G/FQ to that of an equivalent M/G/1 system. To see this, apply Little's law to convert (6.2) into,

$$W_k = \frac{R}{1 - \frac{\lambda_k}{\mu'_k}} = \frac{R}{1 - \frac{\lambda_k \overline{L_k}}{c_k}} \tag{6.5}$$

To better understand virtual capacity, let us attempt to solve for a specific case of virtual capacity allocation using the conservation law of M/G/1 systems. Such conservation law states that [31],

$$\sum_{k=1}^{K} \rho_k W_k = \frac{R\rho}{1-\rho} \tag{6.6}$$

where $\rho = \sum_{k=1}^{K} \rho_k$. In our study, since packets from different flows have the same packet length distribution, (6.6) reduces to,

$$\sum_{k=1}^{K} \lambda_k W_k = \lambda \frac{R}{1-\rho} = \lambda W \tag{6.7}$$

where W is the mean waiting time of the aggregate of all packets arriving at the queueing system. Let us consider the simple case of only two flows, i.e., $\mathbf{K} = \{1, 2\}$. In such a case, (6.7) can be expanded using effective capacities as follows,

$$\lambda_1 \frac{R}{1-\frac{\lambda_1 \overline{L}}{c_1}} + \lambda_2 \frac{R}{1-\frac{\lambda_2 \overline{L}}{c_2}} = \lambda \frac{R}{1-\frac{\lambda \overline{L}}{C}} \tag{6.8}$$

After some algebra, (6.8) becomes,

$$\left(\frac{\left(\overline{L}\right)^2}{c_1 c_2} - \frac{\overline{L}}{\lambda_2 c_1} - \frac{\overline{L}}{\lambda_1 c_2} \right) = \left(\frac{\left(\overline{L}\right)^2}{c_1 C} - \frac{\overline{L}}{\lambda c_1} - \frac{\overline{L}}{\lambda_1 C} \right) + \left(\frac{\left(\overline{L}\right)^2}{c_2 C} - \frac{\overline{L}}{\lambda_2 C} - \frac{\overline{L}}{\lambda c_2} \right) \tag{6.9}$$

There are many possible solutions for (6.9). One possibility is to separately equate the $\overline{L}$ and $\left(\overline{L}\right)^2$ terms on both sides of the equation. This results in two new equations, the first being,

$$\frac{1}{c_1 c_2} = \frac{1}{c_1 C} + \frac{1}{c_2 C} \tag{6.10}$$

which after manipulation is equivalent to,

$$c_1 + c_2 = C \tag{6.11}$$

which is the case of effective capacities adding up to C. As we will prove shortly, this applies *only* when FIFO queueing is used, and is the only case when the effective capacities add up to the total link capacity C. The second equation that we can derive from (6.9) is then,

$$\frac{1}{\lambda_2 c_1} + \frac{1}{\lambda_1 c_2} = \left(\frac{1}{\lambda c_1} + \frac{1}{\lambda_1 C}\right) + \left(\frac{1}{\lambda_2 C} + \frac{1}{\lambda c_2}\right) \tag{6.12}$$

or,

$$\frac{\lambda_1}{\lambda} c_2 + \frac{\lambda_2}{\lambda} c_1 = \frac{\lambda_1}{\lambda}\frac{\lambda_2}{\lambda}(c_1 + c_2) + \frac{c_1 c_2}{C} \tag{6.13}$$

If we substitute (6.11) into (6.13) to solve for c_1, we get,

$$c_1^2 - \left(\frac{2\lambda_1}{\lambda} C\right) \cdot c_1 + \left(\frac{\lambda_1}{\lambda} C\right)^2 = 0 \tag{6.14}$$

This is a quadratic equation that has a *unique* solution in c_1, given by,

$$c_1 = \frac{\lambda_1}{\lambda} C \tag{6.15}$$

In a similar argument, we can show that $c_2 = \lambda_2 C/\lambda$. This means that in this particular solution, virtual capacity is split between the two flows based on their *demand* (arrival rate) regardless of any reservations they might have made. One can immediately see that this corresponds to the well-known FIFO scheduling policy. To prove that this is indeed the case, we substitute (6.15) into (6.5) to calculate the mean waiting time observed by flow 1 packets, which becomes,

$$W_1 = \frac{R}{1 - \frac{\lambda \overline{L}}{C}} \tag{6.16}$$

Similarly, for flow 2 packets, the mean waiting time is $W_2 = R/\left(1 - \lambda \overline{L}/C\right)$.

Notice that the mean waiting times for both flows 1 and 2 are exactly the same and are equal to the mean waiting time of the aggregate of both flows. This *is* the behavior of FIFO scheduling.

