Packet Scheduling Algorithms for Integrated Services Networks

by

Pawan Goyal, M.S., B.Tech.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 1997

Packet Scheduling Algorithms for Integrated Services Networks

Approved by
Dissertation Committee:

To my family

Acknowledgments

It is a tribute to the guidance, support, and company that I have received during my stay at UT Austin that five years of graduate studies, which could have been long and arduous, have gone by quickly. I will try and acknowledge the people to whom I am indebted. However, it is more likely than not that I may forget to acknowledge some people. My apologies to those that I may forget.

I would first like to thank my advisor Prof. Harrick Vin. He was the one who convinced me to get a doctoral degree; this dissertation is a result of his successful persuasion. I will always cherish his mentoring. He has also been a great friend. I have often turned to him for advice on several matters.

I thank Professors S. Keshav, Simon Lam, Al Mok, and Gustavo de Veciana for serving on my committee. I have enjoyed discussing my work with them and am grateful for their constructive comments. Special thanks are due to Prof. S. Keshav for bringing parts of this dissertation to the attention of several people.

I was fortunate in having an intellectually stimulating work environment at the Distributed Multimedia Computing Laboratory. Collaborations and interactions with Alok Goyal, Anshuman Goyal, Xingang Guo, Scott Page, Ed Posnak, Sriram Rao, Prashant Shenoy, and Renu Tewari have helped shape my research perspective. I will always remember the “lunch hour” with them which I used to always look forward to. I would like to thank S. Gooty and Neeraj Mittal for helping me in the implementation of the Hierarchical SFQ scheduling algorithm.

Life in Austin was enjoyable mainly because of my friends. I have enjoyed spending hours discussing topics ranging from philosophy to politics with Om Damani, Rajesh Govindan, Scott Page, Rajmohan Rajaraman, Rahul Singh, Prashant Shenoy, and Anal Shah. I would specifically like to thank Rajmohan Rajaraman and Prashant Shenoy for all the help that I received from them.

Finally, my family provided constant support and encouragement. My elder brother, Sunil, has been a great influence on me. His guidance was instrumental in shaping my academic career. I dedicate this dissertation to my family.

PAWAN GOYAL

The University of Texas at Austin
August 1997

Packet Scheduling Algorithms for Integrated Services Networks

Publication No. ____________________

Pawan Goyal, Ph.D.
The University of Texas at Austin, 1997

Supervisor: Harrick M. Vin

An integrated services network supports applications with diverse Quality of Service (QoS) requirements. Packet scheduling is a fundamental mechanism employed by such a network to meet the QoS requirements of the applications. In this dissertation, we design, analyze, and implement packet scheduling algorithms for integrated services networks.

We first define and analyze a class of Guaranteed Rate (GR) scheduling algorithms. We demonstrate that GR contains most of the known scheduling algorithms that can provide guaranteed QoS. The GR class enables an integrated services network to provide bounded packet delay and loss service to sources with different characteristics in heterogeneous networking environments where each switch may employ a different scheduling algorithm and packet fragmentation/reassembly may occur. The GR framework separates the end-to-end performance guarantees provided by a network from the traffic characteristics of a source and thereby leads to a simple network architecture. We demonstrate that though the GR class does not contain all the algorithms, it does not lead to loss in achievable utilization of a network. Finally, our analysis of GR class demonstrates that scheduling algorithms that are fair and/or achieve separation of rate and delay allocation are desirable.

We then examine the requirements for a suitable packet scheduling algorithm for integrated services network. Our analysis demonstrates that a suitable packet scheduling algorithm

should: (1) achieve low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provide fairness for variable bit rate video; (3) provide fairness over variable rate servers for throughput-intensive flow-controlled data applications; (4) provide QoS guarantees over variable rate servers; (5) support hierarchical link bandwidth allocation to enable a network to provide link-sharing service and facilitate coexistence of multiple services and protocols; and (6) be computationally efficient. To meet these requirements, we first design Start-time Fair Queuing (SFQ) and Fair Airport (FA) algorithms. We then employ them to design Hierarchical SFQ and FA algorithms that achieve hierarchical allocation of link bandwidth. Finally, we present an implementation of Hierarchical SFQ in an IPv6 router. Our analysis and experiments demonstrate that these algorithms meet the requirements and, thus, are suitable scheduling algorithms for integrated services networks.

The key contributions of this dissertation include: (1) a framework for tight analysis of a heterogeneous network; (2) illustration of the need for fairness over variable rate servers and design of algorithms that meet this requirement; (3) design of hierarchical scheduling algorithms that provide quantitative performance guarantees; (4) the single server and end-to-end analysis of SFQ and FA for variable rate Fluctuation Constrained and Exponentially Bounded Fluctuation servers; and (5) an analysis technique for hierarchical scheduling algorithms that yields tight performance guarantees.

Contents

Chapter 1

Introduction

This dissertation focuses on design, analysis, and implementation of packet scheduling algorithms for integrated services networks. In this chapter, we present the motivation for our research and a brief overview of the results of the dissertation.

1.1 Motivation

Packet-switched computer networks have advanced significantly since their inception. From small networks connecting few computers, the largest computer network, the Internet, has evolved into a network connecting more than a million computers [47]. Conventionally, these networks have provided a *best-effort service*. In best-effort service, a network does not guarantee whether and when a packet will be delivered. Although best-effort services provided by these networks have been adequate for applications such as telnet and ftp; several new applications are motivating fundamental changes in the services provided by computer networks.

- Multimedia applications: Over the past few years, advances in communication and computing technologies have yielded many-fold increase in the network bandwidth and processing power of computers. Concurrent advances in video compression technology have led to the development of computationally efficient algorithms for compressing digital video by two orders of magnitude or more. These synergistic advances have made the bandwidth as well as storage space requirements of digital video manageable, and have thereby enabled a large class of multimedia applications including video conferencing, distance learning, news-on-demand services, and virtual reality.

To determine the services required from a computer network by such multimedia applications, consider a typical multimedia application, video conferencing. In video conferencing, the audio and video signals from a participant (source) are digitized, possibly compressed, packetized, and transmitted over a network to the other participants (receivers). On receipt of packets, a receiver decompresses them, if necessary, and plays them back such that the reconstructed audio/video signal is as close to the original as possible. To enable a receiver to achieve this objective, the network must guarantee:

- Minimum bandwidth: Since audio and video signals have an intrinsic data generation rate, to enable the receiver to reconstruct the signal with high fidelity, the network must guarantee a minimum bandwidth which is close to the data generation rate.
- Bound on delay of packets: Due to non-zero delay incurred by packets, the reconstructed signal is a time-shifted version of the original. Let the time-shift of the signal be D. Then, to reconstruct the signal at time $t + D$, the packet containing the signal value at time t should arrive at the receiver no later than $t + D$, i.e., the delay incurred by the packet should be at most D. Otherwise, the signal value at time $t + D$ will not be reconstructed, thereby resulting in loss in fidelity of the reconstructed signal. To avoid such fidelity loss, the network should guarantee an upper bound on packet delay. Furthermore, since the maximum "time-shift" that can be tolerated depends on the interactivity requirements, an application should be able to specify a bound on the delay of packets.

The current networks provide neither a minimum bandwidth guarantee nor a bound on delay of packets and, hence, are inadequate for multimedia applications.

- Real-time applications: Computer networks are increasingly being employed for real-time applications such as control of manufacturing plants and stock-trading. These applications require the network to guarantee bounds on packet delay for their correct functioning. Since general purpose networks do not provide these bounds, special networks are employed for them. To facilitate their wider deployment, it is desirable that general purpose networks provide real-time services.

The need for supporting multimedia and real-time applications along with conventional data applications in computer networks, has led to the design of *integrated services networks*. An integrated services network will support a variety of applications with a wide range of Quality of Service (QoS) requirements. It will provide QoS guarantees with respect to minimum bandwidth and upper bound on packet delay and loss to real-time and continuous media applications (such as audio and video conferencing) as well support best effort services provided by current networks. The design of a network that provides such a flexible QoS is challenging and its realization requires innovations at various layers of a network architecture. Since the network layer of OSI architecture defines the services exported by a network against which the applications are programmed, the most significant and long lasting changes are required at the network layer [47].

A fundamental problem at the network layer is to meet the QoS requirements of packet sources, i.e., avoid *congestion* [32]. To control packet delay and loss, and thus avoid congestion, a network has to manage two resources: link bandwidth and buffers at each packet-switch. In a packet-switched network, resources are not physically partitioned among various sources. Hence, contention for access to link and buffers may occur. Consequently, a network has to employ algorithms for arbitrating access and limiting contention for these resources. Specifically:

- *Algorithms for arbitrating access*: We refer to algorithms that arbitrate access to link bandwidth and buffer as *packet scheduling* and *buffer scheduling* algorithms, respectively. A packet scheduling algorithm determines the order of packet transmission at a link and consequently controls the bandwidth and packet delays and loss for a source. A buffer scheduling algorithm determines the packet to be dropped when buffer overflow occurs and, hence, controls the packet loss for a source. Thus, by discriminating between sources, packet and buffer scheduling algorithms enable a network to provide different QoS to different sources.

- *Techniques for limiting contention*: The technique for limiting contention depends on the QoS requirements of the sources. Consider sources requiring guaranteed QoS and best-effort service.

  - *Guaranteed QoS*: In this case, sources require a network to guarantee QoS parameters such as minimum bandwidth and upper bound on packet delay, delay jitter, and loss. The bounds on delay and loss may be deterministic or probabilistic [13].

    To meet the QoS requirements of such sources, a network employs an open-loop control technique. In this technique, a source specifies its QoS requirements and traffic characteristics such as average rate, peak rate, and burst size to the network [13]. The network, in turn, employs an admission control algorithm to determine if the source can be admitted. The admission control algorithm uses the characteristics of the packet scheduling algorithm at the switches and the set of sources that have already been admitted to determine if the QoS requirement of the new source can be met. If the requirements can be met, the source is admitted and resources are reserved for it. A packet scheduling algorithm is employed to ensure that a source receives service as per the specification.

  - *Best-effort QoS*: In this case, a network does not reserve resources and does not guarantee whether or when a packet will be delivered. Congestion is avoided by employing a closed-loop control technique. Packet sources estimate the network state such as available buffers and link bandwidth. On detection of congestion, either implicitly or through explicit notification from the network, they control their behaviour, using either a window or rate based flow control algorithm, to avoid congestion [32]. This control technique achieves efficient utilization of network resources.

A network may also employ a combination of open and closed loop control techniques for sources that desire a minimum guaranteed QoS for a subset of the packets and best-effort service for the remaining packets. Since an integrated services network will support applications that require guaranteed QoS as well as best-effort services, open-loop and closed-loop control techniques will coexist.

The open-loop and closed-loop control techniques are closely intertwined with, and dependent on, packet and buffer scheduling algorithms. In case of open-loop control, the characteristics of the packet scheduling algorithm determine the set of sources that can be admitted and, hence, the achievable utilization of a network. Furthermore, it ensures that a source receives service as per its reservation. In case of closed-loop control, scheduling algorithms enable: (1) fair allocation of network resources; (2) network state estimation; and (3) coexistence of different feedback control algorithms [32, 45]. Thus, packet and buffer scheduling algorithms enable a network to not only deliver different QoS to different sources, but also meet the QoS requirements of individual sources. Hence, packet and buffer scheduling algorithms are central to the realization of integrated services networks.

Though both packet and buffer scheduling algorithms are important, packet scheduling algorithms play a more fundamental role in meeting QoS requirements. This is because whereas a buffer scheduling algorithm controls the packet loss for a source, a packet scheduling algorithm controls the packet delay, loss, and the bandwidth of a source. Hence, we confine our focus to packet scheduling algorithms.

The design and analysis of packet scheduling algorithms for integrated services networks poses several challenges. Since the traffic characteristics of sources in integrated services networks vary significantly, a scheduling algorithm should be able to provide QoS guarantees to sources with different characteristics. It should also meet the QoS requirement of best-effort sources. Furthermore, since an integrated services network will employ multiple control techniques, it should enable their coexistence. Finally, since future networks will be high speed in nature, it should be computationally efficient. The objective of this dissertation is to meet these challenges.

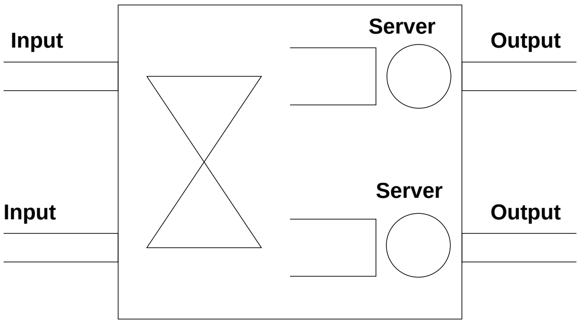


Figure 1.1: An output buffered packet-switch

The rest of this chapter is structured as follows. We present related work in Section 1.2 and an overview of the results of this dissertation in Section 1.3.

1.2 Related Work

Packet sources and sinks are processes at the session layer of the OSI network architecture. The sequence of packets transmitted by a source to a sink is referred to as a *flow* [62]. We use the term flow synonymously with a connection. A flow is serviced by a fixed sequence of packet-switches (packet-switch is used synonymously with a router). For simplicity of analysis, packet-switches are assumed to be output buffered. In an output buffered switch, packets arrive at an input link and are switched to an output link where they are queued until further transmission (see Figure 1.1). We refer to the output link of a switch as a *server*. We assume that *per-flow* state is available at each packet switch. This assumption holds for networks with connection oriented network layer such as Asynchronous Transfer Mode (ATM). It is also applicable in networks with connectionless network layer such as Internet enhanced with resource reservation protocols such as RSVP [63].

To meet the QoS requirements of applications in an integrated services network, several packet scheduling algorithms have been designed [14, 21, 22, 30, 36, 42, 59, 61, 62, 64]. Since a detailed survey of these algorithms has been presented in [58], we review the algorithms only briefly.

The design of a packet scheduling algorithm depends on the desired QoS. Hence, to understand the design of the scheduling algorithms, consider *guaranteed* and *best-effort* QoS.

- *Guaranteed QoS*: In this case, the objective of a packet scheduling algorithm is to guarantee a reserved rate and upper bounds on packet delay, delay jitter, and loss to a flow.

  The simplest approach for achieving this objective, when all the flows have equal length packets, is to service flows in a round robbin manner and transmit equal number of packets of each flow. This approach, however, allocates the same rate to all the flows. This limitation can be addressed by a weighted round robbin scheduling algorithm that: (1) assigns weights to the flows in proportion to their reserved rate; and (2) services flows in round robbin manner transmitting number of packets equal to the weight of a flow in each round. However, this approach is not feasible when flows may have different packet length and a flow may have variable length packets [42, 50]. This limitation has been addressed by Deficit Round Robin scheduling algorithm presented in [50]. The principal advantage of these round based scheduling algorithms is that they are simple to implement. However, they can guarantee only a very large upper bound on packet delay. For example, if a packet of a flow arrives just after the time the flow was scheduled to be serviced in a round, the packet may be delayed by a full round duration. Furthermore, since the maximum delay is governed by the round duration, which is same for all the flows, they can not provide different delay bounds to different flows. These limitations have been addressed by dynamic priority scheduling algorithms.

  The dynamic priority algorithms assign a priority to each packet and schedule packets in increasing order of priority. If the reserved rate for flow f is r_f, they assign a priority to the packets of flow f such that the departure time of a flow f packet is at most a constant more than the departure time in a server of rate r_f [23, 24]. One possible priority assignment for the packets of a flow is their departure time in a server of rate r_f. This priority assignment results in the Virtual Clock scheduling algorithm [62]. To illustrate the priority assignment in Virtual Clock, consider a server of capacity $6pkts/s$ that is serving flows 1 and 2 with reserved rate $1pkt/s$. Let flows flows 1 and 2 send 10 packets at time 0 and 1, respectively. Then, the departure time of j^{th} packet of flow 1 ($j \leq 10$)

in a server of rate $1pkt/s$, and consequently its priority, is j. Similarly, the priority of j^{th} packet of flow 2 ($j \leq 10$) is $1 + j$. It has been shown that Virtual Clock guarantees that departure time of a packet is at most a constant more than the departure time in a server of rate r_f [55]. There are two main limitations of the Virtual Clock scheduling algorithm:

– *Unfairness*: Consider the previous example. Since the server capacity is $6pkts/s$, the first 6 packets of flow 1 will be serviced by time 1. Now, consider the time interval $[1, 2]$. Since the priority of first 6 packets of flow 2 is at most the priority of the 7^{th} packet of flow 1, only flow 2 will be serviced in $[1, 2]$. Consequently, flow 1 will be *unfairly* penalized in the interval $[1, 2]$ for using idle server resources in the interval $[0, 1]$. To address this limitation, *fair* algorithms have been designed [4, 12, 42, 22, 52]. Fair algorithms ensure that in any time interval in which two flows are backlogged, they receive service in proportion to their reserved rates. Hence, in the example scenario a fair algorithm will serve equal number of packets of flows 1 and 2 in the interval $[1, 2]$. Several fair scheduling algorithms such as Weighted Fair Queuing (WFQ) [12] (also known as Packet-by-packet Generalized Processor Sharing [42]), Self Clocked Fair Queuing (SCFQ) [22], and Frame-based Fair Queuing (FFQ) [52] have been designed in the literature. These algorithms essentially emulate a hypothetical bit-by-bit round robin server in which the number of bits of a backlogged flow served in a round is proportional to the rate of the flow.

– *Coupling of rate and delay allocation*: Virtual Clock allocates only rate for a flow. Consequently, the departure time of a flow's packet, and hence its delay, is coupled to the reserved rate of the flow. In the previous example, the bound on the departure time and, hence, delay of the first packet of flow 1 is at least $1s$. If the flow desires delay of $0.5s$, its rate reservation would have to be increased to $2pkt/s$. This inverse relationship between delay and rate reservation may lead to inefficient utilization of resources [61]. Thus, scheduling algorithms that achieve separation of rate and delay allocation have been designed. An example of such an algorithm is Delay EDD [13].

The algorithms that we have considered so far service a packet whenever a packet is backlogged at the server, i.e., they are are work conserving. Non-work conserving algorithms that keep a server idle even when packets are backlogged at the server also have been proposed in the literature. Some non-work conserving scheduling algorithms serve a packet from a flow only at predetermined time instants. Examples of such algorithms include Hierarchical Round Robin and Stop-and-go Queuing [21, 61]. Other non-work conserving algorithms service a packet only after it becomes *eligible* [60]. Several eligibility notions have been defined [60]. One of the simplest notions is to consider a packet eligible at the time at which it would have arrived if the flow was transmitting exactly at the reserved rate. To illustrate, consider a flow that has reserved rate $1pkt/s$ and sends 5 packets at time 0. Then, the first packet will be eligible at time 0, second at time $1s$, third at time $2s$, and so on. This is the concept of eligibility employed by Rate Controlled Static Priority Queuing (RCSP) [60].

The non-work conserving algorithms may either allocate only rate or achieve separation of rate and delay allocation. Examples of algorithms that allocate only rate include Hierarchical Round Robin and Stop-and-go Queuing. Examples of algorithms that achieve separation of rate and delay allocation include RCSP and Jitter EDD [61]. The concept of fairness is not applicable for non-work conserving algorithms.

The principal advantage of the non-work conserving algorithms is that they maintain the smoothness of a flow and thus reduce the buffer requirement for a flow in the network. However, by keeping a server idle when packets are backlogged they increase the average delay of the flows. On the other hand, work conserving algorithms yield better average delay performance at the expense of increased buffer requirement in the network. Further discussion of the tradeoffs between work conserving and non-work conserving algorithms can be found in [34].

- *Best-effort QoS*: In this case, one of the principal objectives of a packet scheduling algorithm is to allocate link bandwidth fairly [12]. The fair algorithms designed to provide guaranteed service can achieve this objective. However, since resources are not reserved

	Rate Allocation	Delay Allocation
Work Conserving	Virtual Clock WFQ SCFQ	Delay EDD
Non-work Conserving	Hierarchical Round Robin Stop-and-go Queuing	Jitter EDD RCSP

Table 1.1: Classification of scheduling algorithms

in best-effort service, fair scheduling algorithms allocate server bandwidth in proportion to weights, rather than reserved rates, of the flows [12].

Thus, most of the scheduling algorithms that have been proposed can be classified as either being work conserving or non-work conserving and either allocating only rate or separating rate and delay allocation. The work conserving algorithms can further be classified as either being fair or unfair. Table 1.1 summarizes this classification.