Finally, we notice from (6.11) that this is the only case when the effective capacities add up to the link capacity C. It is easy to prove this statement because if (6.15) does not hold (i.e., this is not a FIFO policy), then (6.12) would not hold either (remember that the solution for the quadratic equation in (6.14) was *unique*). If (6.12) does not hold, then it follows that (6.11) does not hold as well, because both are derived from a single equation (6.9), which completes the proof.

6.3.2 Experiment: Virtual Capacity Allocation in FIFO and SCFQ

We now perform a simulation experiment to illustrate the differences in virtual capacity allocation in FIFO and FQ policies.

We set up two Poisson steams to be served by either SCFQ or FIFO. The reservations for SCFQ are set to $r_1 = 7$ Mb/s and $r_2 = 3$ Mb/s, and the packet length distribution is kept uniform with a mean of 1000 bytes. The *tagged* flow is flow 1 and its mean arrival rate is varied between 0.5 and 8 Mb/s, while the second flow transmits at a fixed mean rate of 2 Mb/s. These parameters are identical to those we used in the experiment in Section 6.1.1.

The mean waiting times for the tagged and non-tagged flows under both queueing disciplines are shown in Figure 6.7. As expected, both the tagged and non-tagged flows in FIFO received the same mean waiting time, while they were treated differently in SCFQ. Although the difference in mean waiting time between the two scheduling policies is identifiable, we can better understand the specific operations of both queueing algorithms by looking at the virtual capacity diagram shown in Figure 6.8. Figure 6.8 is derived from Figure 6.7 by a straightforward application of (6.5).

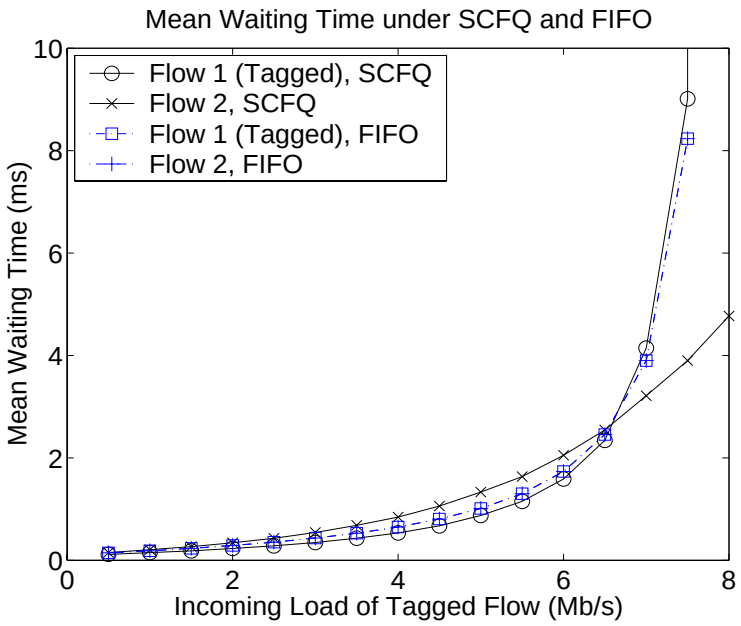

Figure 6.7: Mean waiting time for the tagged and non-tagged flows under SCFQ and FIFO.

The first observation we make about Figure 6.8 is that even though the sum of virtual capacities under FIFO queueing is equal to C, this is not the case under SCFQ; instead we have $c_1 + c_2 > C$.

Another observation we make is how sensitive virtual capacities are to slight variations in the mean waiting time. It is quite clear that the differences between FIFO

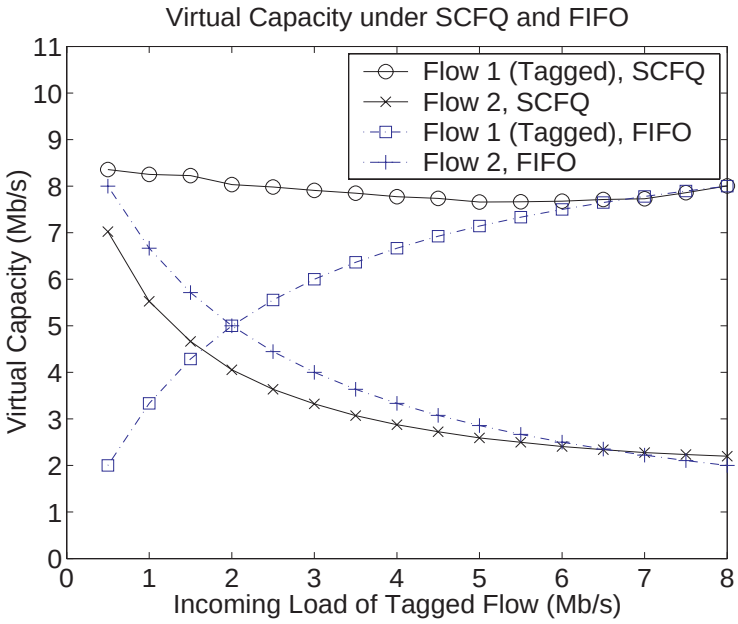

Figure 6.8: Virtual capacity allocation under SCFQ and FIFO.

and SCFQ operations are much more obvious in Figure 6.8 compared to that of Figure 6.7, especially at low load conditions. This makes virtual capacities a very attractive option to illustrate the differences between various queueing systems.

As we explained earlier (cf. (6.15)), the virtual capacity allocation in FIFO is based on the flow *demand* (or arrival rate). Although Figure 6.8 shows this clearly, Figure 6.9 illustrates the concept even better. In Figure 6.9 we plot the ratio of arrival rate to that of virtual capacity for each flow. You can see that for FIFO, this ratio is kept equal for both flows at all times.

Virtual capacity allocation in SCFQ (and similar FQ policies), on the other hand, is a much more complex endeavor. At low loading conditions, the allocation of such capacities is such that the higher reservation flow (the tagged flow) receives the best treatment possible. This is clear from Figures 6.8 and 6.9, where you see that the flow with lower reservation is actually *mistreated* (compared to FIFO queueing) so that the flow with higher reservation receives a lower delay.

This process of protecting the high reservation flow is an interesting one to understand, because FQ algorithms were originally designed to *protect* against *misbehaving* flows, not to favor a certain flow against the other.

One might expect that this process of mistreating the low reservation flow to stop when we reach the condition of $\lambda_1/r_1 = \lambda_2/r_2$, i.e., when flow 1 is transmitting at 4.67 Mb/s. Figures 6.8 and 6.9 show that this is not necessarily the case. The mistreatment of the low reservation flow continues long after this point. However, once

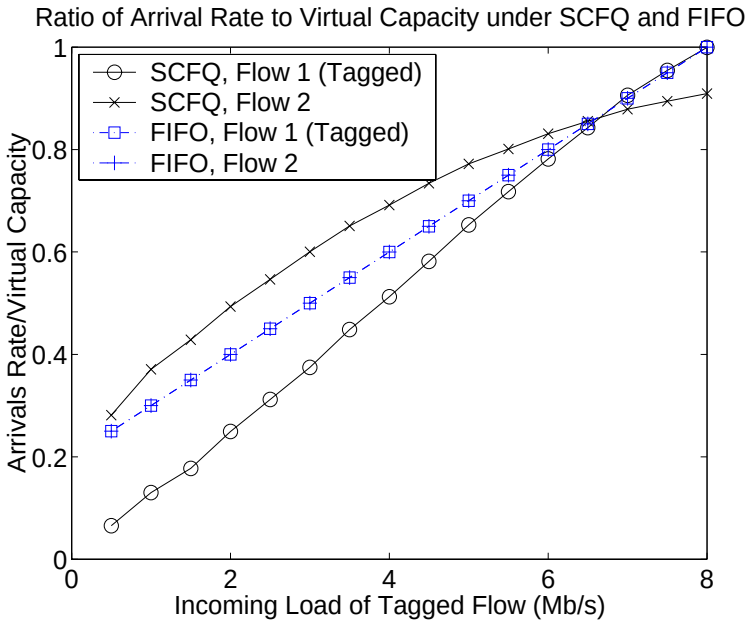

Figure 6.9: Ratio of mean arrival rate to virtual capacity under SCFQ and FIFO.

the high reservation flow starts to misbehave (when its mean arrival rate approaches its reservation of 7 Mb/s), mistreatment of the low reservation flow stops.

6.3.3 Experiment: Two Flows, Same Reservations

The low reservation flow mistreatment phenomenon in FQ policies grows smaller and smaller as the difference in reservation between the two flows gets smaller. In this section, we perform an experiment to illustrate the limiting case of two flows having equal reservations $r_1 = r_2 = 5$ Mb/s. The *tagged* flow is flow 1 and its mean arrival rate is varied between 0.5 and 6 Mb/s, while the second flow transmits at a fixed mean rate of 4 Mb/s. The parameters we use for this experiment are identical to those of case B in Table 6.1 of Section 6.1.2.

Figure 6.10 shows the virtual capacity allocation for both FIFO and SCFQ based on the mean waiting times recorded for this experiment. The results indicate that SCFQ behavior approaches that of FIFO queueing when all flows have the same reservation level.