To provide QoS guarantees to sources, several source traffic specifications also have been proposed. Examples include leaky bucket [9, 42], Flow Specification [55], average and peak rate specification [14], Deterministic Bounding Interval Dependent model [35], and Exponentially Bounded Burstiness [56]. The QoS guarantees of a single server for different source traffic specifications and scheduling algorithms have been analyzed in the literature [9, 42]. The QoS guarantees provided by a single server have been extended to a network of servers [9, 10, 36, 42, 57]. A survey of single server and multiple server analysis techniques can be found in [58].

There are two main limitations of the existing work on packet scheduling algorithms:

- The scheduling algorithms have been analyzed only for a restricted set of source traffic characterizations. Furthermore, most of them have been analyzed assuming that all the flows conform to their declared specifications. Finally, the analyses of a network of servers assumes that the same scheduling algorithm is employed at each of the servers [9, 10, 36, 42, 57]. In an integrated services network, sources have widely different characteristics and may not conform to their advertised specifications. Furthermore, since the network may be part of several autonomous administrative domains, different scheduling algorithms may be employed at different servers. Hence, a framework for analysis of packet scheduling algorithms that does not have these limitations and enables a network to provide QoS guarantees in heterogeneous environments is desirable.

- Most of the scheduling algorithms do not meet the requirements of the wide variety of applications supported by an integrated services network. Furthermore, they do not enable the coexistence of different control techniques that will be employed by an integrated services network (Chapter 4 evaluates the existing algorithms in detail).

This dissertation addresses these limitations.

1.3 Summary of Contributions

In this dissertation, we first develop a framework for the analysis of packet scheduling algorithms in heterogeneous networks and employ it to determine the desirable characteristics of packet scheduling algorithms. We then determine the requirements for suitable packet scheduling algorithms for integrated services networks and develop new algorithms to meet those requirements. Through analysis and experiments we demonstrate that these new scheduling algorithms are suitable for integrated services networks. In what follows, we present our contributions in more detail.

We first define and analyze a class of Guaranteed Rate (GR) scheduling algorithms. The scheduling algorithms in GR provide a service guarantee to a flow that is independent of a traffic specification and the behaviour of the other flows in a network. This enables a GR server to provide QoS guarantees to flows conforming to any specification. We demonstrate that the class of GR scheduling algorithms is broad and includes most of the algorithms proposed in the literature. To efficiently support flows with variable bit rate requirements, we define several algorithms that allocate variable rates to the packets of a flow and demonstrate that they belong to the GR class. We also define scheduling algorithms in GR that are suitable for servers where packet fragmentation occurs.

To enable a network to provide QoS guarantees similar to those provided by a single server, we develop a method for deriving the service guarantee of a network of GR servers. This method enables the derivation of service guarantee of a heterogeneous network in which packet fragmentation and/or reassembly may occur. We demonstrate that the end-to-end service guarantee is similar to a single server guarantee. This reduces the end-to-end QoS determination problem to a single server problem and enables a network to provide guarantees to flows with any traffic specification. We illustrate the end-to-end delay bound computation for flows conforming to leaky bucket and demonstrate that our method leads to tighter results. The key contributions of the GR framework are: (1) it decouples service guarantee provided by a network from source traffic characterization, and (2) it is the first framework that tightly analyzes heterogeneous networks. This framework is now being employed by other researchers to analyze new scheduling algorithms [53].

To gain a further understanding of the GR class, we evaluate the completeness and relative merits of algorithms in GR.

- *Completeness of GR*: Though GR does not contain all the algorithms, we experimentally demonstrate that it does not lead to loss in achievable utilization of a network. In fact, contrary to intuition, we demonstrate that algorithms that are not in GR lead to lower achievable utilization.

- *Relative merits of the algorithms in GR*: We evaluate the merits of algorithms classified along two dimensions:

  - *Fair vs. unfair*: To evaluate fair and unfair algorithms, we introduce the concept of delay-cum-throughput guarantee which refers to a deadline guarantee for a packet based on the departure time of the previous packet of the flow. We derive end-to-end delay-cum-throughput guarantee of an internetwork of servers that provide such a guarantee. We demonstrate that fair algorithms, in addition to the guarantees provided by unfair algorithms, provide this guarantee. This enables fair algorithms to provide better bounds on QoS to a flow when a network is uncongested. Hence, fair algorithms are desirable.

– *Rate-only vs. rate-and-delay allocation algorithms*: We experimentally demonstrate that algorithms that separate rate and delay allocation achieve higher utilization and, hence, are desirable.

Thus, we conclude that GR is a useful class of algorithms. Furthermore, algorithms that are fair and/or achieve separation of rate and delay allocation are desirable.

We then examine the requirements for a suitable packet scheduling algorithm for integrated services network. Our analysis demonstrates that a suitable packet scheduling algorithm should: (1) achieve low average and maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provide fairness for variable bit rate video; (3) provide fairness over variable rate servers for throughput-intensive flow-controlled data applications; (4) provide QoS guarantees over variable rate servers; (5) support hierarchical link bandwidth allocation to enable a network to provide link-sharing service and facilitate coexistence of multiple services and protocols; and (6) be computationally efficient. To meet these requirements, we design Start-time Fair Queuing (SFQ) and a class of Fair Airport (FA) algorithms, and employ them to develop Hierarchical SFQ and FA algorithms for hierarchically allocating link bandwidth. Specifically:

- *SFQ*: We demonstrate that SFQ is computationally efficient and allocates bandwidth fairly regardless of variation in a server rate. We show that its fairness guarantee is close to the optimal and is at least as good as that of all the known algorithms. We analyze the throughput, single server deadline, and end-to-end deadline guarantee of SFQ. This analysis is carried out for variable rate servers that can be modeled as either Fluctuation Constrained (FC) or Exponentially Bounded Fluctuation (EBF) servers.

- *FA algorithms*: We design FA algorithms by combining SFQ with a class of non-work conserving algorithms. Though FA algorithms have higher implementation complexity than SFQ, we demonstrate that they are computationally efficient. We derive fairness and deadline guarantees for FC FA servers. We demonstrate that appropriate choice of a non-work conserving algorithm leads to the design of first fair algorithm that achieves separation of rate and delay allocation. We also show that FA class contains algorithms with different implementation complexities and performance guarantees and, thus, enables tradeoffs between the two. Finally, we demonstrate that it can be generalized to achieve several other objectives.

SFQ and FA belong to the GR class and hence have all the desirable properties of the GR algorithms. Our analysis and experiments demonstrate that SFQ and FA algorithms meet all the requirements for a suitable packet scheduling algorithm.

To allocate link bandwidth hierarchically, we employ SFQ and FA and develop *Hierarchical SFQ* and *FA* scheduling algorithms. Hierarchical SFQ and FA enable a network to provide a link-sharing service and manage heterogeneity in services and protocols [47]. These algorithms allow different scheduling algorithms to be employed for the leaf classes of a hierarchy. We develop a technique for analyzing the performance guarantees of Hierarchical SFQ and FA. Using this technique the QoS guarantees for a flow can be determined if the performance guarantees of a leaf class scheduling algorithm are known for FC/EBF servers. We derive the performance guarantees for a flow when the leaf class scheduling algorithm is either SFQ, FA, Virtual Clock, or Delay EDD. Our analysis technique is general and applicable for other hierarchical schedulers constructed using algorithms that provide guarantees similar to SFQ and FA. Furthermore, this analysis technique is the first in the literature that yields tight performance bounds for hierarchical scheduling algorithms. Finally, we present an implementation of Hierarchical SFQ in an IPv6 router.

The key contributions of our work on SFQ and FA include: (1) illustration of the need for fairness over variable rate servers and design of algorithms that meet this requirement; (2) design of hierarchical scheduling algorithms that provide quantitative performance guarantees; (3) the single server and end-to-end analysis of SFQ and FA for variable rate FC and EBF servers; and (4) analysis technique for hierarchical scheduling algorithms that yields tight performance guarantees.

The rest of the dissertation is organized as follows. We define and analyze the class of Guaranteed Rate scheduling algorithms in Chapter 2. The completeness of GR framework and relative merits of algorithms in GR are evaluated in Chapter 3. We determine the requirements for suitable packet scheduling algorithms and demonstrate that existing algorithms do not satisfy them in Chapter 4. SFQ and FA algorithms and their analysis are presented in Chapters 5 and 6, respectively. Hierarchical SFQ and FA are presented in Chapter 7. We evaluate the SFQ and FA algorithms in Chapter 8. Finally, Chapter 9 presents the conclusions of this dissertation and directions for future research.

Chapter 2

Guaranteed Rate Scheduling Algorithms: A Framework

The traffic characteristics of packet sources in integrated services networks differ significantly. Hence, a scheduling algorithm employed by a server must be able to provide QoS guarantees to sources with different characteristics. For example, it should be able to provide QoS guarantees to audio sources, many of which have constant bit rate requirements, as well as Variable Bit Rate (VBR) video sources which have multiple time-scale variations in the bit rate (Figure 2.1 shows short-term as well as the long-term variations in the bit rate variation of a MPEG compressed video sequence [16]). Furthermore, since sources may not conform to the advertised specification, the QoS guarantees provided to a source should be independent of the behaviour of the other flows.

A network of servers also should provide QoS guarantees similar to a single server. Furthermore, since current networks are, and future networks will remain, heterogeneous along several dimensions, the network should provide QoS guarantees in presence of heterogeneity. For example, it should provide QoS guarantees in a large network consisting of several autonomous domains in which servers may employ different scheduling algorithms (e.g., work conserving, non-work conserving, ones that separate delay and rate allocation, and the ones that only allocate rate). Similarly, it should provide QoS guarantees in internetworks where, due to

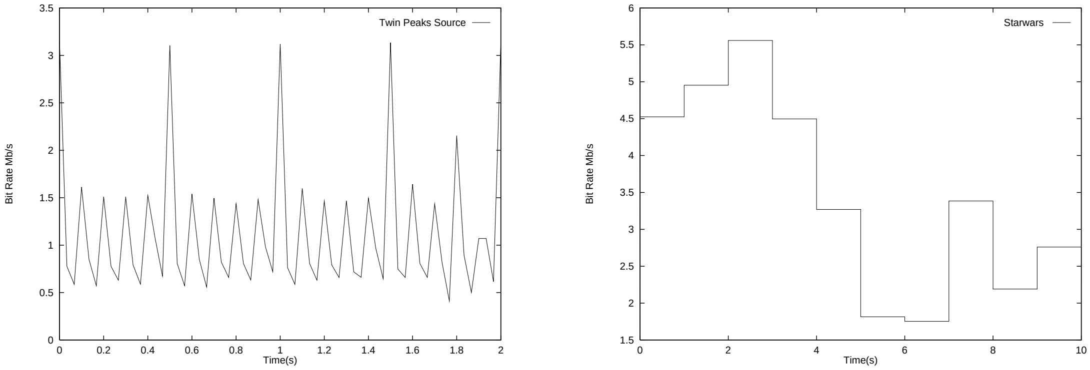


Figure 2.1: Short-term and long-term bit rate variations in a MPEG compressed video sequence

the variation in the size of data transmission unit (e.g., in an internetwork consisting of ATM, FDDI, ethernet, and token ring), packet fragmentation and/or reassembly may occur.

To achieve these objectives, in this chapter we define a class of Guaranteed Rate (GR) scheduling algorithms that provide QoS guarantees to flows with different traffic characteristics, and develop a general method for providing QoS guarantees in a network of GR servers. We present an overview of the results and organization of this chapter in the next section.

2.1 Overview of Results

The class of GR scheduling algorithms guarantees a deadline (referred to as *delay guarantee*) to a packet based on its expected arrival time (the delay guarantee term was introduced in [55]). The delay guarantee of these algorithms is independent of a traffic specification and the behaviour of other flows at the server. This enables a single server employing a scheduling algorithm in GR to isolate flows as well as provide service guarantees to flows conforming to any specification. We demonstrate that the class of GR scheduling algorithms is broad, and includes work conserving and non-work conserving scheduling algorithms as well as algorithms that allocate only rate and those that separate rate and delay allocation. Examples of algorithms in GR are Virtual Clock, Weighted Fair Queuing (WFQ), Self Clocked Fair Queuing (SCFQ), Delay Earliest Due Date (EDD), and Rate Controlled Static Priority Queuing. We define work conserving generalized Virtual Clock, WFQ, and SCFQ scheduling algorithms that can allocate variable rate to the packets of a flow, and show that they also belong to GR. To prove that many scheduling algorithms belong to GR, we employ a proof methodology in which we first show that a preemptive equivalent of the algorithm belongs to GR, and then utilize a relationship between preemptive and non-preemptive scheduling algorithms to show that the non-preemptive algorithm belongs to GR. This methodology leads to the definition of several scheduling algorithms in GR that are suitable for servers at which packet fragmentation may occur. The algorithms that we define for such servers reduce computational complexity as well as delay incurred by packets. Finally, we demonstrate that if a rate control element is employed in conjunction with any scheduling algorithm in GR on a *per flow* basis (this leads to the definition of several scheduling algorithms), the resulting non-work conserving algorithm also belongs to GR and enables a network to support flows with different delay-jitter requirements.

The delay guarantee of the GR class enables a single server to provide service guarantees to flows conforming to any specification. To enable a sequence of servers to provide similar service guarantees, we develop a method for deriving the delay guarantee of a network of servers each of which employs a scheduling algorithm in the GR class. This method enables the derivation of delay guarantee for a network of servers even when: (1) different rates are allocated to the packets of a flow at different servers along the path and the bottleneck server for each packet may be different, and (2) packet fragmentation and/or reassembly may occur in the network. We utilize the delay guarantee of a network of servers to obtain an upper bound on end-to-end delay. We demonstrate that our method of deriving delay guarantee for a network of servers reduces the problem of determining end-to-end delay bound to a single server problem. We illustrate the end-to-end delay bound computation for flows conforming to leaky bucket. We demonstrate that our method for determining these bounds leads to tighter results (e.g., it improves upon the delay bound presented in [41] for flows conforming to leaky bucket in WFQ networks).

Finally, based on the properties of GR class, we present architectural principles for designing networks that provide guaranteed deterministic QoS. We demonstrate that GR class simplifies the design of a network while enabling it to support applications with different characteristics and requirements

The rest of the chapter is organized as follows: In Section 2.2, we define the class of GR scheduling algorithms and demonstrate that several algorithms belong to it. The method for deriving delay guarantee for a network of servers is presented in Section 2.3. We utilize the delay guarantee to derive end-to-end delay bounds in Section 2.4 and then present the architectural principles for designing GR networks in Section 2.5. Finally, Section 2.6 summarizes the results of this chapter.

2.2 Guaranteed Rate Scheduling Algorithms

Packet scheduling algorithms provide guarantees to a flow based on its reserved rate. For simplicity of exposition, let r_f be the fixed rate reserved for flow f. Let the flow be serviced by a server that has capacity greater than r_f and services multiple flows using a packet scheduling algorithm. A common property of many of the packet scheduling algorithms proposed in the literature to provide QoS guarantees is that they ensure that the service received by flow f is approximately the same as service received by it in a server that has capacity r_f and services only flow f. They achieve this objective by ensuring that the departure time of a packet of flow f is at most β (β is a constant) more than the departure time in a server of rate r_f. The class of guaranteed rate scheduling algorithms consists of algorithms that provide such a guarantee.

To formally define the guaranteed rate class (referred to as GR) we define guaranteed rate clock value of a packet. Intuitively, guaranteed rate clock value of a packet is the departure time of a packet in a server of rate r_f. To formally define the guaranteed rate clock values, let p_f^j and l_f^j denote the j^{th} packet of flow f and its length, respectively. Let r_f^i (bits/s) be the rate allocated to flow f at server i and $A^i(p_f^j)$ denote the arrival time of packet p_f^j at server i. Let *expected arrival time* of a packet be the time at which it begins service in a server of rate r_f^i. Then, expected arrival time of p_f^j, denoted by $EAT^i(p_f^j, r_f^i)$, is defined as [61]:

$$EAT^i(p_f^j, r_f^i) = \max\{A^i(p_f^j), EAT^i(p_f^{j-1}, r_f^i) + \frac{l_f^{j-1}}{r_f^i}\} \tag{2.1}$$

where $EAT^i(p_f^0, r_f^i) + \frac{l_f^0}{r_f^i} = 0$. Since it takes $\frac{l_f^j}{r_f^i}$ time to service a packet of length l_f^j in a server of rate r_f^i, if the packet began service at time $EAT^i(p_f^j, r_f^i)$, it will finish service at time

$EAT^i(p_f^j, r_f^i) + \frac{l_f^j}{r_f^i}$. Thus, the guaranteed rate clock value for packet p_f^j at server i, denoted by $GRC^i(p_f^j, r_f^i)$, is given as:

$$GRC^i(p_f^j, r_f^i) = EAT^i(p_f^j, r_f^i) + \frac{l_f^j}{r_f^i} \quad j \geq 1 \tag{2.2}$$

Using (2.1) and (2.2), we can rewrite (2.2) as:

$$GRC^i(p_f^j, r_f^i) = \max\{A^i(p_f^j), GRC^i(p_f^{j-1}, r_f^i)\} + \frac{l_f^j}{r_f^i} \tag{2.3}$$

where $GRC^i(p_f^0, r_f^i) = 0$ [1]. We use the guaranteed rate clock value of a packet to define the class of GR scheduling algorithms as follows.

Definition 2.1 *A scheduling algorithm at server i belongs to class GR for flow f if it guarantees that packet p_f^j will be transmitted by $GRC^i(p_f^j, r_f^i) + \beta^i$ where β^i is a constant which depends on the scheduling algorithm and the server.*

Observe that the first term in the right hand side of (2.3) is the expected arrival time of a packet and the second term can be interpreted as the deadline of the packet. Thus, the scheduling algorithms in GR guarantee a bound on departure time of a packet based on the expected arrival time of the packet. Such a guarantee is referred to as *delay guarantee* [55].

The definition of guaranteed rate clock assumes that a fixed rate is allocated to packets of a flow. However, to efficiently support variable bit rate flows such as video, a packet scheduling algorithm may allocate variable rates to the packets of a flow. The GR class as defined does not include such algorithms. To address this limitation, we generalize the definition of the guaranteed rate clock value. Let $r_f^{j,i}$ denote the rate allocated to packet p_f^j at server i. Then, the generalized guaranteed rate clock value of packet p_f^j, denoted as $GRC^i(p_f^j, r_f^{j,i})$, is given as follows:

$$GRC^i(p_f^j, r_f^{j,i}) = \max\{A^i(p_f^j), GRC^i(p_f^{j-1}, r_f^{j-1,i})\} + \frac{l_f^j}{r_f^{j,i}} \quad j \geq 1 \tag{2.4}$$

[1] The definition of guaranteed rate clock is the same as virtual clock [62]. We have coined a new term for the virtual clock concept to differentiate between the general concept of virtual clock and as it relates to the Virtual Clock scheduling algorithm.

where $GRC^i(p_f^0, r_f^{j,i}) = 0$[2]. Thus, the GR class is redefined as follows:

Definition 2.2 *A scheduling algorithm at server i belongs to class GR for flow f if it guarantees that packet p_f^j will be transmitted by $GRC^i(p_f^j, r_f^{j,i}) + \beta^i$ where β^i is a constant which depends on the scheduling algorithm and the server.*

From the definition, we observe that there are two key properties of the class of GR scheduling algorithms:

- They provide a bound on the departure time of a packet without making any assumptions about the traffic characteristics of a flow. This generality enables a GR server to provide QoS guarantees such as upper bound on packet delay and loss to flows with any traffic characteristics. To illustrate, let $L^i(p_f^j)$ denote the departure time of packet p_f^j from server i. Then,

  $$L^i(p_f^j) \leq GRC^i(p_f^j, r_f^{j,i}) + \beta^i$$

  Since packet p_f^j arrives at time $A^i(p_f^j)$, its delay, denoted by d_f^j, is given as:

  $$d_f^j \leq GRC^i(p_f^j, r_f^{j,i}) - A^i(p_f^j) + \beta^i$$

  To simplify the exposition, let fixed rate r_f^i be allocated to the packets of flow f. Then, $GRC^i(p_f^j, r_f^{j,i})$ is the departure time of p_f^j in a server of capacity r_f^i. Consequently, $GRC^i(p_f^j, r_f^{j,i}) - A^i(p_f^j)$ is the delay experienced by p_f^j in a server of capacity r_f^i. This delay is governed only by the traffic characteristics of flow f and the rate r_f^i. Thus, if the traffic characteristics of flow f are known, an upper bound on delay in the fixed rate server can be determined. Consequently, an upper bound on delay in GR server can also be determined. Similarly, given the buffer space, an upper bound on loss rate can be determined. Thus, GR servers can provide QoS guarantees to flows with any traffic characteristics.

[2] Since link-level header length may vary in an internetwork, the length of a packet does not include the link-level header. To accommodate the effect of link-level headers, we require the actual rate allocation, denoted by $\overline{r}_f^{j,i}$, at server i to be such that $\frac{l_f^j}{r_f^{j,i}} = \frac{l_f^j + \overline{l}_f^{j,i}}{\overline{r}_f^{j,i}}$ where $\overline{l}_f^{j,i}$ is the length of link-level header at server i. Consequently, guaranteed rate clock value of a packet with and without the link-level headers is the same. Hence, we will always refer to the guaranteed rate clock value of a packet without the link-level headers.

- They provide a delay guarantee for a flow that is independent of the behavior of other flows in the network, i.e., they isolate the flows. Isolation of flows enables a network to provide stronger guarantees and is highly desirable, especially in large heterogeneous networks where sources may be malicious [6, 12, 36, 45].

Thus, GR class has desirable properties.

In the following subsections, we show that many of the work conserving as well as non-work conserving scheduling algorithms that either allocate only rate or separate rate and delay allocation belong to GR. To show that a scheduling algorithm belongs to GR, we would be required to prove a bound on the departure time of a packet. As observed in [12, 27], it is typically easier to bound the departure time of a packet in *preemptive* scheduling algorithms. Hence, even though packet scheduling algorithms are inherently *non-preemptive* in nature, to show that a scheduling algorithm belongs to GR, we employ a proof methodology, similar to the methodology in [27, 42], in which we first prove a bound on the departure time of a packet in preemptive scheduling algorithm, and then use a relationship between the departure times of a packet in equivalent preemptive and non-preemptive scheduling algorithm. In what follows, we establish a relationship between preemptive and non-preemptive scheduling algorithms.

2.2.1 Preemptive and Non-Preemptive Scheduling

Many of the scheduling algorithms that we will consider, assign a priority to a packet on its arrival and then schedule the packets in the priority order. In these scheduling algorithms, a packet with higher priority may arrive after a packet with lower priority has been scheduled. In *non-preemptive* scheduling algorithms, transmission of a lower priority packet is not preempted even after a higher priority packet arrives. Consequently, such algorithms ensure that the packet in service is the packet with the highest priority only after the transmission of every packet. On the other hand, a *preemptive* scheduling algorithm always ensures that the packet in service is the packet with the highest priority by possibly preempting the transmission of a packet with lower priority. In contrast to preemptive and non-preemptive algorithms, a *partially-preemptive* scheduling algorithm ensures that the packet in service is the packet with the highest priority after the transmission of every fragment of a packet.

Partially-preemptive algorithms would be helpful in defining scheduling algorithms suitable for servers where fragmentation may occur. Furthermore, non-preemptive algorithms are a subset of partially-preemptive algorithms. Hence, in Theorem 2.1 we establish a relationship between *equivalent* preemptive and partially-preemptive scheduling algorithms. A partially-preemptive scheduling algorithm is considered *equivalent* to a preemptive algorithm if the priority assigned to all the packets is the same in both the algorithms.

Theorem 2.1 *Let PS be a work conserving preemptive scheduling algorithm that assigns priority $\hat{L}_{PS}(p^j)$ to packet p^j, schedules packets in the increasing order of priority and guarantees that regardless of the arrival pattern of other packets, packet p^j will be transmitted by $\hat{L}_{PS}(p^j)$. Let PPS be its equivalent partially-preemptive scheduling algorithm, then*

$$L_{PPS}(p^j) - \hat{L}_{PS}(p^j) \leq \frac{\hat{l}_{max}}{C} \tag{2.5}$$

where $L_{PPS}(p^j)$ denotes the time a packet leaves the server when PPS scheduling algorithm is employed. Also, $\hat{l}_{max}$ is the maximum length of a packet fragment and C is the capacity of the server.

Proof: The structure of the proof is similar to the proof of Theorem 1 of [40] and Theorem 1 of [42]. Observe that since PS and its equivalent PPS are work conserving, their busy periods are the same. Hence, it is sufficient to show that (2.5) holds for all the packets served in a busy period. Let a busy period begin at time t_0 and the packets be indexed by the order in which they complete service in the PPS, that is the i^{th} packet to complete service in PPS is packet p^i.

Consider packet p^j. There are two cases to consider

- A fragment of packet p^n such that $\hat{L}_{PS}(p^n) > \hat{L}_{PS}(p^j)$ is served in the interval $[t_0, L_{PPS}(p^j)]$: Let t_1 be the largest time at which such a packet fragment is scheduled (see Figure 2.2). Also, let t_2 be the least time greater than t_1 such that if a fragment of packet p^k is scheduled in the interval $[t_2, L_{PPS}(p^j)]$, then $\hat{L}_{PS}(p^k) \leq \hat{L}_{PS}(p^j)$. From the definition of t_2 and t_1, we conclude that t_2 can be greater than t_1 by at most the time required to service the packet fragment scheduled at t_1, i.e.,

$$t_2 - t_1 \leq \frac{\hat{l}_{max}}{C}$$

Let S be the set of packets which have at least one fragment scheduled in $[t_2, L_{PPS}(p^j)]$. Since PPS schedules packet fragments based on the priority of the packets, all the packets in S must have arrived after t_1, i.e.,

$$\min_{k \in S}\{A(p^k)\} > t_1$$

PS guarantees that packet p^j will be transmitted by $\hat{L}_{PS}(p^j)$ regardless of the arrival pattern of the other packets. In particular it guarantees that even for an arrival pattern in which the packets in S are serviced before the departure of p^j, the departure time of p^j is at most $\hat{L}_{PS}(p^j)$. Therefore,

$$\hat{L}_{PS}(p^j) \geq t_1 + \sum_{k \in S} \frac{l^k}{C}$$

Since $t_2 - t_1 \leq \frac{\hat{l}_{max}}{C}$,

$$\hat{L}_{PS}(p^j) \geq t_2 - \frac{\hat{l}_{max}}{C} + \sum_{k \in S} \frac{l^k}{C}$$

Since $t_2 + \sum_{k \in S} \frac{l^k}{C} \geq L_{PPS}(p^j)$

$$\begin{aligned} \hat{L}_{PS}(p^j) &\geq L_{PPS}(p^j) - \frac{\hat{l}_{max}}{C} \\ \frac{\hat{l}_{max}}{C} &\geq L_{PPS}(p^j) - \hat{L}_{PS}(p^j) \end{aligned}$$

- No fragment of packet p^n such that $\hat{L}_{PS}(p^n) > \hat{L}_{PS}(p^j)$ is served in the interval $[t_0, L_{PPS}(p^j)]$: Let S be the set of packets served in the interval $[t_0, L_{PPS}(p^j)]$. Since PS guarantees that packet p^j will be transmitted by $\hat{L}_{PS}(p^j)$ regardless of the arrival pattern of the other packets; it guarantees that even for an arrival pattern in which the packets in S are serviced before the departure of p^j, the departure time of p^j is at most $\hat{L}_{PS}(p^j)$. Hence,

$$\begin{aligned} \hat{L}_{PS}(p^j) &\geq t_0 + \sum_{k \in S} \frac{l^k}{C} \\ \hat{L}_{PS}(p^j) &\geq L_{PPS}(p^j) \end{aligned}$$

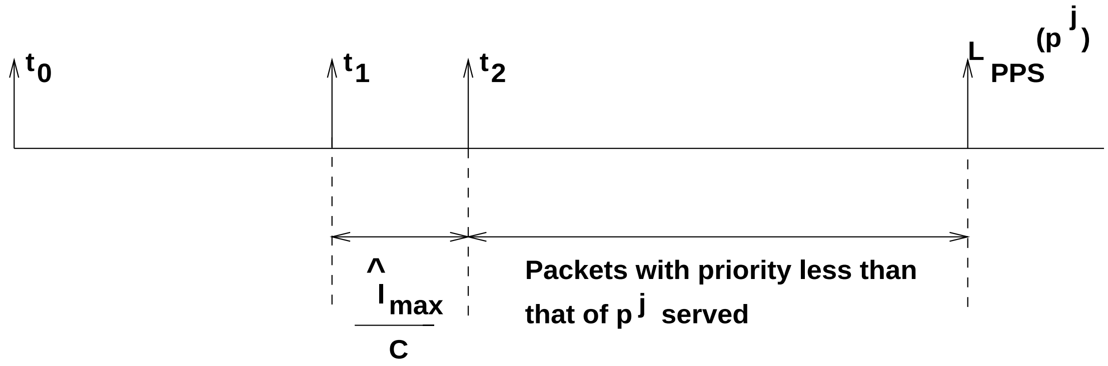


Figure 2.2: Time-line in the partially-preemptive scheduling algorithm

■

Theorem 2.1 is superficially similar to Theorem 1 of [42] and its generalization in [17] which demonstrate that if a scheduling algorithm satisfies some assumptions, then the departure time of a packet in a non-preemptive scheduling algorithm is at most $\frac{\hat{l}_{max}}{C}$ more than its departure time in a preemptive scheduling algorithm. Some of the algorithms that we will be considering (for example, Virtual Clock) do not satisfy the assumptions required in those theorems. Furthermore, Theorem 2.1 proves a weaker result, which suffices for our purpose, then the results in [17, 42]. Specifically, whereas the results in [17, 42] bound the difference in departure time of a packet in a preemptive and non-preemptive scheduling algorithm, Theorem 2.1 relates the departure time of a packet in a non-preemptive algorithm to the *bound* on the departure time of the packet in a preemptive scheduling algorithm.

We now show that, using Theorem 2.1 wherever applicable, most of the work conserving and non-work conserving scheduling algorithms that either allocate only rate or separate rate and delay allocation belong to GR.

2.2.2 Work Conserving Algorithms

We first show that several scheduling algorithms that allocate only rate only belong to GR. We then show that algorithms which separate delay and rate allocation also belong to GR.

A common characteristic of the work-conserving algorithms that allocate only rate is that they do not allow the rate allocated to flows to vary over time. Since to achieve efficient utilization of network resources, variable rate may be allocated to flows carrying VBR video, these algorithms may not be able to provide QoS guarantees to VBR video. We address this limitation by defining several variable rate allocation algorithms in the next section.

2.2.2.1 Variable Rate Allocation Algorithms

In this section, we define work conserving generalized Virtual Clock, WFQ, and SCFQ algorithms that allocate variable rate to packets of a flow and show that all of these scheduling algorithms belong to GR. Generalized Virtual Clock, WFQ, and SCFQ when employed to allocate fixed rate to the packets of a flow, i.e., $r_f^{j,i} = r_f^i$, are the same as Virtual Clock, WFQ, and SCFQ, respectively. Hence, Virtual Clock, WFQ, and SCFQ also belong the GR class. These algorithms have different delay and fairness properties as well as implementation complexity, and hence demonstrate that the GR class is broad (an exposition of these algorithms along these dimensions can be found in [22, 42]). We will analyze the algorithms assuming that rate allocation may vary with every packet. However, a particular network architecture may choose to allocate variable rate for every message [36], in response to change in reservation [28, 25], or employ some other policy.

Virtual Clock We define generalized Virtual Clock (VC) scheduling algorithm analogous to the Virtual Clock algorithm [62]. The generalized VC algorithm is defined as follows:

1. On arrival at server i, packet p_f^j assigned rate $r_f^{j,i}$ is stamped with virtual clock value, denoted by $VC^i(p_f^j, r_f^{j,i})$, computed as:

$$VC^i(p_f^j, r_f^{j,i}) = \max\{A^i(p_f^j), VC^i(p_f^{j-1}, r_f^{j-1,i})\} + \frac{l_f^j}{r_f^{j,i}} \tag{2.6}$$

   where $VC^i(p_f^0, r_f^{0,i}) = 0$.

2. Packets are serviced in increasing order of the virtual clock value.

We show that generalized VC belongs to GR by first proving a bound on the departure time of a packet in preemptive VC. To do so, let us first define $R^i_f(t)$ for flow f as follows:

$$R^i_f(t) = \begin{cases} r^{j,i}_f & \text{if } \exists j \ni \left(A^i(p^j_f) \leq t\right) \wedge \left(VC^i(p^{j-1}_f, r^{j-1,i}_f) < t \leq VC^i(p^j_f, r^{j,i}_f)\right) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Let S be the set of flows served by server i. Then server i with capacity C^i is defined to have exceeded its capacity at time t if $\sum_{n\in S} R^i_n(t) > C^i$. The following theorem bounds the departure time of a packet in preemptive VC.

Theorem 2.2 *If a server's capacity is not exceeded, then the time at which the transmission of packet p^j_f is completed in preemptive VC, denoted by $L^i_{P-VC}(p^j_f)$ is:*

$$L^i_{P-VC}(p^j_f) \leq VC^i(p^j_f, r^{j,i}_f)$$

Proof: It is easily observed from (2.6) and (2.7) that the cumulative length of all flow n packets that arrive in interval $[t_1, t_2]$ and have virtual clock value no greater than t_2, denoted by $AP_n(t_1, t_2)$, is given as:

$$AP_n(t_1, t_2) \leq \int_{t_1}^{t_2} R^i_n(t)dt$$

We now prove the theorem by contradiction. Let for packet p^j_f, $L^i_{P-VC}(p^j_f) >$ $VC^i(p^j_f, r^{j,i}_f)$. Also, let t_0 be the beginning of the busy period in which p^j_f is served and $t_2 = VC^i(p^j_f, r^{j,i}_f)$. Let t_1 be the least time smaller than t_2 during the busy period such that no packet with virtual clock value greater than t_2 is served in the interval $[t_1, t_2]$ (such a t_1 exists). Clearly, all the packets served in the interval $[t_1, t_2]$ arrive in this interval and have virtual clock value less than or equal to t_2. Since the server is busy in the interval $[t_1, t_2]$ and packet p^j_f is not serviced by t_2, we have:

$$\sum_{n\in S} AP_n(t_1, t_2) \quad > \quad C^i(t_2 - t_1)$$

$$\begin{aligned}\sum_{n \in S} \int_{t_1}^{t_2} R_n^i(t)dt &> C^i(t_2 - t_1) \\ \int_{t_1}^{t_2} \sum_{n \in S} R_n^i(t)dt &> C^i(t_2 - t_1) \end{aligned} \tag{2.8}$$

Since the server capacity is not exceeded, $\sum_{n \in S} R_n^i(t) \leq C^i$. Hence, $\int_{t_1}^{t_2} \sum_{n \in S} R_n^i(t)dt \leq C^i(t_2 - t_1)$. This contradicts (2.8), and hence the theorem follows. ■

Since preemptive VC algorithm is work conserving and guarantees that packet p_f^j will be transmitted by $VC^i(p_f^j, r_f^{j,i})$ for any arrival pattern of packets, Corollary 2.1 immediately follows from Theorems 2.1 and 2.2.

Corollary 2.1 *If a server's capacity is not exceeded, then the time at which the transmission of packet p_f^j is completed in generalized Virtual Clock, denoted by $L_{VC}^i(p_f^j)$ is:*

$$L_{VC}^i(p_f^j) \leq VC^i(p_f^j, r_f^{j,i}) + \frac{l_{max}^i}{C^i}$$

where C^i is the capacity of the server and l_{max}^i is the maximum length of a packet serviced by server i.

Since the equations for virtual clock and guaranteed rate clock are the same, it is easily observed that generalized Virtual Clock scheduling algorithm belongs to GR for flow f with $\beta^i = \frac{l_{max}^i}{C^i}$.

Observe that generalized Virtual Clock is work conserving and permits variable rate to be allocated to the packets of a flow as long as the capacity of a server is not exceeded. This is in contrast to the non-work conserving Burst Scheduling algorithm presented in [36] in which Virtual Clock scheduling algorithm has been employed to allocate variable rate by defining the notion of an *active flow*. A flow has a constant rate allocated to it as long as it is active. Rate assignment of a flow is changed only after it makes a transition from active to inactive state. Hence, to allocate variable rate, a *flow regulator* which enforces such transitions is required to be implemented at each server. Generalized Virtual Clock does not have such a requirement, and hence reduces the implementation complexity.

Weighted Fair Queuing Weighted Fair Queuing (WFQ) scheduling algorithm is a practical realization of Generalized Processor Sharing(GPS) service discipline [12] (WFQ is also known as Packet-by-packet Generalized Processor Sharing (PGPS) [42]). We first show that GPS belongs to GR and then show that a generalized virtual time implementation of WFQ belongs to GR.

GPS is a fluid flow service discipline in which flows receive service in infinitesimally divisible units. In GPS, each flow f is assigned a constant ϕ_f^i at server i. GPS guarantees that if flow f is backlogged at time t, then it receives service at rate $\frac{\phi_f^{j,i} C^i}{\sum_{k \in B^i(t)} \phi_k^i}$ where $B^i(t)$ is the set of backlogged flows at time t and C^i is the capacity of the server. To allocate variable rate to packets of a flow, we assign a different constant $\phi_f^{j,i}$ to each packet p_f^j. From the definition of GPS we know that at time t, packet p_f^j will be serviced at the rate of $\frac{\phi_f^{j,i} C^i}{\sum_{k \in B^i(t)} \phi_k^{a,i}}$ where packet p_k^a of flow k is in service at time t. Hence, we define a GPS server to have assigned rate $r_f^{j,i}$ to packet p_f^j if $\frac{\phi_f^{j,i} C^i}{\sum_{k \in B^i(t)} \phi_k^{a,i}} \geq r_f^{j,i}$ as long as the packet is in service. Therefore, if packet p_f^j is assigned rate $r_f^{j,i}$, it is served at least at rate $r_f^{j,i}$. Since GPS serves packets of a flow in FCFS order, we get:

$$L_{GPS}^i(p_f^j) \leq \max\{A^i(p_f^j), L_{GPS}^i(p_f^{j-1})\} + \frac{l_f^j}{r_f^{j,i}} \tag{2.9}$$

Let $L_{GPS}^i(p_f^0) = GRC^i(p_f^0, r_f^{0,i}) = 0$. From (2.9) and (2.4), it can be shown that

$$L_{GPS}^i(p_f^j) \leq GRC^i(p_f^j, r_f^{j,i}) \tag{2.10}$$

Hence, GPS belongs to GR. We now define a virtual time implementation of packet-by-packet GPS which is a generalization of the implementation in [42]. Let $v^i(t)$ be the virtual time associated with server i at time t. Let $v^i(0) = 0$ and $v^i(t)$ not change when no packet is backlogged. Otherwise, define $v^i(t)$ as:

$$\frac{dv^i(t)}{dt} = \frac{C^i}{\sum_{k \in b^i(t)} \phi_k^{a,i}} \tag{2.11}$$

Define finish tag of packet p_f^j, denoted by $F^i(p_f^j, \phi_f^{j,i})$ as:

$$F^i(p_f^j, \phi_f^{j,i}) = \max\{v^i(A^i(p_f^j)), F^i(p_f^{j-1}, \phi_f^{j-1,i})\} + \frac{l_f^j}{\phi_f^{j,i}} \tag{2.12}$$

where $F^i(p_f^0, \phi_f^{0,i}) = 0$. If finish tag of a packet is defined as above and $\hat{t}$ is the time at which packet p_f^j departs, then $v^i(\hat{t}) = F^i(p_f^j, \phi_f^{j,i})$. To observe this, consider packet p_f^j of flow f. Let $W_f^{j,i}(t)$ be the amount of service that packet p_f^j has received within time interval t after it begins service. From the definition of GPS, we get:

$$\frac{dW_f^{j,i}(t)}{dt} = \frac{C^i \phi_f^{j,i}}{\sum_{k \in b^i(t)} \phi_k^{a,i}} \tag{2.13}$$

Hence from (2.11) and (2.13), we get:

$$\frac{dv^i(t)}{dW_f^{j,i}(t)} = \frac{1}{\phi_f^{j,i}} \tag{2.14}$$

From (2.14) we conclude, if packet p_f^j begins service at virtual time $\hat{v}$, it will depart at virtual time $\hat{v} + \frac{l_f^j}{\phi_f^{j,i}}$. Since GPS serves packets of a flow in FCFS order, it is easy to observe from (2.12) that if $\hat{t}$ is the time at which packet p_f^j departs, then $v^i(\hat{t}) = F^i(p_f^j, \phi_f^{j,i})$. Since $v^i(t)$ is monotonically increasing, packets leave a GPS server in the increasing order of finish tag. Hence, a scheduling algorithm that schedules packets in increasing order of the finish tags will simulate GPS. However, as WFQ is non-preemptive, from (2.10) and Theorem 1 of [42], we get

$$L_{WFQ}^i(p_f^j) \leq GRC^i(p_f^j, r_f^{j,i}) + \frac{l_{max}^i}{C^i}$$

Hence, WFQ scheduling algorithm belongs to GR for flow f with $\beta^i = \frac{l_{max}^i}{C^i}$.