6.4 Summary

Although we cannot perform all possible simulation experiments to study the stochastic performance of packet-based FQ algorithms, we have managed to carry out a wide range of such experiments that involve a variety of possible scenarios and flow setup

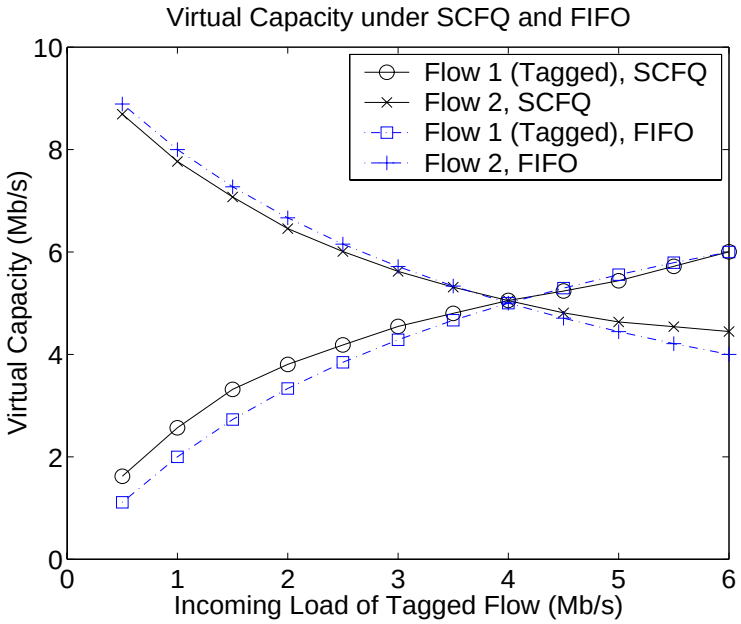

Figure 6.10: Virtual capacity allocation under both SCFQ and FIFO.

parameters. The results we obtained indicate that if none of the flows supported by the scheduler is misbehaving, then the mean waiting times of the three FQ algorithms WFQ, SCFQ and SPFQ are quite the same under random Poisson arrivals.

We also showed that the mean waiting time one expects from the idealized GPS is not exactly the same as that observed under WFQ, SCFQ or SPFQ.

Finally, using the concept of virtual capacity we confirmed the fact that FQ algorithms not only protect against misbehaving flows during congestion scenarios, but also provide some sort of bias (or protection) to high reservation flows at the expense of low reservation flows during normal operating conditions.

Chapter 7

Concluding Remarks and Future Work

WE finish up this dissertation with a quick summary of the contributions of this research effort and a few pointers to future work. We also introduce a totally new queueing algorithm that we stumbled upon during our research by coincidence. The study of this new queueing system is left entirely for future work but we mention it here because it is related to the general area of research we pursued in this work.

7.1 Summary of Contributions and Future Work

- **Contribution**: Introduced **a new QoS scheduling architecture for both IEEE 802.16 and DOCSIS**.

  Discussion: The new architecture supports diverse QoS guarantees for various flow types suggested by the two standards. It employs a dynamic minislot allocation scheme that should improve performance under varying load conditions, and it lends itself to a straightforward implementation in hardware, thus gaining a performance advantage over other software-based alternatives.

  Future Work: As we mentioned earlier, the process of demonstrating the efficiency and performance of this new scheduler was stalled by critical bugs in the DOCSIS module of OPNET. Because of no immediate feasible alternatives, we were forced to leave the process of verifying the performance of our scheduling architecture within a complete DOCSIS (or IEEE 802.16) environment for future work.

- **Contribution**: Presented **a new analysis method that produces simple and reasonably tight upper and lower bounds on mean waiting time of FQ algorithms under Poisson arrivals**.

Discussion: The analysis method uses the bounded fairness criterion of FQ algorithms (represented by the bounds on mean service lag distribution) in order to derive the desired bounds on mean waiting time and mean buffer occupancy. We showed several experiments that illustrate the validity of such delay bounds and how they accommodate different FQ policies with the same fairness criterion.

Future Work: We are currently investigating the possibility of deriving tighter bounds on the mean service lag for some of the well-known FQ policies and using those values to derive tighter bounds on mean waiting times experienced by such scheduling algorithms.

- **Contribution**: **Studied the stochastic performance of three packet-based FQ algorithms, namely: WFQ, SCFQ and SPFQ**.

  Discussion: We showed that if none of the incoming flows is misbehaving, then the mean waiting times of the above three algorithms are quite the same under random Poisson arrivals, even though they have different deterministic delay bounds. This is an important statement because it implies that more sophisticated FQ algorithms (with higher implementation complexities and smaller deterministic delay bounds) are not necessarily the best choice in certain applications where the main concern is average delay performance.

  Future Work: This result is also relevant to mathematical analysis of FQ algorithms. One can argue that since the above three FQ policies exhibit virtually the same performance under Poisson arrivals, analyzing one algorithm might be sufficient to provide a very good solution for all the other algorithms. Hence, one might focus on the FQ algorithm that exhibits the simplest (and most elegant) solution to find a less challenging answer that will work for other FQ algorithms as well.