Self Clocked Fair Queuing The Self Clocked Fair Queuing scheme, originally proposed in [11] and later analyzed in [22], was designed to facilitate the implementation of a fair queuing scheme in broadband networks. We define a generalized SCFQ algorithm which can allocate variable rate to packets of a flow analogous to SCFQ as follows:

1. On arrival, a packet p_f^j is stamped with finish tag $F^i(p_f^j, r_f^{j,i})$, computed as:

$$F^i(p_f^j, r_f^{j,i}) = \max\{v^i(A^i(p_f^j)), F^i(p_f^{j-1}, r_f^{j-1,i})\} + \frac{l_f^j}{r_f^{j,i}}$$

where $F^i(p_f^0, r_f^{j,0}) = 0$.

2. The server virtual time at time t, $v^i(t)$, is defined to be equal to the finish tag of the packet in service at time t. $v^i(t) = t$ when the server is idle.

3. Packets are serviced in increasing order of their finish tags.

Define $R^i_f(v)$ for flow f as follows:

$$R^i_f(v) = \begin{cases} r^{j,i}_f & \text{if } \exists j \ni \left(v^i(A^i(p^j_f)) \leq v\right) \wedge \left(F^i(p^{j-1}_f, r^{j-1,i}_f) < v \leq F^i(p^j_f, r^{j,i}_f)\right) \\ 0 & \text{otherwise} \end{cases}$$

Let S be the set of flows served by server i. Then server i with capacity C^i is defined to have exceeded its capacity at virtual time v if $\sum_{n \in S} R^i_n(v) > C^i$. The following theorem proves that SCFQ algorithm also belongs to the class of GR scheduling algorithms.

Theorem 2.3 *If the server's capacity is not exceeded, then the departure time of packet p^j_f in SCFQ, denoted by $L^i_{SCFQ}(p^j_f)$, is given by*

$$L^i_{SCFQ}(p^j_f) \leq GRC^i(p^j_f, r^{j,i}_f) + \sum_{n \in S \wedge n \neq f} \frac{l^{max}_n}{C^i}$$

where l^{max}_n is the maximum length for packets in flow n.

Proof: Let set B^i_f be defined as follows.

$$B^i_f = \{n | n > 0 \wedge F^i(p^{n-1}_f, r^{n-1,i}_f) < v^i(A^i(p^n_f))\}$$

Let $k \leq j$ be largest integer in B^i_f. Also, let $v_1 = v^i(A^i(p^k_f))$ and $v_2 = F^i(p^j_f, r^{j,i}_f)$. The set of packets served by the server in the virtual time interval $[v_1, v_2]$ can be partitioned into two sets:

- This set consists of packets p^m_n such that $F^i(p^m_n, r^{m,i}_n) \leq v_2$ and $\max\{v^i(A^i(p^m_n)),$ $F^i(p^{m-1}_n, r^{m-1,i}_n)\} \geq v_1$. Then, from the definition of $R^i_n(v)$ and $F^i(p^m_n, r^{m,i}_n)$, we know that the cumulative length of such flow n packets served by the server in the virtual time interval $[v_1, v_2]$, denoted by $AP_n(v_1, v_2)$, is given as:

$$AP_n(v_1, v_2) \leq \int_{v_1}^{v_2} R^i_n(v) dv$$

Hence, aggregate length of packets in this set, $\sum_{n\in S} AP_n(v_1,v_2)$, is given as:

$$\begin{aligned}
\sum_{n\in S} AP_n(v_1,v_2) &\leq \sum_{n\in S}\int_{v_1}^{v_2} R_n^i(v)dv \\
&\leq \int_{v_1}^{v_2}\sum_{n\in S} R_n^i(v)dv \\
&\leq \int_{v_1}^{v_2} C^i dv \\
&\leq C^i(v_2-v_1)
\end{aligned}$$

But $v_2 - v_1 = \sum_{n=0}^{n=j-k}\frac{l_f^{k+n}}{r_f^{k+n,i}}$. Hence,

$$\sum_{n\in S} AP_n(v_1,v_2) \leq C^i\sum_{n=0}^{n=j-k}\frac{l_f^{k+n}}{r_f^{k+n,i}}$$

- This set consists of packets p_n^m such that $F^i(p_n^m, r_n^{m,i}) \geq v_1$ and $\max\{v^i(A^i(p_n^m)),$ $F^i(p_n^{m-1}, r_n^{m-1,i})\} < v_1$. At most one packet of all flows other than f can belong to this set. Consequently, the maximum aggregate of packets in this set is $\sum_{n\in S\wedge n\neq f} l_n^{max}$.

Hence, the aggregate length of packets served by the server in the interval $[v_1, v_2]$, is less than or equal to:

$$C^i\sum_{n=0}^{n=j-k}\frac{l_f^{k+n}}{r_f^{k+n,i}} + \sum_{n\in S\wedge n\neq f} l_n^{max}$$

Since packet p_f^j departs at system virtual time v_2, we get:

$$\begin{aligned}
A^i(p_f^k) + \frac{C^i\sum_{n=0}^{n=j-k}\frac{l_f^{k+n}}{r_f^{k+n,i}} + \sum_{n\in S\wedge n\neq f} l_n^{max}}{C^i} &\geq L_{SCFQ}^i(p_f^j) \\
A^i(p_f^k) + \sum_{n=0}^{n=j-k}\frac{l_f^{k+n}}{r_f^{k+n,i}} + \sum_{n\in S\wedge n\neq f}\frac{l_n^{max}}{C^i} &\geq L_{SCFQ}^i(p_f^j)
\end{aligned}$$

From (2.4) we get

$$GRC^i(p_f^j, r_f^{j,i}) + \sum_{n\in S\wedge n\neq f}\frac{l_n^{max}}{C^i} \geq L_{SCFQ}^i(p_f^j)$$

■

Hence, SCFQ scheduling algorithm belongs to GR for flow f with $\beta^i = \sum_{n\in S\wedge n\neq f}\frac{l_n^{max}}{C^i}$.

2.2.2.2 Delay Allocation Algorithms

Though several algorithms which allocate rate have been proposed, Delay Earliest Due Date (Delay EDD) is the only work conserving scheduling algorithm that separates delay and rate allocation in a networking environment [3]. In this section, we show that Delay EDD also belongs to GR.

It is not known whether Delay EDD can separate delay and rate allocation while assigning variable rate to packets of a flow. Hence, we assume that rate r_f^i is assigned to all packets of a flow. Furthermore, in Delay-EDD, the length of the packets is assumed to be the same, i.e., $l_f = l_f^j$. Then Delay-EDD is defined as follows:

1. If d_f^i is the delay bound for flow f at server i, then on arrival packet p_f^j is stamped with a deadline, denoted by $D^i(p_f^j)$, which is computed as follows:

$$D^i(p_f^j) = \max\{A^i(p_f^j), GRC^i(p_f^{j-1}, r_f^i)\} + d_f^i \tag{2.15}$$

2. Packets are served in the increasing order of deadline.

It was shown in [13, 64] that if certain schedulability conditions are met and the minimum inter-arrival time of packets is at least $\frac{l_f}{r_f^i}$, then a packet would depart by $D^i(p_f^j)$. However, in a networking environment even if the minimum inter-arrival time is at least $\frac{l_f}{r_f^i}$ at the network entry point, it may become smaller than $\frac{l_f}{r_f^i}$ at a server which is downstream on the path of a flow. This problem was addressed in [31, 64] by requiring the clocks of the servers to be synchronized. We demonstrate that this is an unnecessary restriction by proving that regardless of the inter-arrival time of packets, preemptive Delay EDD guarantees that packet p_f^j will be transmitted by $D^i(p_f^j)$. We then use this property to show that Delay EDD belongs to GR.

Theorem 2.4 *If S is the set of flows serviced by the server and*

$$\forall t > 0 : \quad \sum_{n \in S} \max\{0, \left\lceil \frac{(t - d_n^i) r_n}{l_n} \right\rceil \frac{l_n}{C^i}\} \leq t \tag{2.16}$$

[3] Consistent Relative Session Treatment (CRST) rate assignment has been used in WFQ networks to separate rate and delay allocation. However, since this rate assignment requires all the flows to conform to leaky bucket specification, we do not consider it further.

then the time at which the transmission of packet p_f^j is completed in preemptive Delay EDD, denoted by $L_{P-EDD}^i(p_f^j)$ is:

$$L_{P-EDD}^i(p_f^j) \leq D^i(p_f^j)$$

Proof: From (2.15) and (2.4), we conclude that the cumulative length of packets of flow f that arrive in the interval $[t_1, t_2]$ and have deadline less than or equal to t_2, denoted by $AP_f(t_1, t_2)$, is given as:

$$AP_f(t_1, t_2) \leq \left\lceil \frac{(t_2 - t_1 - d_f^i) r_f}{l_f} \right\rceil l_f \tag{2.17}$$

We now prove the theorem by contradiction. Let for packet p_f^j, $L_{P-EDD}^i(p_f^j) > D^i(p_f^j)$. Also, let t_0 be the beginning of the busy period in which p_f^j is served and $t_2 = D^i(p_f^j)$. Let t_1 be the least time less than t_2 during the busy period such that no packet with deadline greater than t_2 is served in the interval $[t_1, t_2]$ (such t_1 exists). Clearly, all the packets served in the interval $[t_1, t_2]$ arrive in this interval and have deadline less than or equal to t_2. Since the server is busy in the interval $[t_1, t_2]$ and packet p_f^j is not serviced by t_2, from (2.17) we have:

$$\sum_{n \in S} \max\{0, \left\lceil \frac{(t_2 - t_1 - d_n^i) r_n}{l_n} \right\rceil \frac{l_n}{C^i}\} > (t_2 - t_1) \tag{2.18}$$

Substituting $t = (t_2 - t_1)$ in (2.18) contradicts (2.16) and hence theorem follows. ■

Due to high computational complexity, it may not be feasible to employ (2.16) as schedulability test. Hence, conditions stronger than (2.16) which have lower computational complexity have been developed in [64]. Clearly, the theorem holds under the stronger conditions developed in [64] as well.

Since Delay EDD guarantees that packet p_f^j will be transmitted by $D^i(p_f^j)$ regardless of the arrival pattern of the packets, the following corollary is immediate from Theorem 2.1 and Theorem 2.4.

Corollary 2.2 *If (2.16) is satisfied, then the time at which transmission of packet p_f^j is completed in Delay-EDD, denoted by $L_{EDD}^i(p_f^j)$, is given as:*

$$L_{EDD}^i(p_f^j) \leq D^i(p_f^j) + \frac{l_{max}^i}{C^i}$$

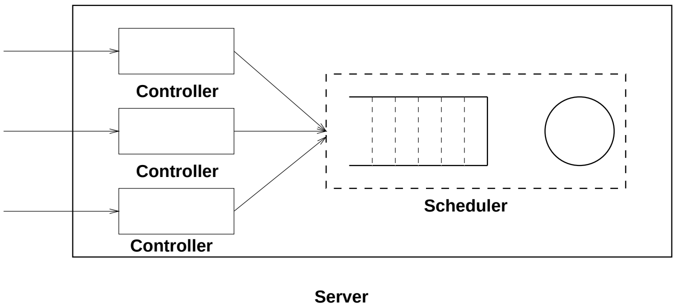


Figure 2.3: Rate Controlled Service Disciplines

where C^i is the capacity of the server and l^i_{max} is the maximum length of a packet serviced by server i.

To observe that Delay EDD belongs to GR, rewrite (2.15) as:

$$\begin{aligned} D^i(p_f^j) &= \max\{A^i(p_f^j), GRC^i(p_f^{j-1}, r_f^i)\} + \frac{l_f}{r_f^i} - (\frac{l_f}{r_f^i} - d_f^i) \\ &= GRC^i(p_f^j, r_f^i) - (\frac{l_f}{r_f^i} - d_f^i) \end{aligned}$$

Hence, Delay EDD belongs to GR with $\beta^i = \frac{l^i_{max}}{C^i} - (\frac{l_f}{r_f^i} - d_f^i)$. If schedulability conditions for non-preemptive Delay EDD are used, then one can similarly show that β^i is given as $\beta^i = -(\frac{l_f}{r_f^i} - d_f^i)$.

2.2.3 Non-Work Conserving Algorithms

A general framework for reasoning about the end-to-end performance guarantee of a class of non-work conserving algorithms termed *Rate Controlled Service Disciplines* has been presented in [60]. In this section, we show that some of the rate controlled service disciplines also belong to the GR class.

Rate controlled service disciplines consist of a rate controller and a scheduler (see Figure 2.3). The rate controller ensures that the traffic entering the scheduler conforms to a negotiated traffic specification and the scheduler guarantees that each packet of flow f would experience a maximum delay of d_f^i. In such disciplines, different types of rate controllers may be employed. Since these disciplines have been studied predominantly for constant rate allocation (see section 2.2.4.2 for non-work conserving disciplines that allocate variable rate), a common characteristic of most of the rate controllers is that they do not delay packets more than necessary to enforce the average rate. Hence, if r^i_f is the rate associated with flow f and l_f the length of packets, then the time at which a packet p^j_f departs the rate controller, denoted by $L^i_{RC}(p^j_f, r^i_f)$, is given as:

$$L^i_{RC}(p^j_f, r^i_f) \leq \max\{A^i(p^j_f), GRC^i(p^{j-1}_f, r^i_f)\} + \gamma^i \tag{2.19}$$

where γ^i is a constant for a rate controller. Since the scheduler guarantees a maximum delay of d^i_f to each packet, the time that packet p^j_f departs server i which employs a rate controlled service discipline, denoted by $L^i_{RCSD}(p^j_f)$, is given as:

$$\begin{aligned} L^i_{RCSD}(p^j_f) &\leq \max\{A^i(p^j_f), GRC^i(p^{j-1}_f, r^i_f)\} + \gamma^i + d^i_f \\ &\leq \max\{A^i(p^j_f), GRC^i(p^{j-1}_f, r^i_f)\} + \frac{l_f}{r^i_f} - (\frac{l_f}{r^i_f} - \gamma^i - d^i_f) \end{aligned}$$

From (2.4) we get:

$$L^i_{RCSD}(p^j_f) \leq GRC^i(p^j_f, r^i_f) - (\frac{l_f}{r^i_f} - \gamma^i - d^i_f)$$

Therefore, rate controlled service disciplines that employ rate controllers consistent with (2.19) also belong to the GR class. The rate controller for Rate Controlled Static Priority queuing is consistent with (2.19) and hence belongs to GR. Note that if a rate controlled service discipline employs a rate controller that does not satisfy (2.19), then it may not belong to the GR class. We discuss this issue in greater detail in Chapter 3.

2.2.4 Packet Fragmentation and Rate Control

In heterogeneous networks, packets may be fragmented. Furthermore, to reduce the buffer requirement in the network, some flows may require rate controllers to be employed on the path. We now consider: (1) scheduling algorithms suitable for servers where packet fragmentation may occur, and (2) the effect of employing rate controller for a flow on the delay guarantee.

2.2.4.1 Scheduling in Presence of Fragmentation

Consider a server that can receive packets that are larger than it can transmit (this situation may occur in an internetwork consisting of ethernet, ATM, and FDDI subnetworks). In such a case, the server has to fragment a packet before transmitting it. A fragmented packet can be scheduled for transmission by the server in various ways:

- The server can compute a priority for each packet fragment and hence schedule each fragment individually.
- The server can compute a priority for each packet and schedule each packet.

Whereas scheduling each packet fragment increases the computation overhead, from Theorem 2.1 we know that scheduling packets and disabling preemption of a packet transmission increases the maximum delay incurred. Hence, a scheduling algorithm that minimizes computational overhead while minimizing delay incurred due to non-preemption is desirable. Observe that since the transmission unit is a packet fragment, even if a server schedules packet, it can allow packet preemption to occur after the transmission of every packet fragment, i.e., it can schedule the packets in a *partially-preemptive* manner. This would simultaneously reduce the computational overhead and the delay incurred by the packets. Partially-preemptive equivalents of the generalized Virtual Clock, Delay EDD and WFQ can be defined using the definition of partially-preemptive scheduling algorithms. Also using Theorem 2.1, Theorem 2.2 and Theorem 2.4, we know that partially preemptive generalized Virtual Clock and Delay EDD also belong to GR with β^i being derived by substituting $\frac{l^i_{max}}{C^i}$ with $\frac{\hat{l}^i_{max}}{C^i}$ where $\hat{l}^i_{max}$ is the maximum length of a packet fragment served by server i. Similarly, Theorem 1 of [42] can be generalized in a manner analogous to Theorem 1 to show that partially-preemptive WFQ also belongs to GR with $\beta^i = \hat{l}^i_{max}$.

2.2.4.2 Effect of Rate Control

It has been observed that if a network employs a work-conserving scheduling algorithm, then the burstiness of a flow and hence buffer requirement increases at the downstream nodes. Several

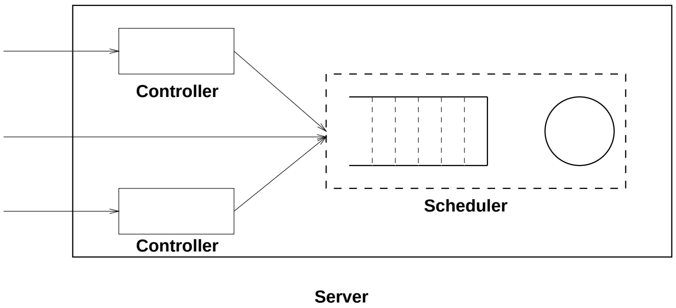


Figure 2.4: Rate controller employed for a subset of flows

non-work conserving scheduling algorithms have been proposed to address this limitation [19, 60]. Though non-work conserving algorithms reduce the buffer requirement in the network, they increase the average delay for all the flows. Even though this may be desirable for some flows, it may not be desirable for other flows. We address this limitation of non-work conserving algorithms by employing rate controllers on a per flow basis (see Figure 2.4). However, if arbitrary rate controllers are employed, then they may increase not only the average delay, but also the worst case delay for the flows for which they are employed. To avoid increase in the worst case delay, we require that if a rate controller is employed for flow f at the server, then it should guarantee that departure time of packet p_f^j associated with rate $r_f^{j,i}$ from the rate controller, denoted by $L_{RC}^i(p_f^j, r_f^{j,i})$, is given as:

$$L_{RC}^i(p_f^j, r_f^{j,i}) \leq \max\{A^i(p_f^j), GRC^i(p_f^{j-1}, r_f^{j-1,i})\} \tag{2.20}$$

In the special case of $r_f^i = r_f^{j,i}$, this definition captures the characteristics of rate control elements like leaky bucket. Let such a rate control element be employed at server i for flow f. Theorem 2.5 demonstrates that such a rate control element does not change the bound on the departure time of a packet when the scheduling algorithm employed is in GR.

Theorem 2.5 *If a rate control element that satisfies (2.20) is employed at server i for flow f, then:*

$$GRC^i(p_f^j, r_f^{j,i}) = \widehat{GRC}^i(p_f^j, r_f^{j,i})$$

where $GRC^i(p_f^j, r_f^{j,i})$ is the guaranteed rate clock value at the scheduler when no rate controller is employed for the flow and $\widehat{GRC}^i(p_f^j, r_f^{j,i})$ is the value when a rate controller is em-

ployed.

Proof: Let $A^i(p_f^j)$ be the arrival time of p_f^j at the rate controller when a rate controller is employed and at the scheduler when it is not. Also, let $\widehat{A}^i(p_f^j)$ be the arrival time of the packet at the scheduler (or equivalently, departure time at the rate controller) when a rate control element is employed. The proof is by induction on j.

Base Case: $j = 1$. As $GRC^i(p_f^0) = 0$, from (2.20) we know $A^i(p_f^j) = \widehat{A}^i(p_f^j)$. Hence we conclude:

$$\begin{aligned} \widehat{GRC}^i(p_f^1, r_f^{1,i}) &= \widehat{A}^i(p_f^1) + \frac{l_f^1}{r_f^{1,i}} \\ &= A^i(p_f^1) + \frac{l_f^1}{r_f^{1,i}} \\ &= GRC^i(p_f^1, r_f^{1,i}) \end{aligned}$$

Therefore the theorem holds for $j = 1$.

Induction Hypothesis: Let us assume that the theorem holds for $1 \leq j \leq m$.

Induction: We now show that the theorem holds for $1 \leq j \leq m+1$. From the definition of GRC:

$$\widehat{GRC}^i(p_f^{m+1}, r_f^{m+1,i}) = \max\{\widehat{A}^i(p_f^{m+1,i}), \widehat{GRC}^i(p_f^m, r_f^{m,i})\} + \frac{l_f^{m+1}}{r_f^{m+1,i}} \tag{2.21}$$

There are two cases to consider:

1. $\widehat{A}^i(p_f^{m+1}) \leq \widehat{GRC}^i(p_f^m, r_f^{m,i})$: From (2.21) we get:

$$\widehat{GRC}^i(p_f^{m+1}, r_f^{m+1,i}) = \widehat{GRC}^i(p_f^m, r_f^{m,i}) + \frac{l_f^{m+1}}{r_f^{m+1,i}}$$

   Using induction hypothesis we get:

$$\begin{aligned} \widehat{GRC}^i(p_f^{m+1}, r_f^{m+1,i}) &= GRC^i(p_f^m, r_f^{m,i}) + \frac{l_f^{m+1}}{r_f^{m+1,i}} \\ &= GRC^i(p_f^{m+1}, r_f^{m+1,i}) \end{aligned} \tag{2.22}$$

2. $\widehat{A}^i(p_f^{m+1}) > \widehat{GRC}^i(p_f^m, r_f^{m,i})$: From induction hypothesis, $\widehat{GRC}^i(p_f^m, r_f^{m,i}) = GRC^i(p_f^m, r_f^{m,i})$. Hence, we conclude $\widehat{A}^i(p_f^{m+1}) > GRC^i(p_f^m, r_f^{m,i})$. But from (2.20) this implies $\widehat{A}^i(p_f^{m+1}) = A^i(p_f^{m+1})$. Hence, from (2.21) we get:

$$\begin{aligned}
\widehat{GRC}^i(p_f^{m+1}, r_f^{m+1,i}) &= \widehat{A}^i(p_f^{m+1}) + \frac{l_f^{m+1}}{r_f^{m+1,i}} \\
&= A^i(p_f^{m+1}) + \frac{l_f^{m+1}}{r_f^{m+1,i}} \\
&= \max\{A^i(p_f^{m+1}), GRC^i(p_f^m, r_f^{m,i})\} + \frac{l_f^{m+1}}{r_f^{m+1,i}} \\
&= GRC^i(p_f^{m+1}, r_f^{m+1,i}) \qquad (2.23)
\end{aligned}$$

From (2.22), (2.23) and the induction hypothesis, we conclude that the theorem holds for $1 \leq j \leq m+1$. Hence the theorem follows. ■

Theorem 2.5 demonstrates that if a scheduling algorithm belongs to GR for a flow, then the equivalent non-work conserving algorithm obtained by employing any rate controller element satisfying (2.20) also belongs to GR. Observe that this defines non-work conserving equivalents of generalized Virtual Clock, WFQ, and SCFQ algorithms. It also defines a scheduling algorithm that combines Delay EDD and a variant of Jitter EDD.

2.2.5 Summary

In the previous sections, we have defined the class of GR scheduling algorithms and shown that several work conserving and non-work conserving algorithms that either allocate only rate or separate rate and delay allocation belong to GR. We defined generalized Virtual Clock, WFQ, and SCFQ scheduling algorithms that can allocate variable rate to the packets of a flow. These algorithms are work conserving and do not require a flow regulator. We also defined scheduling algorithms suitable for servers where packet fragmentation may occur. We demonstrated that if a class of rate controllers is employed for a flow in conjunction with any scheduling algorithm in GR, then the resulting non-work conserving algorithm also belongs to GR. This lead to the definition of non-work conserving equivalents of generalized Virtual Clock, WFQ, and SCFQ algorithms as well as combination of Delay EDD and a variant of Jitter EDD.

Several algorithms, other than those considered in the previous sections, also belong to GR. For example, since Frame-based Fair Queuing [52] and Worst-case Fair Weighted Fair Queuing [4] also guarantee that, regardless of behaviour of other flows in the network, the bound on departure time of a packet is the same as in WFQ, they also belong to GR. Similarly, since Deficit Round Robin (DRR [50]) also isolates flows and guarantees a minimum throughput, it can be shown that it also belongs to GR (although β for DRR will be very high). Hence, the class of GR algorithms is broad.

If a server employs any of these scheduling algorithm in GR, then it guarantees that packet p_f^j will be transmitted by $GRC^i(p_f^j, r_f^{j,i}) + \beta^i$. Since $GRC^i(p_f^j, r_f^{j,i})$, and hence the delay guarantee, is independent of a traffic characterization, a server employing a scheduling algorithm in GR can provide various service guarantees to flows conforming to any traffic specification. To enable a network of GR servers to provide similar guarantees, in what follows, we present a method for deriving the delay guarantee of a network of GR servers.

2.3 A Network of GR Servers

To derive the delay guarantee for a network of servers, each of which employs a scheduling algorithm in GR, consider flow f that is serviced by K servers. Let the i^{th} server on the path be denoted by i. Then, the network guarantees that packet p_f^j will depart from the network by $GRC^K(p_f^j, r_f^{j,K}) + \beta^K$. This delay guarantee depends on the arrival process at the K^{th} server, i.e., $A^K(p_f^j)$. Though $A^K(p_f^j)$ depends on the arrival process of a flow, i.e., $A^1(p_f^j)$, due to the variability in the delay experienced by the packets of a flow, it may not always be possible to determine relationship between $A^K(p_f^j)$ and $A^1(p_f^j)$. Since, $A^1(p_f^j)$, and not $A^K(p_f^j)$, is always known, it is desirable to characterize the delay guarantee of a network of servers such that it is determined by $A^1(p_f^j)$.

Observe that $GRC^K(p_f^j, r_f^{j,K})$ depends on $A^K(p_f^j)$, which in turn depends on $GRC^{K-1}(p_f^j, r_f^{j,K-1})$. Applying this argument recursively, $GRC^K(p_f^j, r_f^{j,K-1})$ can be related to $GRC^1(p_f^j, r_f^{j,1})$. Consequently, the delay guarantee of a network of servers can be characterized based on $GRC^1(p_f^j, r_f^{j,1})$ which is completely determined by $A^1(p_f^j)$ (i.e., the arrival process of a flow), and the rate assigned to the packets. This enables a network of servers, as in the case of a single server, to provide service guarantee to flows conforming to any specification.

To derive the delay guarantee of a network of servers, we will first relate the guaranteed rate clock value of a packet at two adjacent servers. A similar approach of analyzing two adjacent servers to analyze end-to-end behaviour of a network of servers has been used in [8, 42]. Henceforth we will always refer to a single flow f and hence, for ease of presentation, drop the subscript f from all the variables.

2.3.1 Two Server Case

In large networks, due to the variability in load at different servers, different rates may be allocated to packets at different servers. Since throughput of a network for a flow is governed by throughput of the bottleneck server, instead of relating $GRC^{i+1}(p^j, r^{j,i+1})$ and $GRC^i(p^j, r^{j,i})$, we will establish a relationship between $GRC^{i+1}(p^j, \widehat{r}^{j,i})$ and $GRC^i(p^j, \widehat{r}^{j,i})$ where $GRC^i(p^j, \widehat{r}^{j,i})$ denotes the guaranteed rate clock value computed using $\widehat{r}^{k,i}; \quad k \leq j$, and $\widehat{r}^{j,i}$ represents the bottleneck rate for p^j which is defined as[4]:

$$\widehat{r}^{j,i} \leq \min\{r^{j,i}, r^{j,i+1}\}$$

Observe that this definition captures the scenario where different servers may be the bottleneck server for different packets of the same flow. Although we relate the guaranteed rate clock values computed using bottleneck rate, our analysis will demonstrate that allocating different rates at different servers leads to smaller end-to-end delay than allocating bottleneck rate at each server.

We would find the following inequality useful in establishing the relationship between guaranteed rate clock values at adjacent nodes. Since the guaranteed rate clock value of a packet computed using a smaller rate is larger, $GRC^i(p^j, r^{j,i}) \leq GRC^i(p^j, \widehat{r}^{j,i})$. Hence,

$$\begin{aligned} GRC^i(p^j, r^{j,i}) &= \max\{A^i(p^j), GRC^i(p^{j-1}, r^{j-1,i})\} + \frac{l^j}{r^{j,i}} \quad j \geq 1 \\ &\leq \max\{A^i(p^j), GRC^i(p^{j-1}, \widehat{r}^{j-1,i})\} + \frac{l^j}{r^{j,i}} \quad j \geq 1 \qquad (2.24) \end{aligned}$$

[4] To facilitate the proof of multiple server case, we have chosen an inequality, rather than an equality.

In heterogeneous networks, the data transmission unit may vary and hence packet fragmentation and reassembly may occur. Such a scenario, for example, would occur in an internetwork consisting of ATM, ethernet, and FDDI subnetworks. The relationship between $GRC^{i+1}(p_f^j, r_f^{j,i+1})$ and $GRC^i(p_f^j, r_f^{j,i})$ depends on the occurrence of such a scenario. Hence, in the following subsections, we first establish the relationship when packet fragmentation and reassembly do not occur and then consider their effects.

2.3.1.1 No Fragmentation and/or Reassembly

Theorem 2.6 *If the scheduling algorithm at server i belongs to GR for flow f , then*

$$GRC^{i+1}(p^j, \hat{r}^{j,i}) \leq GRC^i(p^j, \hat{r}^{j,i}) + \max_{k \in [1..j]} \frac{l^k}{r^{k,i}} + \alpha^i \quad j \geq 1 \tag{2.25}$$

where $\alpha^i = \beta^i + \tau^{i,i+1}$ and $\tau^{i,i+1}$ is an upper bound on the propagation delay between servers i and $i+1$.

Proof: The proof is by induction on j.

Base Case: $j = 1$

$$GRC^{i+1}(p^1, \hat{r}^{1,i}) = A^{i+1}(p^1) + \frac{l^1}{\hat{r}^{1,i}} \tag{2.26}$$

Since scheduling algorithm at server i belongs to GR for flow f, $A^{i+1}(p^1) \leq GRC^i(p^1, r^{1,i}) + \alpha^i$. From (2.26) and (2.24) we get:

$$\begin{aligned}
GRC^{i+1}(p^1, \hat{r}^{1,i}) &\leq \max\{A^i(p^1), GRC^i(p^0, \hat{r}^{0,i})\} + \frac{l^1}{r^{1,i}} + \frac{l^1}{\hat{r}^{1,i}} + \alpha^i \\
&\leq \left(\max\{A^i(p^1), GRC^i(p^0, \hat{r}^{0,i})\} + \frac{l^1}{\hat{r}^{1,i}}\right) + \frac{l^1}{r^{1,i}} + \alpha^i \\
&\leq GRC^i(p^1, \hat{r}^{1,i}) + \max_{k \in [1..1]} \frac{l^k}{r^{k,i}} + \alpha^i
\end{aligned}$$

Therefore (2.25) holds for $j = 1$

Induction Hypothesis: Assume (2.25) holds for $1 \leq j \leq m$.

Induction: We need to show (2.25) holds for $1 \leq j \leq m+1$. From (2.4) (definition of GRC), we get:

$$GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) = \max\{A^{i+1}(p^{m+1}), GRC^{i+1}(p^m, \hat{r}^{m,i})\} + \frac{l^{m+1}}{\hat{r}^{m+1,i}} \tag{2.27}$$

There are two cases to consider:

1. $A^{i+1}(p^{m+1}) > GRC^{i+1}(p^m, \hat{r}^{m,i})$: Since scheduling algorithm at server i belongs to GR for flow f, we know $A^{i+1}(p^{m+1}) \leq GRC^i(p^{m+1}, r^{m+1,i}) + \alpha^i$. Hence, from (2.27) we get:

$$GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) \leq GRC^i(p^{m+1}, r^{m+1,i}) + \alpha^i + \frac{l^{m+1}}{\hat{r}^{m+1,i}}$$

   Using (2.24) we get:

$$\begin{aligned} GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) &\leq \max\{A^i(p^{m+1}), GRC^i(p^m, \hat{r}^{m,i})\} + \frac{l^{m+1}}{r^{m+1,i}} + \alpha^i + \\ &\quad \frac{l^{m+1}}{\hat{r}^{m+1,i}} \\ &\leq \left(\max\{A^i(p^{m+1}), GRC^i(p^m, \hat{r}^{m,i})\} + \frac{l^{m+1}}{\hat{r}^{m+1,i}}\right) + \\ &\quad \frac{l^{m+1}}{r^{m+1,i}} + \alpha^i \\ &\leq GRC^i(p^{m+1}, \hat{r}^{m+1,i}) + \max_{k \in [1..m+1]} \frac{l^k}{r^{k,i}} + \alpha^i \qquad (2.28) \end{aligned}$$

2. $A^{i+1}(p^{m+1}) \leq GRC^{i+1}(p^m, \hat{r}^{m,i})$: From (2.27) we get:

$$GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) \leq GRC^{i+1}(p^m, \hat{r}^{m,i}) + \frac{l^{m+1}}{\hat{r}^{m+1,i}}$$

   Using induction hypothesis we get,

$$\begin{aligned} GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) &\leq GRC^i(p^m, \hat{r}^{m,i}) + \max_{k \in [1..m]} \frac{l^k}{r^{k,i}} + \alpha^i + \frac{l^{m+1}}{\hat{r}^{m+1,i}} \\ &\leq \left(GRC^i(p^m, \hat{r}^{m,i}) + \frac{l^{m+1}}{\hat{r}^{m+1,i}}\right) + \max_{k \in [1..m]} \frac{l^k}{r^{k,i}} + \alpha^i \\ &\leq GRC^i(p^{m+1}, \hat{r}^{m+1,i}) + \max_{k \in [1..m+1]} \frac{l^k}{r^{k,i}} + \alpha^i \qquad (2.29) \end{aligned}$$

From (2.28), (2.29) and the induction hypothesis, we conclude that (2.25) holds for $1 \leq j \leq m+1$. Hence the theorem follows. ■

Notice that the proof of Theorem 2.6 only requires $A^{i+1}(p_f^j) \leq GRC^i(p^j, r^{j,i}) + \alpha^i$. Hence, it holds even if a server employs a jitter controller, similar to jitter controller employed

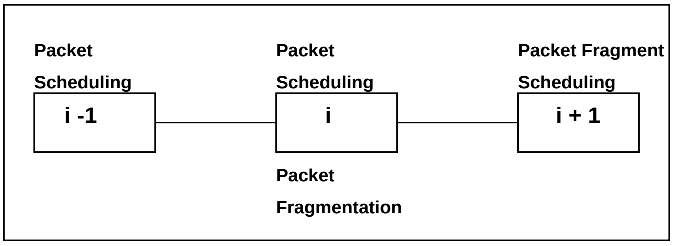


Figure 2.5: Packet scheduling at the switch where fragmentation occurs

by Jitter EDD, to ensure that the arrival time of packet p_f^j is exactly $GRC^i(p^j, r^{j,i}) + \alpha^i$. Also, observe that even though we have related guaranteed clock values computed based on the bottleneck rate, the denominator in the second term of the right hand side of (2.25) is $r^{j,i}$ instead of $\widehat{r}^{j,i}$. This would enable us to derive tighter end-to-end delay bounds in section 2.4.

2.3.1.2 Packet Fragmentation

Consider an internetwork of ethernet, ATM, and FDDI subnetworks. In such an internetwork, a server may receive packet larger than it can transmit. Let server i receive a packet larger than it can transmit. As we had mentioned in Section 2.2.4.1, server i may schedule packets or packet fragments. Let us consider the two cases:

- If server i schedules packets (see Figure (2.5)), it may or may not allow packet transmission to be preempted after transmission of every packet fragment. In either case, $GRC^i(p^j, \widehat{r}^{j,i-1})$ can be related to $GRC^{i-1}(p^j, \widehat{r}^{j,i-1})$ using Theorem 2.6. However, we need to relate the guaranteed rate clock value of packet at server i and $i+1$.

- If server i schedules packet fragments(see Figure (2.6)), then guaranteed rate clock values of packet fragments at server i and $i+1$ can be related using Theorem 2.6. However, we need to relate the guaranteed rate clock value of packet at server $i-1$ and i.

From the two cases, we can infer that, in the presence of fragmentation, we need to relate the guaranteed rate clock values of packets between two adjacent servers that have different scheduling units, i.e, packets and packet fragments. Let packet p^j be fragmented into $\theta(j)$ fragments, $p^{j,k}$ denote the k^{th} fragment of p^j and let $l^{j,k}$ denote the length of packet fragment

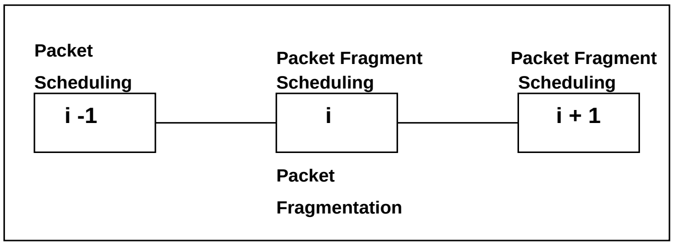


Figure 2.6: Packet fragment scheduling at the switch where fragmentation occurs

$p^{j,k}$. Since the delay guarantee of a packet is determined by the delay guarantee of the last fragment of a packet, we relate the guaranteed rate clock value of the last fragment of a packet at adjacent servers in Theorem 2.7.

Theorem 2.7 *If the scheduling algorithm at server i schedules packets and belongs to GR for flow f, and server $i+1$ schedules packet fragments, then*

$$GRC^{i+1}(p^{j,\theta(j)}, \hat{r}^{j,i}) \leq GRC^{i}(p^{j}, \hat{r}^{j,i}) + \max_{k \in [1..j]} \frac{l^k}{r^{k,i}} + \alpha^i \quad j \geq 1 \tag{2.30}$$

where $\alpha^i = \beta^i + \tau^{i,i+1}$ and $\tau^{i,i+1}$ is an upper bound on the propagation delay between servers i and $i+1$.

Proof: The proof is by induction on j.

Base Case: $j = 1$. Let $p^{1,o}$ be the last packet fragment to have arrived before $p^{1,\theta(1)}$ such that $A^{i+1}(p^{1,o}) > GRC^{i+1}(p^{1,o-1}, \hat{r}^{1,i})$. Then, from (2.4) (definition of GRC) we get:

$$GRC^{i+1}(p^{1,\theta(1)}, \hat{r}^{1,i}) = A^{i+1}(p^{1,o}) + \sum_{k=o}^{k=\theta(1)} \frac{l^{1,k}}{\hat{r}^{1,i}}$$

Since scheduling algorithm at server i belongs to GR for flow f, $A^{i+1}(p^{1,o}) \leq GRC^{i}(p^1, r^{1,i}) + \alpha^i$. Hence

$$\begin{aligned} GRC^{i+1}(p^{1,\theta(1)}, \hat{r}^{1,i}) &\leq GRC^{i}(p^1, r^{1,i}) + \sum_{k=o}^{k=\theta(1)} \frac{l^{1,k}}{\hat{r}^{1,i}} + \alpha^i \\ &\leq A^{i}(p^1) + \frac{l^1}{r^{1,i}} + \sum_{k=o}^{k=\theta(1)} \frac{l^{1,k}}{\hat{r}^{1,i}} + \alpha^i \end{aligned}$$

Since $\sum_{k=o}^{k=\theta(1)} \frac{l^{1,k}}{\widehat{r}^{1,i}} \leq \frac{l^1}{\widehat{r}^{1,i}}$, we get

$$\begin{aligned} GRC^{i+1}(p^{1,\theta(1)}, \widehat{r}^{1,i}) &\leq A^i(p^1) + \frac{l^1}{\widehat{r}^{1,i}} + \frac{l^1}{r^{1,i}} + \alpha^i \\ &\leq GRC^i(p^1, \widehat{r}^{1,i}) + \frac{l^1}{r^{1,i}} + \alpha^i \\ &\leq GRC^i(p^1, \widehat{r}^{1,i}) + \max_{k \in [1..1]} \frac{l^1}{r^{1,i}} + \alpha^i \end{aligned}$$

Therefore (2.30) holds for $j = 1$

Induction Hypothesis: Assume (2.30) holds for $1 \leq j \leq m$.