- **Contribution**: **Compared the performance of the three packet-based FQ algorithms (WFQ, SCFQ and SPFQ) to that of the idealized GPS policy**.

  Discussion: We showed that the mean waiting time one expects from the idealized GPS is not exactly the same as that observed under WFQ, SCFQ or SPFQ.

  Future Work: This means that even if one finds an exact analysis for GPS, such analysis would not be totally applicable to practical packet-based FQ systems. An exact GPS analysis would be considered only an approximation to the actual packet-based FQ policies unless, of course, appropriate corrections are introduced.

- **Contribution**: During our study of FQ algorithms under random arrivals, **we noticed that ESTF FQ policies might be more capable of emulating GPS behavior than EFTF FQ algorithms**.

  Discussion: In ESTF FQ policies (e.g., SFQ), flows with smaller reservations are usually overprotected (i.e., experience a smaller mean waiting time in the queue compared to EFTF FQ algorithms). Low reservation flows also receive smaller mean waiting times in GPS. This raises the question of how close is the stochastic performance of ESTF algorithms to that of the idealized GPS policy?

  Future Work: This requires a further study that is focused on ESTF scheduling policies.

- **Contribution**: Introduced **the virtual capacity concept, and used it to derive some useful insights into the operation of packet-based FQ algorithms**.

  Discussion: We used the virtual capacity concept to show that FQ algorithms not only protect against misbehaving flows during congestion scenarios, but also provide some sort of bias (or protection) to high reservations flows at the expense of low reservation flows during normal operating conditions.

  Future Work: We are currently investigating the possibility of an approximate mathematical analysis of packet-based FQ algorithms under random Poisson arrivals. We believe that using the virtual capacity concept should make the analysis easier. Such analysis should be applicable to more than just one FQ policy (see Section 6.4).

7.2 H-F^{2}Q Queueing Algorithm

As we were developing a new simulation module for WFQ (one of the FQ algorithms studied in Chapters 5 and 6), we stumbled upon a new promising queueing discipline. It turns out that introducing a minor modification to the operations of WFQ transforms such a fair queueing policy into a different system that exhibits very interesting properties. We now call the new algorithm the Hybrid-FIFO/FQ system (or H-F^{2}Q for short) because it exhibits desirable properties from both WFQ and FIFO scheduling policies.

The description of this new queueing algorithm is intended to be informative only. The details of any investigation into the properties of such system will be left entirely for future work. The following sections describe the modifications required to transform WFQ into H-F^{2}Q and explain the preliminary observations we made about this new queueing discipline.

7.2.1 Modifications to Create H-F^{2}Q

For a review of the operations of packet-based FQ algorithms (including WFQ) the reader is referred to Chapter 2. The operations of such FQ systems are dependent on the *virtual time*, $v(t)$, which in WFQ is calculated using (2.2), which we repeat here for convenience,

$$v(t_2) - v(t_1) = \frac{C}{\sum\limits_{k \in B(t_1,t_2)} r_k} \cdot (t_2 - t_1) \tag{7.1}$$

where C is the total output link capacity in bits/s, $B(t_1,\, t_2)$ is the set of backlogged flows in an arbitrary subinterval $[t_1,\, t_2]$ of a busy period of the associated reference GPS system. Hence, $v(t)$ is a piecewise linear increasing function of time with a slope that changes whenever the set of backlogged flows $B(t_1,\, t_2)$ changes.

At each new packet arrival at WFQ, we need to make sure that the virtual time (which we represent by a C++ variable called *roundNumber*) is updated. Of course, each time we update the variable *roundNumber*, we need to update the *lastRUpdate-Time* variable, which indicates the last time when the *roundNumber* was updated. These two variables are used in our C++ code to implement (7.1) as follows,

$$roundNumber = roundNumber + \frac{(currentTime - lastRUpdateTime)}{weightSum} \cdot capacity\,;$$

All we need to create the H-F^{2}Q policy is stop updating the *lastRUpdateTime* variable in only one scenario: When the *roundNumber* variable is not updated at a new packet arrival *because* none of the flows in the reference GPS system are backlogged. For WFQ, you have to update the *lastRUpdateTime* at every packet arrival irrespective of the queueing system status. For the interested reader, the following two subsections explain this idea further.

7.2.1.1 WFQ Operations

In WFQ, calculating the virtual time during a *busy period* is done using (7.1), where the break points t_1 and t_2 are the instants of new arrivals and/or departures as seen by the reference GPS system. Notice that the *arrivals* at the GPS system are the same as those at the WFQ system. On the other hand, *departure* instants might be different for both systems.