Induction: We need to show (2.30) holds for $1 \leq j \leq m + 1$.

$$GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) = \max\{A^{i+1}(p^{m+1,\theta(m+1)}), GRC^{i+1}(p^{m+1,\theta(m+1)-1}, \widehat{r}^{m+1,i})\} + \frac{l^{m+1,\theta(m+1)}}{\widehat{r}^{m+1,i}} \tag{2.31}$$

Thus, there are two cases to consider:

1. $A^{i+1}(p^{m+1,\theta(m+1)}) > GRC^{i+1}(p^{m+1,\theta(m+1)-1}, \widehat{r}^{m+1,i})$: From (2.31) we get:

$$GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) \leq A^{i+1}(p^{m+1,\theta(m+1)}) + \frac{l^{m+1,\theta(m+1)}}{\widehat{r}^{m+1,i}}$$

Since scheduling algorithm at server i belongs to GR for flow f, $A^{i+1}(p^{m+1,\theta(m+1)})$ $\leq GRC^i(p^{m+1}, r^{m+1,i}) + \alpha^i$.

$$GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) \leq GRC^i(p^{m+1}, r^{m+1,i}) + \frac{l^{m+1,\theta(m+1)}}{\widehat{r}^{m+1,i}} + \alpha^i$$

Hence, using (2.24) we get:

$$\begin{aligned} GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i} &\leq \max\{A^i(p^m), GRC^i(p^m, \widehat{r}^{m,i})\} + \frac{l^{m+1}}{r^{m+1,i}} + \\ &\quad \frac{l^{m+1,\theta(m+1)}}{\widehat{r}^{m+1,i}} + \alpha^i \\ &\leq \max\{A^i(p^m), GRC^i(p^m, \widehat{r}^{m,i})\} + \frac{l^{m+1,\theta(m+1)}}{\widehat{r}^{m+1,i}} \\ &\quad + \frac{l^{m+1}}{r^{m+1,i}} + \alpha^i \\ &\leq GRC^i(p^{m+1}, \widehat{r}^{m+1,i}) + \frac{l^{m+1}}{r^{m+1,i}} + \alpha^i \\ &\leq GRC^i(p^{m+1}, \widehat{r}^{m+1,i}) + \max_{k \in [1..m+1]} \frac{l^k}{r^{k,i}} + \alpha^i \end{aligned} \tag{2.32}$$

2. $A^{i+1}(p^{m+1,\theta(m+1)}) \leq GRC^{i+1}(p^{m+1,\theta(m+1)-1}, \widehat{r}^{m+1,i})$: From (2.31) we get:

$$GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) \leq GRC^{i+1}(p^{m+1,\theta(m+1)-1}, \widehat{r}^{m+1,i}) + \frac{l^{m+1,\theta(m+1)}}{\widehat{r}^{m+1,i}} \tag{2.33}$$

Let $p^{m+1,0}$ refer to $p^{m,\theta(m)}$. There are two cases to consider:

(a) $\forall k \in [1..\theta(m+1)] : A^{i+1}(p^{m+1,k}) \leq GRC^{i+1}(p^{m+1,k-1}, \widehat{r}^{m+1,i})$: From (2.33) we get:

$$GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) \leq GRC^{i+1}(p^{m,\theta(m)}, \widehat{r}^{m,i}) + \sum_{k=1}^{k=\theta(m+1)} \frac{l^{m+1,k}}{\widehat{r}^{m+1,i}}$$

Using induction hypothesis, we get:

$$\begin{aligned} GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) &\leq GRC^{i}(p^{m,\theta(m)}, \widehat{r}^{m,i}) + \max_{k\in[1..m]} \frac{l^k}{r^{k,i}} + \\ &\quad \alpha^i + \frac{l^{m+1}}{\widehat{r}^{m+1,i}} \\ &\leq \left(GRC^{i}(p^{m,\theta(m)}, \widehat{r}^{m,i}) + \frac{l^{m+1}}{\widehat{r}^{m+1,i}}\right) + \\ &\quad \max_{k\in[1..m]} \frac{l^k}{r^{k,i}} + \alpha^i \\ &\leq GRC^{i}(p^{m+1}, \widehat{r}^{m+1,i}) + \max_{k\in[1..m+1]} \frac{l^k}{r^{k,i}} + \\ &\quad \alpha^i \end{aligned} \tag{2.34}$$

(b) $\exists k \in [1..\theta(m+1)] : A^{i+1}(p^{m+1,k}) > GRC^{i+1}(p^{m+1,k-1}, \widehat{r}^{m+1,i})$: Let o be the greatest integer less than or equal to $\theta(m+1)$ such that $A^{i+1}(p^{m+1,o}) > GRC^{i+1}(p^{m+1,o-1}, \widehat{r}^{m+1,i})$. Then, from (2.33) we get:

$$\begin{aligned} GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) &\leq A^{i+1}(p^{m+1,o}) + \sum_{k=o}^{k=\theta(m+1)} \frac{l^{m+1,k}}{\widehat{r}^{m+1,i}} \\ &\leq A^{i+1}(p^{m+1,o}) + \frac{l^{m+1}}{\widehat{r}^{m+1,i}} \end{aligned}$$

Since scheduling algorithm at server i belongs to GR for flow f, we get:

$$GRC^{i+1}(p^{m+1,\theta(m+1)}, \widehat{r}^{m+1,i}) \leq \max\{A^i(p^{m+1}), GRC^i(p^m, \widehat{r}^{m,i})\} + \frac{l^{m+1}}{r^{m+1,i}}$$

$$
\begin{aligned}
&+\alpha^i + \frac{l^{m+1}}{\widehat{r}^{m+1,i}} \\
\leq\;& \max\{A^i(p^{m+1}), GRC^i(p^m, \widehat{r}^{m,i})\} + \frac{l^{m+1}}{\widehat{r}^{m+1,i}} \\
&+\frac{l^{m+1}}{r^{m+1,i}} + \alpha^i \\
\leq\;& GRC^i(p^{m+1}, \widehat{r}^{m+1,i}) + \frac{l^{m+1}}{r^{m+1,i}} + \alpha^i \\
\leq\;& GRC^i(p^{m+1}, \widehat{r}^{m+1,i}) + \\
&\max_{k\in[1..m+1]} \frac{l^k}{r^{k,i}} + \alpha^i
\end{aligned}
\tag{2.35}
$$

From (2.32), (2.34), (2.35), and the induction hypothesis, we conclude that (2.30) holds for $1 \leq j \leq m+1$. Hence the theorem follows. ■

2.3.1.3 Packet Reassembly

Let server $i+1$ perform reassembly of packet fragments. Theorem 2.8 relates the guaranteed rate clock value of the last fragment of a packet at server i and $i+1$.

Theorem 2.8 *If the scheduling algorithm at server i belongs to GR for flow f and server $i+1$ performs reassembly, then*

$$GRC^{i+1}(p^j, \widehat{r}^{j,i}) \leq GRC^i(p^{j,\theta(j)}, \widehat{r}^{j,i}) + \max_{k\in[1..j]} \frac{l^k}{\widehat{r}^{k,i}} + \alpha^i \quad j \geq 1 \tag{2.36}$$

where $\alpha^i = \beta^i + \tau^{i,i+1}$ and $\tau^{i,i+1}$ is an upper bound on the propagation delay between servers i and $i+1$.

Proof: The proof is by induction on j.

Base Case: $j = 1$

$$GRC^{i+1}(p^1, \widehat{r}^{1,i}) = A^{i+1}(p^1) + \frac{l^1}{\widehat{r}^{1,i}}$$

Since scheduling algorithm at server i belongs to GR for flow f, $A^{i+1}(p^1) \leq GRC^i(p^{1,\theta(1)}, \widehat{r}^{1,i}) + \alpha^i$. Hence,

$$
\begin{aligned}
GRC^{i+1}(p^1, \widehat{r}^{1,i}) \;&\leq\; GRC^i(p^{1,\theta(1)}, \widehat{r}^{1,i}) + \frac{l^1}{\widehat{r}^{1,i}} + \alpha^i \\
&\leq\; GRC^i(p^{1,\theta(1)}, \widehat{r}^{1,i}) + \max_{k\in[1..1]} \frac{l^k}{\widehat{r}^{1,i}} + \alpha^i
\end{aligned}
$$

Therefore (2.36) holds for $j = 1$

Induction Hypothesis: Assume (2.36) holds for $1 \leq j \leq m$.

Induction: We need to show (2.36) holds for $1 \leq j \leq m + 1$.

$$GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) = \max\{A^{i+1}(p^{m+1}), GRC^{i+1}(p^m, \hat{r}^{m,i})\} + \frac{l^{m+1}}{\hat{r}^{m+1,i}} \tag{2.37}$$

There are two cases to consider:

1. $A^{i+1}(p^{m+1}) > GRC^{i+1}(p^m, \hat{r}^{m,i})$: From (2.37) we get:

$$GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) \leq A^{i+1}(p^{m+1}) + \frac{l^{m+1}}{\hat{r}^{m+1,i}}$$

Since scheduling algorithm at server i belongs to GR for flow f, we get:

$$\begin{aligned} GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) &\leq GRC^i(p^{m+1,\theta(m+1)}, \hat{r}^{m+1,i}) + \alpha^i + \frac{l^{m+1}}{\hat{r}^{m+1,i}} \\ &\leq GRC^i(p^{m+1,\theta(m+1)}, \hat{r}^{m+1,i}) + \max_{k\in[1..m+1]} \frac{l^k}{\hat{r}^{k,i}} \\ &\quad +\alpha^i \end{aligned} \tag{2.38}$$

2. $A^{i+1}(p^{m+1}) \leq GRC^{i+1}(p^m, \hat{r}^{m,i})$: From (2.37) we get:

$$GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) \leq GRC^{i+1}(p^m, \hat{r}^{m,i}) + \frac{l^{m+1}}{\hat{r}^{m+1,i}}$$

Using induction hypothesis we get,

$$\begin{aligned} GRC^{i+1}(p^{m+1}, \hat{r}^{m+1,i}) &\leq GRC^i(p^{m,\theta(m)}, \hat{r}^{m,i}) + \max_{k\in[1..m]} \frac{l^k}{\hat{r}^{k,i}} + \frac{l^{m+1}}{\hat{r}^{m+1,i}} + \alpha^i \\ &\leq \left(GRC^i(p^{m,\theta(m)}, \hat{r}^{m,i}) + \frac{l^{m+1}}{\hat{r}^{m+1,i}}\right) + \\ &\quad \max_{k\in[1..m+1]} \frac{l^k}{\hat{r}^{k,i}} + \alpha^i \\ &\leq GRC^i(p^{m+1,\theta(m+1)}, \hat{r}^{m+1,i}) + \\ &\quad \max_{k\in[1..m+1]} \frac{l^k}{\hat{r}^{k,i}} + \alpha^i \end{aligned} \tag{2.39}$$

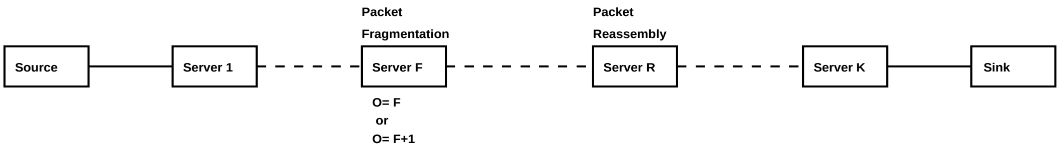


Figure 2.7: The path configuration

From (2.38), (2.39), and the induction hypothesis, we conclude that (2.36) holds for $1 \leq j \leq m+1$. Hence the theorem follows. ■

Observe that the second term in the right hand side of (2.36) is $\max_{k\in[1..j]} \frac{l^k}{\widehat{r}^{k,i}}$ which is different from the corresponding term, $\max_{k\in[1..j]} \frac{l^k}{r^{k,i}}$, in (2.25) and (2.30).

2.3.2 Multiple Server Case

In Theorems 2.6, 2.7, and 2.8, we have related the guaranteed rate clock values of a packet at adjacent servers under various scenarios. These relationships can be employed to relate the guaranteed rate clock value of the packet at K^{th} server to that at the first server. Clearly, such a relationship depends on the configuration of the path of a flow. We illustrate the relationship between the guaranteed rate clock value of the packet at the K^{th} server and its value at the first server for the path configuration shown in Figure 2.7. In Figure 2.7 packet fragmentation occurs at server F and packet reassembly occurs at server R. Let server O be the first server on the path of the flow that schedules packet fragments. If server F schedules packets, then the next server schedules packet fragments and hence $O = F+1$; otherwise $O = F$.

Theorem 2.9 *If the scheduling algorithm at each of the servers on the path of a flow belongs to GR for flow f, then*

$$\begin{aligned} GRC^K(p^j, \widehat{r}^j) \quad \leq \quad & GRC^1(p^j, \widehat{r}^j) + \sum_{i=1}^{i=O-1} \max_{n\in[1..j]} \frac{l^n}{r^{n,i}} + \sum_{i=O}^{i=R-2} \max_{n\in[1..j]} \frac{\hat{l}^n}{r^{n,i}} \\ & + \max_{n\in[1..j]} \frac{l^n}{\widehat{r}^n} + \sum_{i=R}^{i=K-1} \max_{n\in[1..j]} \frac{l^n}{r^{n,i}} + \sum_{n=1}^{n=K-1} \alpha^n \end{aligned}$$

where $\alpha^n = \beta^n + \tau^{n,n+1}$, K is the number of servers on the path of the flow, $\hat{l}^n$ is the length of the biggest fragment of p^n, and $\widehat{r}^j$ is the bottleneck rate for packet p^j, i.e., $\widehat{r}^j = \min_{i\in[1..K]} r^{j,i}$.

Proof: Since $\hat{r}^j \leq \hat{r}^{j,i}$ for each server on the path of the flow and the scheduling algorithm at each server belongs to GR, Theorem 2.6 can be employed. By repeated application of Theorem 2.6:

$$GRC^K(p^j, \hat{r}^j) \leq GRC^R(p^j, \hat{r}^j) + \sum_{i=R}^{i=K-1} \max_{n \in [1..j]} \frac{l^n}{r^{n,i}} + \sum_{n=R}^{n=K-1} \alpha^n$$

Since server R does packet reassembly, using Theorem 2.8 we get:

$$GRC^R(p^j, \hat{r}^j) \leq GRC^{R-1}(p^{j,\theta(j)}, \hat{r}^j) + \max_{n \in [1..j]} \frac{l^n}{\hat{r}^n} + \alpha^{R-1}$$

Using Theorem 2.6 again, we get:

$$GRC^{R-1}(p^{j,\theta(j)}, \hat{r}^j) \leq GRC^O(p^{j,\theta(j)}, \hat{r}^j) + \sum_{i=O}^{i=R-2} \max_{n \in [1..j]} \frac{\hat{l}^n}{r^{n,i}} + \sum_{n=O}^{n=R-2} \alpha^n$$

Since server O schedules packet fragments and server $O-1$ schedules packets, from Theorem 2.7 we get:

$$GRC^O(p^{j,\theta(j)}, \hat{r}^j) \leq GRC^{O-1}(p^j, \hat{r}^j) + \max_{n \in [1..j]} \frac{l^n}{r^{n,O-1}} + \alpha^{O-1}$$

Using Theorem 2.6 again , we get:

$$GRC^{O-1}(p^j, \hat{r}^j) \leq GRC^1(p^j, \hat{r}^j) + \sum_{i=1}^{i=O-2} \max_{n \in [1..j]} \frac{l^n}{r^{n,i}} + \sum_{n=1}^{n=O-2} \alpha^n$$

The theorem follows from the above steps.

■

It can be similarly shown that if packet fragmentation does not occur, then:

$$GRC^K(p^j, \hat{r}^j) \leq GRC^1(p^j, \hat{r}^j) + \sum_{i=1}^{i=K-1} \max_{n \in [1..j]} \frac{l^n}{r^{n,i}} + \sum_{n=1}^{n=K-1} \alpha^n \tag{2.40}$$

Observe that even though we have derived the delay guarantee based on the bottleneck rate, the denominator in $\sum_{i=1}^{i=K-1} \max_{n \in [1..j]} \frac{l^n}{r^{n,i}}$ is the actual rate allocation at each server. Clearly, this term is smaller than the similar term that would have been derived assuming bottleneck rate allocation at each server.

The delay guarantee of a network of GR servers, just like a single GR server, is based on the guaranteed rate clock value of a packet. Thus, just like a single server, a network can provide various service guarantees to flows conforming to any specification. For example, as we illustrate in the next section, the delay guarantee can be employed to guarantee an upper bound on end-to-end delay of packets of a flow.

2.4 End-to-End Delay Bound

To determine an upper bound on the end-to-end delay of packets of a flow, consider a flow which is served by K servers. Also, let server 0 be the source and server $K+1$ be the destination. Let d^j be the delay experienced by the packets of a flow. Since server K guarantees that packet p^j will be transmitted by time $GRC^K(p^j, r^{j,K}) + \beta^K$ and the packet arrives at the first node at time $A^1(p^j)$, we get:

$$d^j \leq GRC^K(p^j, r^{j,K}) + \alpha^K - A^1(p^j)$$

where $\alpha^K = \beta^K + \tau^{K,K+1}$. Observe that $GRC^K(p^j, \widehat{r^j}) - GRC^K(p^j, r^{j,K}) \geq \frac{l^j}{\widehat{r^j}} - \frac{l^j}{r^{j,K}}$. Hence,

$$d^j \leq GRC^K(p^j, \widehat{r^j}) - \left(\frac{l^j}{\widehat{r^j}} - \frac{l^j}{r^{j,K}}\right) + \alpha^K - A^1(p^j)$$

If each server on the path of the flow employs a scheduling algorithm in GR, then given the path configuration, $GRC^K(p^j, \widehat{r^j})$ can be related to $GRC^1(p^j, \widehat{r^j})$. For instance, when packet fragmentation and reassembly does not occur along the path, then using (2.40) we get:

$$d^j \leq \left(GRC^1(p^j, \widehat{r^j}) - A^1(p^j)\right) + \left(\sum_{i=1}^{i=K-1} \max_{n\in[1..j]} \frac{l^n}{r^{n,i}} - \left(\frac{l^j}{\widehat{r^j}} - \frac{l^j}{r^{j,K}}\right)\right) + \left(\sum_{n=1}^{n=K} \alpha^n\right) \quad (2.41)$$

Hence, the end-to-end delay of a packet consists of three components:

- $\sum_{n=1}^{n=K} \alpha^n$: Since $\alpha^n = \beta^n + \tau^{n,n+1}$, this term is completely characterized by the scheduling algorithms and the propagation delay in the network.

- $\sum_{i=1}^{i=K-1} \max_{n\in[1..j]} \frac{l^n}{r^{n,i}} - (\frac{l^j}{\widehat{r^j}} - \frac{l^j}{r^{j,K}})$: This term depends on the length of the packets transmitted by a source and the rate allocated to it at various servers. Hence, this term is known if the length of the packets transmitted by a source and the rate assignments are known.

- $GRC^1(p^j, \widehat{r}^j) - A^1(p^j)$: This is the only term that depends on arrival process characteristics of a flow. This term is the queuing delay experienced by a packet at a single server with variable capacity; the capacity being the bottleneck rate for the packet in service. If fixed rate is assigned to all the packets of a flow, then it is the queuing delay at server with capacity $\widehat{r}$ where $\widehat{r}$ is the bottleneck rate for the flow. Hence, the network can be abstracted as a single server (with either fixed or variable capacity). Consequently the problem of determining end-to-end delay is reduced to the problem of determining delay at a single node. If a flow is characterized using a deterministic traffic specification, then we can employ single server queuing analysis to determine an upper bound on the delay of packets of a flow. If it is characterized using a statistical specification, queuing analysis can be employed to determine a bound on distribution of delays experienced by packets of a flow. We now illustrate the delay bound computation for flows conforming to leaky bucket which is a deterministic specification.

  Leaky bucket is used to specify the traffic characteristics of sources that transmit at an average rate but may deviate from it for short durations [42]. To define leaky bucket specification, let $AP(t_1, t_2)$ be a function that denotes the number of bits that arrive in the interval $[t_1, t_2]$ (The bits of a packet are considered to have arrived only after the arrival of the last bit of the packet.). Then a flow conforms to leaky bucket with burst size σ and average rate r, if for all time intervals $[t_1, t_2]$:

$$AP(t_1, t_2) \leq \sigma + r(t_2 - t_1) \tag{2.42}$$

  Now, consider a flow that conforms to (σ, r) leaky bucket. Let rate $\widehat{r} \geq r$ be assigned to all the packets of the flow. To determine $GRC^1(p^j, \widehat{r}) - A^1(p^j)$, let $t_1 = A^1(p^k)$ and $t_2 = A^1(p^j)$ and $k \leq j$ is the largest integer such that $A^1(p^j) = \max\{A^1(p^j), GRC^1(p^{j-1}, \widehat{r}\}$. Then,

$$\begin{aligned} GRC^1(p^j, \widehat{r}^j) &= t_1 + \frac{\sum_{n=k}^{n=j} l^n}{\widehat{r}} \\ &= t_1 + \frac{AP(t_1, t_2)}{\widehat{r}} \end{aligned} \tag{2.43}$$

Since flow f conforms to leaky bucket specification, we get

$$AP(t_1, t_2) \leq \sigma + r(t_2 - t_1)$$

Hence, from (2.43) we get,

$$GRC^1(p_f^j) \leq t_1 + \frac{\sigma + r(t_2 - t_1)}{\widehat{r}}$$

Since $\widehat{r} \geq r$, we get:

$$GRC^1(p_f^j) \leq \frac{\sigma}{\widehat{r}} + t_2$$

which implies

$$GRC^1(p^j) - t_2 \leq \frac{\sigma}{\widehat{r}} \tag{2.44}$$

Since $t_2 = A^1(p^j)$, substituting (2.44) in (2.41) gives an upper bound on the end-to-end delay.

The above method not only determines an upper bound on end-to-end delay for any source specification in an internetwork in a conceptually simple manner but also leads to tighter results. Observe that if $r^{j,K} = \widehat{r}^j$ (assumed for ease of exposition) then the only variable term in (2.41), that depends on network and not flow characteristics is:

$$\sum_{i=1}^{i=K-1} \max_{n \in [1..j]} \frac{l^n}{r^{n,i}} \tag{2.45}$$

This term is smaller than other similar analysis in the literature in several ways. We illustrate the subtleties in (2.45) by comparing it with the seminal analysis presented in [41] for Rate Proportional Processor Sharing (RPPS) rate assignment of WFQ networks. The delay bound in[41] for a flow that conforms to leaky bucket with parameters (σ, r) and has minimum rate $\widehat{r} \geq r$ assigned to the packets ($\widehat{r} \geq r$) is:

$$d^j \leq \frac{\sigma}{\widehat{r}} + 2 \sum_{i=1}^{i=K-1} \frac{l^{max}}{\widehat{r}} + \sum_{n=1}^{n=K} \alpha^n$$

where l^{max} is the maximum length of a packet of the flow. Hence, the term corresponding to (2.45) is:

$$2 \sum_{i=1}^{i=K-1} \frac{l^{max}}{\widehat{r}} \tag{2.46}$$

(2.46) is larger than (2.45) in several ways:

- The factor 2 makes (2.46) significantly larger than (2.45).

- Even when different rates may be allocated at different servers, the denominator in (2.46) is $\widehat{r}$. In contrast, the denominator in (2.45) is $r^{n,i}$ where $r^{n,i} \geq \widehat{r}$. To illustrate the differences numerically, consider a flow with packets of length 100 bytes that is being served by two servers such that the rate allocations at server 1 and 2 are 64Kb/s and 32Kb/s, respectively. Then, after neglecting factor 2, (2.46) evaluates to 24.4 ms which is larger than the 12.2 ms computed from (2.45).

- The numerator in (2.46) is l^{max} which may be larger than the numerator, $\max_{n \in [1..j]} l^n$ (assuming a constant rate allocation for packets of a flow), in (2.45).

  This difference can be made larger by defining a network *busy period*. For ease of exposition, let no fragmentation and reassembly occur. Then, a network is considered busy for flow f at time t if

  $$GRC^1(p^j, \widehat{r}^j) + \sum_{i=1}^{i=K-1} \max_{n \in [1..j]} \frac{l^n}{r^{n,i}} + \sum_{n=1}^{n=K-1} \alpha^n \leq t$$

  where p^j is the last packet to have arrived before t. Let the packets be renumbered such that packet p^j is the j^{th} packet to arrive in a network busy period. From the proofs of Theorems 2.6, 2.7 and 2.8, we know that Theorem 2.9 still continues to hold. Consequently, in (2.45), the maximum is over a subsequence of packets rather than all the previous packets.

  To illustrate the difference numerically, consider a flow that has 5 servers on the path and has 1Mb/s rate allocated to its packets. Consider two busy periods 1 and 2 such that the maximum packet length during the periods is 1000 bytes and 100 bytes, respectively. Then, (2.45) evaluates to 30.5 ms and 3.05 ms for busy periods 1 and 2, respectively. If the maximum packet length the flow ever transmits is 1500 bytes, then (2.46) will always yield 45.7 ms.

- Whereas (2.46) does not quantify the effect of variable rate allocation to packets of a flow, (2.45) does. Furthermore, (2.45) when used in conjunction with the concept of busy period enables derivation of better delay bounds. To illustrate numerically, consider a flow that has 5 servers on the path and transmits 1000 byte packets. Consider two busy periods 1 and 2 in which the rate allocations are 100Kb/s and 1Mb/s, respectively. Then, (2.45) evaluates to 305 ms and 30.5 ms for busy periods 1 and 2, respectively. Without the notion of busy period, the bound would have been 305 ms in both the busy periods.

2.5 Architectural Principles for GR Networks

The class of GR scheduling algorithms has several desirable properties that simplify the design of a network that provides deterministic QoS. They also enable it to support application with different characteristics and requirements. We now present a few important architectural principles, based on the properties of GR, for the design of networks employing scheduling algorithms in GR class.

- *A source is not required to specify the characteristics of the traffic to the network*: It has been argued in the literature that a network provides QoS guarantees such as packet delay and loss based on the traffic specification of a source. However, as is evident from Theorem 2.9, a network employing scheduling algorithms in GR can provide a delay guarantee without requiring the source to specify the characteristics of the traffic. Furthermore, by keeping track of guaranteed rate clock values associated with its flow at the first server, a source can determine end-to-end delays without specifying the characteristics of the traffic to the network. Hence, a network can provide QoS guarantees without requiring a traffic specification. Such a network architecture is desirable as a source may not have a good characterization of the traffic or the characterization may not be known a priori. Moreover, even if the characterization is known, it may not conform to the set of characterizations supported by a network.

- *Policing of traffic is not required*: Observe that the guaranteed rate clock values of a packet of a flow are independent of the behaviour of the other flows in the network. Consequently, the guarantees offered by a network employing scheduling algorithms in

GR are independent of the behaviour of other flows, i.e., the network provides isolation between sources. Hence, unlike conventional network architectures where source traffic is policed to ensure that it conforms to the advertised characteristics [13], policing of traffic is not required in GR networks.

- *A source should be able to request buffer reservation in the network*: If a source does not specify the characteristics of the traffic to the network, then a network may not know the buffer space that should be reserved for a flow in order to provide the desired packet loss. A network, rather than requiring a source traffic specification and then deriving the buffer requirements, should allow a source to explicitly request buffer space [5]. This would keep the design of the network simple and enable it to support applications with different characteristics and requirements. Though conceptually simple, observe that due to increased burstiness of a flow at downstream servers, buffer requirement is different at different servers on the path of a flow. However, from the end-to-end delay analysis, we know that the buffer requirement at a server can be partitioned into two components:

  - Buffer required if the network was abstracted as a single server with the bottleneck rate: This component is dependent on the behaviour of a flow and is known to the source.
  - Buffer required due to the increase in burstiness of a flow at downstream servers: This component at a server is dependent on the rate reservation at upstream servers and can be determined without traffic specification (an example method for determining this component is presented in [48]).

  Whereas a source should be able to request the buffer required due to the first component, a network should reserve the additional buffers required, due to the second component, at each server.

- *A source should be able to request a network to employ rate controllers*: As Theorem 2.5 illustrates, in GR networks the worst case delay of packet does not change when one or

[5] In reservation setup protocols like RSVP, sinks initiate the reservation [63]. However, for the purpose of this discussion, this distinction does not make any difference.

more servers on the path of a flow employ a rate controller. Though rate controllers do not change the worst case end-to-end delay, they increase the average case delay while reducing the buffer requirement in the network. Instead of a network deciding the tradeoff between the buffer requirement and the average case delay for all the sources, a source should be provided with the flexibility of deciding the tradeoff. A source would have the flexibility if it can request a network to employ a rate controller.

- *GR networks should provide good average case performance as well*: As is evident from the end-to-end delay bound determination method, a GR network which is designed to provide only worst case guarantee, can do so by buffering all the data at the network periphery and serving it at the rate desired by the packets. However, such a method would increase the average case delay significantly and may be undesirable for applications requiring good average case performance. Hence, even though a network may not provide any guarantees, it is desirable to design it to provide good average case performance as well. If a source does not require good average case delay, it can reduce its buffer requirement by buffering the data or requesting a network to employ rate controllers on its path. Hence, if GR networks provide good average performance, they can support applications with different requirements.

2.6 Conclusions

In this chapter, we defined and analyzed the class of GR scheduling algorithms to enable a network to provide QoS guarantees to sources with different characteristics in heterogeneous environments.

We demonstrated that several work conserving and non-work conserving algorithms that either allocate only rate or separate rate and delay allocation belong to GR. We defined work conserving generalized Virtual Clock, Weighted Fair Queuing and Self Clocked Fair Queuing scheduling algorithms that can allocate variable rate to the packets of a flow. We also defined scheduling algorithms suitable for servers where packet fragmentation may occur. We demonstrated that if a class of rate controllers is employed for a flow in conjunction with any scheduling algorithm in GR, then the resulting non-work conserving algorithm also belongs to GR. This lead to the definition of non-work conserving equivalents of generalized Virtual Clock, WFQ and SCFQ algorithms and a combination of Delay EDD and a variant of Jitter EDD.

We presented a method for deriving the delay guarantee of a network of servers when: (1) different rates are allocated to packets of a flow at different servers on the path and the bottleneck server for each packet may be different, and (2) packet fragmentation and/or reassembly may occur. The delay guarantee enables a network to provide various service guarantees to flows conforming to any specification. The delay guarantee was then employed to illustrate the derivation of delay bounds for flows conforming to leaky bucket. Our method for determining these bounds is valid in internetworks and leads to tighter results. We finally presented architectural principles for the design of networks that employ scheduling algorithms in GR class. GR class simplifies the design of networks and enables support for applications with different characteristics and requirements.

Chapter 3

Understanding the Guaranteed Rate Class

In the previous chapter, we defined the class of Guaranteed Rate (GR) scheduling algorithms and demonstrated that large number of algorithms with different properties belong to the GR class. We then developed a framework for analysis of a network of GR servers and demonstrated that the GR class can provide QoS guarantees to sources with different traffic characteristics in heterogeneous networking environments. This demonstrated that the GR class is a useful class of algorithms. To gain a further understanding of the GR class, we explore two issues:

- Completeness of the GR class: The GR class contains most, but not all, of the algorithms proposed in the literature for providing QoS guarantees. The algorithms that do not belong to the GR class may enable a network to achieve higher utilization of its resources. Hence, the question of interest is whether the algorithms that do not belong to the GR class indeed lead to higher utilization, and, if they do, how much is the gain.

- Relative merits of algorithms in the GR class: The GR class contains algorithms with different properties. It includes algorithms : (1) that are work conserving and non-work conserving; (2) that are fair and unfair; and (3) that allocate only rate or achieve separation of rate and delay allocation. Since all these algorithms provide end-to-end QoS guarantees in heterogeneous environments, a natural question that arises is whether some subclasses of GR have advantages over the others.

We explore these two issues in this chapter. Our results demonstrate that: (1) the algorithms that do not belong to GR do not lead to higher utilization of a network; and (2) algorithms that are fair and/or achieve separation of rate and delay allocation are desirable.

The rest of this chapter is organized as follows. We first evaluate the completeness of the GR class in Section 3.1 and then consider the relative merits of algorithms in GR in Section 3.2. The conclusions of this chapter are presented in Section 3.3.

3.1 Evaluation of the GR Class

Most of the scheduling algorithms that have been proposed in the literature can be shown to belong to either the GR class or the class of Rate Controlled Service Disciplines (RCSD) presented in [19]. Since the RCSD class is known to be optimal when flows are served by a single server, we evaluate the completeness of the GR class by comparing its achievable utilization with the RCSD class [19].

In the next section we present the definition of the GR and RCSD classes and illustrate the differences between them. To facilitate the comparison between GR and RCSD, we present a definition of GR class which is different in form but equivalent to the definition presented in Chapter 2.

3.1.1 Background and Problem Formulation

Let there be K servers on the path of a flow and let the i^{th} server on the path be server i. For simplicity of exposition, let the propagation delay between the servers be 0 [1]. Then the GR and RCSD classes are defined as follows:

[1] All the subsequent analysis is applicable when the propagation delay is non-zero but bounded. However, use of non-zero propagation delay would clutter the presentation without providing any insight into the problem under investigation.

- GR class: The class of GR scheduling algorithms is defined based on *expected arrival time* of a packet. To define expected arrival time of a packet, let p_f^j be the j^{th} packet of flow f and r_f^j be rate assigned to it. Then, expected arrival time of packet p_f^j at server i, denoted by $EAT^i(p_f^j, r_f^j)$, is defined as:

  $$EAT^i(p_f^j, r_f^j) = \max\left\{A^i(p_f^j), EAT^i(p_f^{j-1}, r_f^{j-1}) + \frac{l_f}{r_f^{j-1}}\right\}$$

  where l_f is the length of packets of flow f [2], $A^i(p_f^j)$ is the arrival time of p_f^j at server i, and $EAT^i(p_f^0) + \frac{l_f}{r_f^0}$ is defined to be 0.

  Scheduling algorithm at server i belongs to GR if it guarantees that packet p_f^j will be transmitted by $EAT^i(p_f^j, r_f^j) + \alpha_f^{j,i}$ where $\alpha_f^{j,i}$ depends on the scheduling algorithm as well as the server and the flow characteristics. Since we are assuming fixed packet lengths for a flow, we will drop the superscript j in $\alpha_f^{j,i}$ and simplify the notation to α_f^i. We demonstrated in Chapter 2 that if each of the server on the path of a flow employs a scheduling algorithm in GR, then the end-to-end delay of packet p_f^j is given as:

  $$d_f^j \leq EAT^1(p_f^j, r_f^j) - A(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n \tag{3.1}$$

  where $A(p_f^j)$ is the time at which packet p_f^j is generated at the source.

- RCSD class: The RCSD class is defined based on the concept of shapers [19]. A shaper is a network element that ensures that its output traffic satisfies some burstiness constraints. If $AP^i(t_1, t_2)$ denotes the number of bits that are output from shaper S^i in time interval $[t_1, t_2]$, then shaper S^i delays the departure time of packets to ensure that $AP^i(t_1, t_2) \leq F^i(t_2 - t_1)$, where the function F^i characterizes the shaper S^i.

  Scheduling algorithm at server i belongs to the RCSD class with shaper S_f^i and delay d_f^i if it ensures that the departure time of a packet from the server is at most d_f^i more than its departure time from shaper S_f^i. RCSD class contains work conserving as well as non-work conserving algorithms (see [19, 60] for some examples). It has been shown in

[2] For ease of exposition, we have assumed that the length of the packets of a flow does not vary.

[19] that if each server on the path of a flow employs a scheduling algorithm in RCSD, then the network guarantees that the maximum end-to-end delay of any packet, denoted by d_f, is given as:

$$d_f = D_f^{\overline{S}} + \sum_{n=1}^{n=K} d_f^n \tag{3.2}$$

where $D_f^{\overline{S}}$ is the maximum delay incurred by the source traffic at shaper $\overline{S}_f$, which is a composition of shapers $S_f^1, .., S_f^K$. Intuitively, $\overline{S}_f$ is a shaper composed by sequentially connecting shapers $S_f^1, .., S_f^K$ (see [19] for a precise definition of composition of shapers).

To determine the relationship between the GR and RCSD classes, observe that variable rate allocation algorithms such as generalized Virtual Clock, SCFQ, and WFQ belong to the GR class. However, since there is no concept of variable rate allocation in RCSD, they do not belong to the RCSD class. On the other hand, if we restrict GR to reserve fixed rate R_f for flow f, then GR becomes a proper subset of RCSD. To observe this, note that the bound on the departure time of a packet in GR at server i can be interpreted as being at most α_f^i more than the departure time in a shaper that delays a packet till its expected arrival time. Since such a shaper generates a smooth constant bit rate (CBR) flow, we will refer to it as a CBR shaper. Hence, if scheduling algorithm at server i belongs to GR, it also belongs to RCSD with $d_f^i = \alpha_f^i$ and CBR shaper S_f^i. However, since arbitrary non-CBR shapers may be employed in RCSD, the converse is not true, i.e., a scheduling algorithm in RCSD may not belong to GR. Hence, if GR is restricted to reserve fixed rate for a flow, then it becomes a proper subset of RCSD[3]. The relationship between GR and RCSD is shown in Figure 3.1. To understand the difference between RCSD and GR class, in the rest of this chapter, we will assume that fixed rate R_f is reserved for flow f, i.e., $R_f = r_f^j$.

The main difference between GR and RCSD class is that if the burstiness of a source is completely smoothed at the edge of a network to reduce the buffer requirement in the network, then whereas the end-to-end delay does not increase in case of a network of GR servers, it may increase in case of a network of RCSD servers. To observe this, let a CBR shaper be employed

[3]The GR analysis leads to tighter results than the RCSD analysis when packet lengths of a flow vary over time. However, we will ignore this difference.

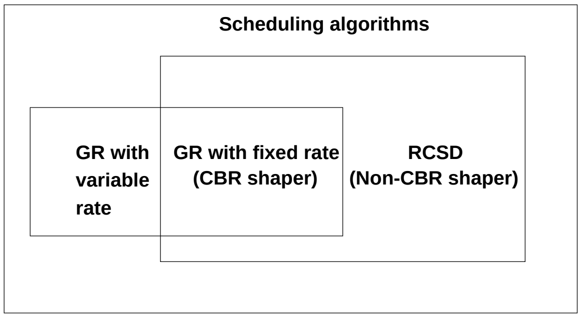


Figure 3.1: Relationship between GR and RCSD classes

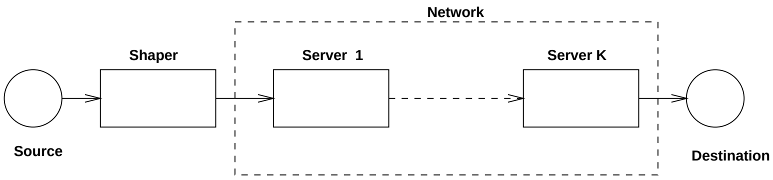


Figure 3.2: Network with Traffic Shaper

at the edge of the network, i.e., between the source and the first server (see Figure 3.2). Let the departure time of packet p_f^j from such a shaper, denoted by $L^S(p_f^j)$, be equal to its expected arrival time, i.e., $L^S(p_f^j) = EAT^S(p_f^j, R_f)$. Now consider the end-to-end delay in such a system in a network of GR and RCSD servers.

- Network of GR servers: The maximum delay incurred by a packet consists of two components - the delay in the shaper and the delay in the network. The delay in the shaper of packet p_f^j is $L^S(p_f^j) - A(p_f^j) = EAT^S(p_f^j, R_f) - A(p_f^j)$. Using (3.1), we conclude that the maximum delay of packet p_f^j in the network is $\widehat{EAT}^1(p_f^j, R_f) - \widehat{A}^1(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n$ where $\widehat{EAT}^1(p_f^j, R_f)$ and $\widehat{A}^1(p_f^j)$ are the expected arrival time and actual arrival time of p_f^j at server 1 when the shaper is employed. Hence, we conclude:

  $$d_f^j \leq EAT^S(p_f^j, R_f) - A(p_f^j) + \widehat{EAT}^1(p_f^j, R_f) - \widehat{A}^1(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n$$

  Since propagation delay has been assumed to be 0, the arrival time of a packet at the shaper is the same as the arrival time at the first server in a network without the shaper. Hence, $EAT^S(p_f^j, R_f) = EAT^1(p_f^j, R_f)$. Furthermore, due to the use of shaper,

$\widehat{EAT}^1(p_f^j, R_f) = \widehat{A}^1(p_f^j)$. Hence, we get:

$$d_f^j \leq EAT^1(p_f^j, R_f) - A(p_f^j) + \sum_{n=1}^{n=K} \alpha_f^n \tag{3.3}$$

Since (3.3) is same as (3.1), we conclude that the end-to-end delay does not change due to the presence of the shaper.