Let us say that at time τ_e a busy period ends. At this time you should reset $v(t)$ to zero or stop updating $v(t)$, which means $v(t)$ would remain at $v\,(\tau_e)$[1]. When a new

[1]Busy periods always refer to the reference GPS system because this is where $v(t)$ is calculated

busy period starts again at τ_s by a new arrival (say at flow 1), the first packet arrival receives a time stamp based on $v(\tau_s) = v(\tau_e)$. For the next (second) arrival, $v(t)$ would be correspondingly updated to (given that the first packet did not depart the GPS system just yet),

$$v(\tau_2) = v(\tau_s) + \frac{C}{r_1} \cdot (\tau_2 - \tau_s) \tag{7.2}$$

7.2.1.2 H-F^{2}Q Operations

Calculating the virtual time in H-F^{2}Q is exactly the same as in WFQ until the end of a busy period. At the end of a busy period τ_e, we maintain the value of the virtual time $v(\tau_e)$, but *do not* update the variable *lastRUpdateTime*. This means that at a new busy period, the first arrival at τ_s receives a time stamp based on $v(\tau_s) = v(\tau_e)$.

However, for the next (second) arrival, $v(t)$ would be updated to be,

$$v(\tau_2) = v(\tau_s) + \frac{C}{r_1} \cdot (\tau_2 - \tau_e) \tag{7.3}$$

which is different from (7.2). This means that the second arrival gets a much bigger timestamp than it is supposed to (i.e., is delayed more).

Of course, if the first packet already left the GPS system, then τ_2 would be the time of GPS departure of that first packet. However, things get a bit cumbersome because this means that we reached the end of another busy period.

7.2.2 Properties of H-F^{2}Q

Preliminary investigation of the H-F^{2}Q queueing system indicates that if the utilization of such system is *smaller* than unity (i.e., $\rho \leq 1$), the queueing system provides the same mean packet delay for *all* supported flows (similar to the behavior of first-in first-out FIFO queueing). However, if the utilization grows beyond unity (i.e., when the system encounters a congestion period), the system reverts back to operate as a WFQ system, providing protection against misbehaving flows. In such mode, the mean packet delay of misbehaving flows (flows that send above their capacity reservations) grows dramatically while the delay of other behaving flows is kept finite, in the same way WFQ operates.

Although we do not go into details of proving the above statement (this is left for future work), this behavior has two important implications: First, H-F^{2}Q is a new and very interesting queueing system that nobody has thought of before. It will probably find applications not only in computer networking and processor sharing algorithms, but also in many other applied science disciplines that utilize queueing theory.

Second, and more importantly, since the modifications that convert WFQ to H-F^{2}Q are minor, a careful analysis of H-F^{2}Q might be a key to finding an expression for the mean waiting time of the WFQ algorithm (and hence an approximate solution to SCFQ and SFQ as we explained earlier in the concluding remarks in Section 6.4).

This might be done by arguing that,

$$W_{k,WFQ} = W_{k,H-F^2Q} + \Delta_k = W_{FIFO} + \Delta_k, \quad \rho \leq 1 \tag{7.4}$$

where Δ_k is a result of the minor modification required to convert WFQ into H-F^{2}Q, and W_{FIFO} is the mean waiting time under FIFO queueing (which is equal for all supported flows $k \in \mathbf{K}$).

Bibliography

[1] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: An Overview," IETF RFC 1633, June 1994.

[2] R. Braden *et al.*, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specifications," IETF RFC 2205, September 1997.

[3] S. Blake *et al.*, "An architecture for Differentiated Services," IETF RFC 2475, December 1998.

[4] Luca Martini *et al.*, "Transport of Layer 2 Frames Over MPLS," IETF Network Working Group, draft-martini-l2circuit-trans-mpls-10.txt, August 2002.

[5] M. Gagnaire, "An Overview of Broadband Access Technologies," Proceedings of the IEEE, Vol. 85, No. 12, pp. 1958–1972, December 1997.

[6] Y. Lin, W. Yin, C. Huang, "An Investigation into HFC MAC Protocols: Mechanisms, Implementation, and Research Issues," IEEE Communications Surveys, Third Quarter 2000, http://www.comsoc.org/pubs/surveys/.

[7] D. Fellows, D. Jones, "DOCSIS^{TM} Cable Modem Technology," IEEE Communications Magazine, Vol. 39, No. 3, pp. 202–209, March 2001.

[8] Data-Over-Cable Service Interface Specifications, Radio Frequency Interface Specification, SP-RFIv1.1-I07-010829, http://www.cablemodem.com/.

[9] C. Shirali, M. Shahar, K. Doucet, "High-bandwidth Interface for Multimedia Communications over Fixed Wireless Systems," IEEE Multimedia, Vol. 8, No. 3, pp. 87–95, 2001.

[10] IEEE 802.16 Standard, IEEE Draft Standard for Local and Metropolitan Area Networks – Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE Draft P802.16/D5, http://grouper.ieee.org/groups/802/16/published.html.

[11] A. K. Parekh, R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," INFOCOM'92, pp. 915–924, May 1992.

[12] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," Journal of Internetworking Research and Experience, pp. 3–26, October 1990. Also in ACM SIGCOMM'89, pp. 3–12.

[13] S. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," INFOCOM'94, pp. 636–646, April 1994.

[14] P. Goyal, H. M. Vin, H. Chen, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," SIGCOMM'96, pp.157–169, September 1996.

[15] D. Stiliadis, A. Varma, "Efficient Fair Queueing Algorithms for Packet-Switched Networks," IEEE/ACM Transactions on Networking, Vol. 6, No. 2, pp. 175–185, April 1998.

[16] E. L. Hahne, R. G. Gallager, "Round Robin Scheduling for Fair Flow Control in Data Communication Networks," International Conference on Communications, pp. 103–107, June 1986.

[17] S. J. Golestani, "Network Delay Analysis of a Class of Fair Queueing Algorithms," IEEE Journal on Selected Areas in Communications, Vol. 13, No. 6, pp. 1057–1070, August 1995.

[18] P. Goyal, S. S. Lam, H. M. Vin, "Determining End-to-End Delay Bounds in Heterogeneous Networks," Proc. Of 5th International Workshop on Network and Operating System Support for Digital Audio and Video, pp. 287–298, April 1995.

[19] D. Stiliadis, A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," INFOCOM '96, pp. 111–119, March 1996.

[20] F. M. Chiussi, A. Francini, "Implementing Fair Queueing in ATM switches – Part 1: A Practical Methodology for the Analysis of Delay Bounds," GLOBECOM 97, Vol. 1, pp. 509–518, November 1997.

[21] N. Pekergin, "Stochastic Bounds on Delays of Fair Queueing Algorithms," INFOCOM'99, pp. 1212–1219, 1999.

[22] N. Golmie, F. Mouveaux, D. Su, "Differentiated Services over Cable Networks," GLOBECOM'99, December 1999.

[23] R. Rabbat, K. Y. Siu, "QoS Support for Intergrated Services over CATV," IEEE Communications, Vol. 37, No. 1, pp. 64–68, January 1999.

[24] Xiaojun Xiao, W.K.G. Seah, Yong Huat Chew, Chi Chung Ko, "Upstream Resource Reservation and Scheduling Strategies for Hybrid Fiber/Coaxial Networks," APCC/OECC '99.

[25] M. Droubi, N. Idirene, C. Chen, "Dynamic Bandwidth Allocation for the HFC DOCSIS MAC Protocol," International Conference on Computer Communications and Networks, pp. 54–60, 2000.

[26] K. Sriram, "Performance of MAC Protocols for Broadband HFC and Wireless Access Networks," Advances in Performance Analysis, Vo1. 1, No. 1, pp. 1–37, 1998.

[27] N. Golmie, Y. Saintillan, D. Su, "A Review of Contention Resolution Algorithms for IEEE 802.14 Networks," IEEE Communications Surveys, First Quarter 1999, http://www.comsoc.org/pubs/surveys/.

[28] Y-D. Lin, C-Y. Huang, W-M. Yin, "Allocation and Scheduling Algorithms for IEEE 802.14 and MCNS in Hybrid Fiber Coaxial Networks," IEEE Transactions on Broadcasting, Vol. 44, No. 4, pp. 427–435, December 1998.

[29] W-M. Yin, Y-D. Lin, "Statistically Optimized Minislot Allocation for Initial and Collision Resolution in Hybrid Fiber Coaxial Networks," IEEE JSAC, Vol. 18, No. 4, September 2000.

[30] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, Vol. 1, No. 4, August 1993.

[31] Dimitri P. Bertsekas, Robert Gallager, "Data Networks," Prentice Hall, Second Edition, 1991.

Appendices

Appendix A

WE prove the equivalence of the two fairness bounds Ψ and $\psi_k(t)$ for packet-based FQ algorithms, and show that knowledge of one can lead to the other.

Assume that we have established that an arbitrary FQ system has the following fairness bound on the service lag of a flow k,

$$0 \leq \delta_k(t) \leq \psi_k(t), \qquad k \in \mathbf{K} \tag{A.1}$$

We show in this appendix that this fairness bound can be translated into another equivalent fairness bound in the form,

$$\left| \frac{W_k(t_1, t_2)}{r_k} - \frac{W_j(t_1, t_2)}{r_j} \right| \leq \Psi, \qquad j, k \in B(t_1, t_2) \tag{A.2}$$

We start by noticing that the fairness bound in (A.1) can be written more conveniently as follows,

$$0 \leq \delta_k(t) \leq \max_t \left[\psi_k(t)\right], \qquad k \in \mathbf{K} \tag{A.3}$$

This means that the fairness bound on the quantity $|\delta_k(t_2) - \delta_k(t_1)|$, $t_2 > t_1$, can be written as follows,

$$|\delta_k(t_2) - \delta_k(t_1)| \leq \max_t \left[\psi_k(t)\right], \qquad k \in \mathbf{K} \tag{A.4}$$

If we expand the service lag difference in (A.4), we get,

$$\delta_k(t_2) - \delta_k(t_1) = [v(t_2) - v_k(t_2)] - [v(t_1) - v_k(t_1)] = [v(t_2) - v(t_1)] - [v_k(t_2) - v_k(t_1)] \tag{A.5}$$

When flow k is backlogged during the entire interval $[t_1,\, t_2]$, we have,

$$v_k(t_2) - v_k(t_1) = \frac{W_k(t_2)}{r_k} - \frac{W_k(t_1)}{r_k} = \frac{W_k(t_1,t_2)}{r_k}, \qquad k \in B(t_1,t_2) \tag{A.6}$$

Substituting this into (A.5) and using (A.4), we get the following bound when flow k is backlogged,

$$\left| [v(t_2) - v(t_1)] - \frac{W_k(t_1,t_2)}{r_k} \right| \leq \max_t [\psi_k(t)], \qquad k \in B(t_1,t_2) \tag{A.7}$$

The same expression applies if we consider another flow $j \in B(t_1,t_2)$. Subtracting the expression (A.7) corresponding to flow k from that corresponding to flow j, we get,

$$\left| \frac{W_k(t_1,t_2)}{r_k} - \frac{W_j(t_1,t_2)}{r_j} \right| \leq \max_t [\psi_k(t)] + \max_t [\psi_j(t)], \qquad k,j \in B(t_1,t_2) \tag{A.8}$$

which has the same form as (A.2) with the parameter Ψ being,

$$\Psi = \max_t [\psi_k(t)] + \max_t [\psi_j(t)] \tag{A.9}$$

The reverse process can be easily proved in a similar way. To give the reader an example of the fairness bound, we notice that in SCFQ $\max_t [\psi_k(t)] = L_k^{\max}/r_k$, $\max_t [\psi_j(t)] = L_j^{\max}/r_j$ and $\Psi_{SCFQ} = L_k^{\max}/r_k + L_j^{\max}/r_j$.

Appendix B

WE show that the means of two *similar* probability distributions (e.g., the ones shown in Figures B.1(a) and (b)) are actually related.

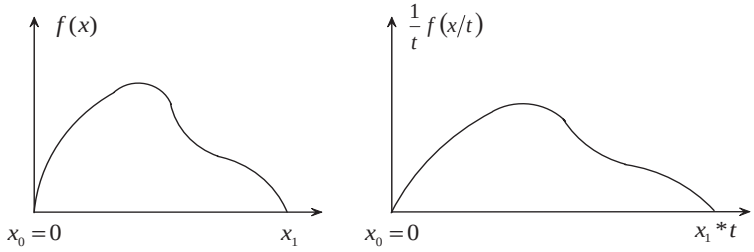

Figure B.1: Two similar probability distributions.

Assume that the probability distribution in Figure B.1(a) is $f(x)$ and its mean is given by,

$$m = \int_0^{x_1} x\, f(x)\, dx$$

The probability distribution in Figure B.1(b) is given by $(1/t)f(x/t)$, and its mean is,

$$\int_{x=0}^{x=t*x_1} (x/t) f(x/t)\, dx \;=\; \int_{u=0}^{u=t*x_1/t} t \cdot u\, f(u)\, du = t \cdot m$$

Both of which are related by t.

Vita

Mohammed Hawa was born in Dubai (UAE). He graduated from the University of Jordan (Jordan) in 1997 with a B.Sc. degree in Electrical Engineering, and received his M.Sc. degree in Telecommunications (with Distinction honor) from the University of London/UCL (UK) in 1999. Mr. Hawa then joined the Information and Telecommunications Technology Center (ITTC) at the University of Kansas as a Research Assistant. He also worked as a part-time Lecturer for a junior level "Electrical Circuits" class where he excelled in teaching. Mr. Hawa received his Ph.D. degree in Electrical Engineering form the University of Kansas (Kansas, USA) in 2003.

Mr. Hawa is the recipient of the Fulbright Scholarship (University of Kansas, 1999) and the Shell Centenary Scholarship (University of London/UCL, 1998). He is a published author and a member of the IEEE and IEEE Communications Society. He also served as a reviewer for the IEEE Communications Letters Journal and the ICC 2001 Conference.

During his Ph.D. study, Mr. Hawa researched Quality-of-Service in IP-based broadband wireless networks. His main research interests include networking, Quality-of-Service, queuing systems, multimedia applications and wireless networks.

Permanent address:
P.O. Box 961200
Amman, 11196,
JORDAN

This dissertation was typeset with LaTeX by the author.