- Network of RCSD servers: Let the maximum delay incurred by packets in the shaper at the edge be denoted by D_f^C. Also, let the maximum delay incurred by the packets at shaper $\overline{S}_f$ after being shaped at the edge be $\widehat{D}_f^{\overline{S}}$. Hence, maximum end-to-end delay is given as:

  $$\widehat{d}_f = D_f^C + \widehat{D}_f^{\overline{S}} + \sum_{n=1}^{n=K} d_f^n$$

  Now, $D_f^C + \widehat{D}_f^{\overline{S}}$ may be greater than the delay $D_f^{\overline{S}}$ experienced by packets at shaper $\overline{S}_f$ when traffic is not shaped at the edge. This may occur because whereas a non-CBR flow may incur zero delay in shaper $\overline{S}_f$, i.e., $D_f^{\overline{S}} = 0$, it will incur a non zero delay in a CBR shaper, i.e., $D_f^C > 0$. To formally observe this, let $\overline{S}_f$ be a (kl_f, r_f) leaky bucket shaper (k is an integer) and flow f conform to (kl_f, r_f) leaky bucket. A shaper is a (σ, r) leaky bucket shaper, where σ is the burstiness and r the average rate, if it ensures that for all intervals $[t_1, t_2]$:

  $$AP(t_1, t_2) \leq \sigma + r(t_2 - t_1)$$

  A flow conforms to (σ, r) leaky bucket if it does not incur any delay in a (σ, r) leaky bucket shaper. Then, since flow f conforms to the shape enforced by the shaper $\overline{S}_f$, $D_f^{\overline{S}} = 0$. However, since D_f^C is given as $\frac{(k-1)l_f}{r_f}$ (see Chapter 2), if $k > 1$, $D_f^C + \widehat{D}_f^{\overline{S}} > 0$. Hence, $\widehat{d}_f$ may be greater than d_f. Thus in case of RCSD, the end-to-end delay may increase due to the presence of the shaper.

Hence, if buffer requirement in the network is reduced to that of a CBR flow by employing a shaper at the network edge then:

- In GR networks the delay does not increase. Hence, increasing the available buffer space does not effect the achievable utilization.

- In RCSD networks the delay may increase. Hence, to ensure that end-to-end delay requirements of the flow are met, higher resources may have to be reserved which may result into loss in achievable utilization.

Hence, we conclude that the available buffer space does not effect the achievable utilization of GR networks but may effect that of RCSD networks. Thus, RCSD networks can potentially achieve higher utilization by employing additional buffers.

To summarize, RCSD scheduling algorithms that employ non-CBR shapers do not belong to GR. Furthermore, RCSD networks that employ such algorithms may achieve higher utilization by using more buffers than that required in GR networks. We now determine the difference in achievable utilization of a network GR and RCSD servers. Since a RCSD scheduling algorithm does not belong to GR only when it employs a non-CBR shaper, henceforth, we will assume that non-CBR shapers are employed in RCSD.

To gain an intuitive understanding for the difference in achievable utilization of a network of GR and RCSD servers, first consider a single server scenario. Let the flows conform to leaky buckets, i.e., flow f conforms to a (σ_f, r_f) leaky bucket. Furthermore, for scheduling algorithms in GR, let $\alpha_f^i = \frac{l_f}{R_f} + \frac{l_{max}^i}{C^i}$ where C^i is the capacity of server i and l_{max}^i is the maximum length of packets served by server i (as shown in Chapter 2 scheduling algorithms such as WFQ and Virtual Clock satisfy this assumption). Since EDF has the largest schedulability region, let the RCSD scheduling algorithms employ EDF scheduler and a (σ_f, r_f) leaky bucket shaper for flow f [19].

Consider two flows f and m that conform to leaky bucket with parameters $(10pkt, 1pkt/s)$ and $(1pkt, 1pkt/s)$, respectively. Let both flows f and m require delay of $5.5s$ and be served by a single server with capacity $2pkts/s$. Then, from the results in [19], it can be shown that a RCSD server satisfies the deadline requirements of both the flows. On the other hand, in case of GR, from (3.1) and results in Chapter 2 we know that flow f would have to reserve at least rate R_f such that:

$$\frac{\sigma_f - l_f}{R_f} + \frac{l_f}{R_f} + \frac{l_{max}}{C} = 5.5$$

Hence, $\frac{10}{R_f} + 0.5 = 5.5$ which yields $R_f = 2pkt/s$, which is the capacity of the server. Hence, the server would not be able to accept flow m. Thus, the achievable utilization is higher in case of RCSD.

Though achievable utilization is higher in case of a single server for RCSD, it is not clear whether the same result holds in a network of servers. To observe this, let flow f have an end-to-end delay requirement of d_f. Since we have assumed that the shaper employed is the same as the input traffic, $D_f^{\overline{S}} = 0$. Hence, from (3.2) we get:

$$d_f = \sum_{n=1}^{n=K} d_f^n$$

If we assume each of the server is equally loaded, then the delay of the flow at each server is $\frac{d_f}{K}$. Thus, *intuitively*, for a given end-to-end delay, the resource requirement at each server for a flow increases linearly with the number of servers on the path. On the other hand, in a network of GR servers, we know from (3.1) and results in Chapter 2 that the reserved rate $R_f \geq r_f$ should be such that:

$$d_f \geq \frac{\sigma_f - l_f}{R_f} + K\frac{l_f}{R_f} + \sum_{i=1}^{i=K} \frac{l_{max}^i}{C^i} \tag{3.4}$$

Then, if σ_f is large, i.e., when RCSD servers may be expected to have larger utilization, R_f increases sub-linearly with the number of servers on the path. Hence, intuitively, whereas resource requirement increases linearly with the number of servers in a RCSD network, it increases sub-linearly in a GR network. Consequently, RCSD networks will have higher utilization only if the reduction in resource requirement yielded by additional buffer is sufficient to offset the difference in the linear and sub-linear increase. Otherwise, RCSD networks may not have higher utilization and, contrary to intuition, may have lower utilization.

To determine the difference in achievable utilization of RCSD and GR networks, we conduct a number of experiments. In the next section, we describe our experimental methodology and present the experimental results in Section 3.1.3.

3.1.2 Experimental Methodology

To experimentally determine the difference in achievable utilization of a network of GR and RCSD servers, we assume that all the flows conform to leaky bucket [4]. Hence, flow f is characterized by the quintuple $(\sigma_f, r_f, l_f, d_f, K_f)$ where σ_f,r_f,l_f,d_f, and K_f denote the burstiness, rate, packet length, end-to-end delay requirement, and the number of servers on the path of flow f, respectively. We make the following assumptions for GR and RCSD networks:

- GR: We assume that for GR servers $\alpha_f^i = \frac{l_f}{R_f} + \frac{l_{max}^i}{C^i}$ where C^i is the capacity of server i and l_{max}^i is the maximum length of packets served by server i (we remove this restriction in Section 3.1.3.2). Hence, rate $R_f \geq r_f$ that would satisfy the end-to-end delay requirements of flow f can be determined by (3.4). We assume that R_f is reserved for flow f at each of the servers.

- RCSD: We assume that RCSD servers employ EDF scheduler and a (σ_f, r_f) leaky bucket shaper for flow f. Furthermore, we assume each of the server is equally loaded and hence the delay of flow f at each server is $\hat{d}_f = \frac{d_f}{K}$.

We assume that the capacity of each server is C and maximum packet length at each server is l_{max}. The schedulability conditions that we employ for GR and RCSD are as follows:

- GR: We ensure that aggregate rate reserved by the flows at a server is less than the server capacity. Hence, if Q flows are served by a server, then the schedulability condition is $\sum_{i=1}^{i=Q} R_i \leq C$.

- RCSD: We use the schedulability condition presented in [18], which is a generalization of Theorem 1 of [19] for networks that have variable packet sizes, to determine the schedulability region. Specifically, flows $[1..Q]$ are schedulable if $\sum_{i=1}^{i=Q} r_i \leq C$ and the following set of inequalities hold:

$$\sum_{i=1}^{i=k} l_i + \sum_{i=1}^{i=k} (\sigma_i - l_i) + l_{max} \leq \hat{d}_k \left(C - \sum_{i=1}^{i=k-1} r_i \right) + \sum_{i=1}^{i=k-1} r_i \hat{d}_i \quad 1 \leq k \leq Q \tag{3.5}$$

[4] Since the traffic model used by the Internet and ITU standard bodies is leaky bucket, this is not a restrictive assumption.

Without loss of generality we have assumed that $\hat{d}_1 \leq .... \leq \hat{d}_Q$. If all the flows have the same packet length l, then we use Theorem 1 of [19] and ensure that the following set of inequalities hold:

$$\min\{(k+1)l, Ql\} + \sum_{i=1}^{i=k}(\sigma_i - l) \leq \hat{d}_k \left(C - \sum_{i=1}^{i=k-1} r_i\right) + \sum_{i=1}^{i=k-1} r_i \hat{d}_i \quad 1 \leq k \leq Q \quad (3.6)$$

To experimentally evaluate the difference in achievable utilization between RCSD and GR networks, we first assume that there are only two types of flow at the servers: low throughput and high throughput flows (experimental methodology for heterogeneous flows is presented in Section 3.1.3.3). Let $r_l < r_h$ and the quintuples $(\sigma_l, r_l, l_l, d_l, K_l)$ and $(\sigma_h, r_h, l_h, d_h, K_h)$ characterize the low and high throughput flows, respectively. We employ the schedulability conditions presented above to compute the schedulability region for RCSD and GR servers. The schedulability region is given by a set of tuples (n_h, n_l) such that n_h and n_l are the maximum number of high and low throughput flows, respectively, that can be admitted simultaneously (Figure 3.3 shows a hypothetical schedulability region for RCSD and GR servers). To compare schedulability regions of GR and RCSD and hence determine the difference in achievable utilization between GR and RCSD, we define two metrics: *average utilization gain* and *maximum utilization gain*. To define these metrics, we first define the functions $U^r(i)$ and $U^g(i)$ that compute the achievable utilization of a RCSD and a GR server, respectively, when i high throughput flows are admitted.

Let $L^r(i)$ and $L^g(i)$ be the maximum number of low throughput flows that are admitted in RCSD and GR when i high throughput flows are admitted, respectively. Also, let n_h^r and n_h^g be the maximum number of high throughput flows that can be admitted by RCSD and GR server, respectively. We need to define $U^r(i)$ and $U^g(i)$ for $i \in [1..N]$ where $N = \max\{n_h^g, n_h^r\}$. Consider the definition of $U^r(i)$. Clearly, when $i \leq n_h^r$, $U^r(i)$ is:

$$U^r(i) = \frac{r_h i + r_l L^r(i)}{C} \quad i \leq n_h^r$$

If $n_h^r \leq n_h^g$, then we will require $U^r(i)$ for $i > n_h^r$ also. In such a case, the number of high throughput flows that should be used in determining utilization is n_h^r. Furthermore, since $L^r(i)$ is not defined for $i > n_h^r$, $L^g(i)$ should be chosen as the number of low throughput flows. However, since at most $L^r(n_h^r)$ low throughput flows may be admissible along with n_h^r high throughput flows, we conclude that $\min\{L^r(n_h^r), L^g(i)\}$ number of low throughput flows should be used in the computation of $U^r(i)$ when $i > n_h^r$. Hence, we get:

$$U^r(i) = \frac{r_h n_h^r + r_l \min\{L^r(n_h^r), L^g(i)\}}{C} \quad n_h^r < i \leq N$$

The function $U^g(i)$ is defined analogously. Note that in the definition of $U^g(i)$, the actual rates of the flows, and not the reserved rates, are used.

We now define the two metrics as follows:

- *Average utilization gain*: This metric, denoted by U, is defined to capture the expected increase in utilization yielded by a RCSD server. It is defined as:

$$U = \frac{1}{N}\left(\sum_{i=1}^{i=N} \frac{U^r(i) - U^g(i)}{U^g(i)}\right)$$

- *Maximum average utilization gain*: This metric, denoted by U^+ is defined to capture the maximum expected increase in utilization yielded by a RCSD server. It is based on the observation that since GR class is a proper subset of RCSD class, it is always possible for a RCSD server to achieve the utilization of a GR server. Hence, for determining maximum expected increases in utilization, any loss in utilization yielded by RCSD may be ignored. Specifically, it is defined as:

$$U^+ = \frac{1}{N}\left(\sum_{i=1}^{i=N} \frac{\max\{0, (U^r(i) - U^g(i))\}}{U^g(i)}\right)$$

We are now ready to define our experimental methodology. In an experiment, we choose a set of low and high throughput flows. For each pair of low and high throughput flows chosen from their respective sets, we determine the schedulability region in RCSD and GR. For each pair, we compute U and U^+. Thus, we get a set of values for the two metrics. We then determine a cumulative distribution function for the metrics and employ it to determine the difference in achievable utilization between GR and RCSD servers. In the next section, we present and analyze the results of some of the experiments that we have conducted.

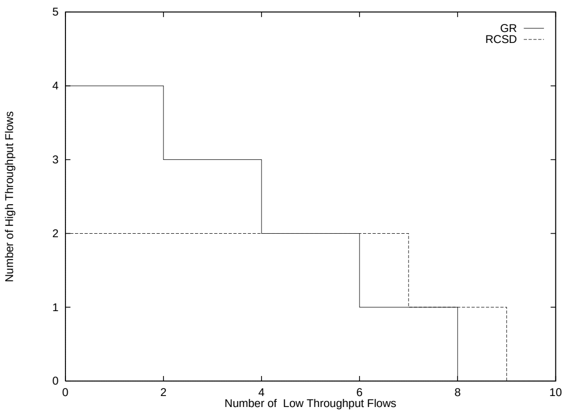


Figure 3.3: Example Schedulability Regions for GR and RCSD

3.1.3 Experimental Evaluation

3.1.3.1 Basic Experiments

We construct several sets of low and high throughput flows and choose several of their combinations. In a particular set, all the flows have equal number of servers on their path. Hence, let $S_l^{i,k}$ and $S_h^{i,k}$ denote the i^{th} low and high throughput set, respectively, with all flows having k servers on their path. Let l_l and l_h be the length of low and high throughput flow packets, respectively. Also, let Σ_l, Φ_l, and D_l^i be set of burstiness, rate, and delay values for the low throughput flows. Then, the set $S_l^{i,k}$, $i \in [1..3]$, is defined as follows:

$$S_l^{i,k} = \{(\sigma, r, d, l_l, k)| \ \ \sigma \in \Sigma_l \wedge r \in \Phi_l \wedge d \in D_l^i\}$$

The set $S_h^{i,k}$ is defined analogous to $S_l^{i,k}$. Observe that flows in set $S_l^{1,k}..S_l^{3,k}$ differ only in their delay values. The sets Σ, Φ, and D^i for low and high throughput flows are defined in Tables 3.1 and 3.2. We let the number of servers on the path be either $1, 3, 5, 10$, or 15. Note that since the average number of hops in Internet is 17, the number of servers we choose is conservative [29]. We conduct three experiments:

- EXP1: $S_l^{1,k}$ and $S_h^{1,k}$ are chosen to be the low and high throughput sets. In this case, some flows in the low throughput set have higher delay requirement than the flows in the high throughput set, and vice versa.

Type	Burstiness Σ	Rate Φ (Kb/s)
Low	$\{l_l, 10l_l, 20l_l\}$	$\{32, 64, 128\}$
High	$\{l_h, 10l_h, 20l_h\}$	$\{640, 1280, 2560\}$

Table 3.1: Burstiness and rate parameters for the experiments

Type	Delay		
	D^1 (ms)	D^2 (ms)	D^3 (ms)
Low	$\{25, 50, 75, 100, 200\}$	$\{25, 50, 75, 100\}$	$\{100, 125, 150, 175, 200\}$
High	$\{25, 50, 75, 100, 200\}$	$\{100, 125, 150, 175, 200\}$	$\{25, 50, 75, 100\}$

Table 3.2: Delay parameters for the experiments

- EXP2: $S_l^{2,k}$ and $S_h^{2,k}$ are chosen to be the low and high throughput sets. In this case, flows in the low throughput set always have lower delay requirement than the flows in the high throughput set.

- EXP3: $S_l^{3,k}$ and $S_h^{3,k}$ are chosen to be the low and high throughput sets. In this case, flows in the high throughput set always have lower delay requirement than the flows in the low throughput set.

We assume that the capacity of each server is $50Mb/s$ and conduct these experiments for network environments with fixed as well as variable size packets (leading to the computation of 52650 schedulability regions).

- Networks with fixed packet size: We choose $l_l = l_h = 512$ bits. Figures 3.4 and 3.5 plot the cumulative distribution function for the metrics U and U^+, respectively, for different values of k. Figure 3.4 shows that $70 - 80\%$ of the time, average gain is less than 0, and it is at most 0.15 in all the experiments when $k > 1$. Figure 3.5 shows that maximum average gain also is at most 0.15.

- Networks with variable packet size: In networks with variable packet size, we expect low throughput applications (such as audio) to have small packet size and high throughput

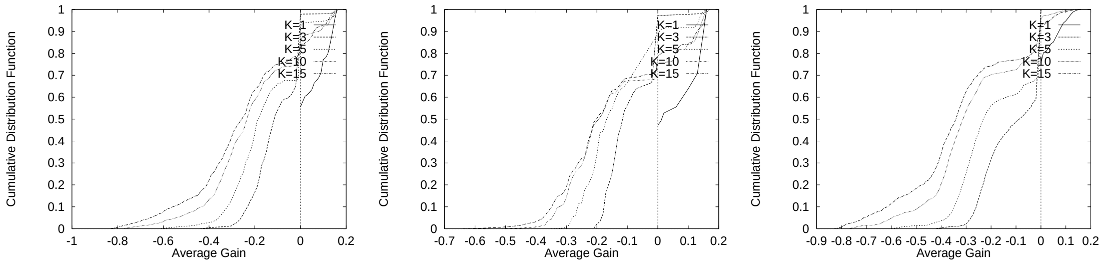


Figure 3.4: Fixed size packet networks: Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

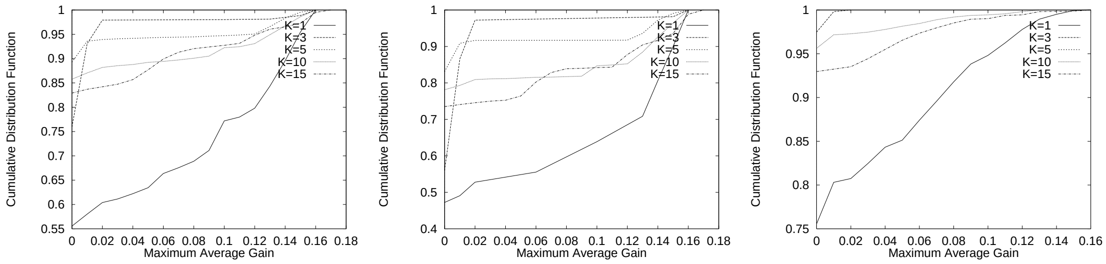


Figure 3.5: Fixed size packet networks: Maximum Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

applications such as video to have large packet sizes. Hence, we choose $l_l = 512$ bits and $l_h = 8000$ bits. Figures 3.6 and 3.7 plot the cumulative distribution function for the metrics U and U^+, respectively, for different values of k. Figure 3.6 shows that $85 - 90\%$ of the time average gain is less than 0 in all the experiments when $k > 1$. Figure 3.7 shows that 90% of the time the maximum average gain is less than 0.02 in all the experiments when $k > 1$. It also shows that sometimes the maximum average gain is as high as 1.

From the above experiments, we conclude, for both type of networks, that: (1) as per the intuition presented in Section 3.1.1, the average gain and maximum average gain decreases with increase in the number of servers on the path; (2) $70 - 80\%$ of the time, the average gain is less than 0; (3) $85 - 90\%$ of the time, the maximum average gain is less than $0.02 - 0.03$; and (4) $10 - 20\%$ of the time, the maximum average gain is non-negligible.

A further examination of the experimental results for the $10 - 20\%$ cases, in which the maximum average gain was non-negligible, demonstrated that most of the significant increase in utilization of RCSD occurred when both the low and high throughput flows have burstiness

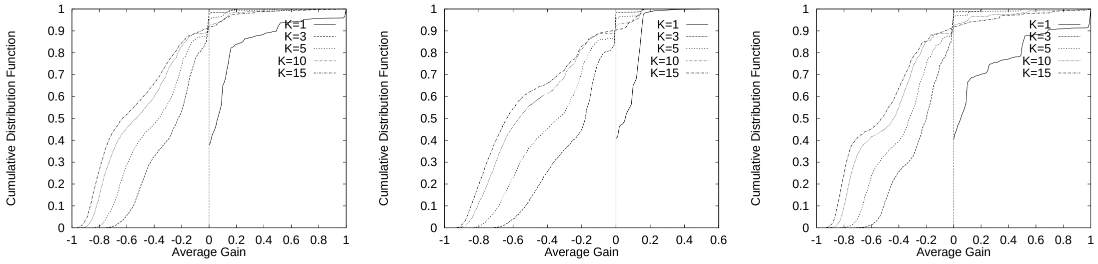


Figure 3.6: Variable size packet networks: Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

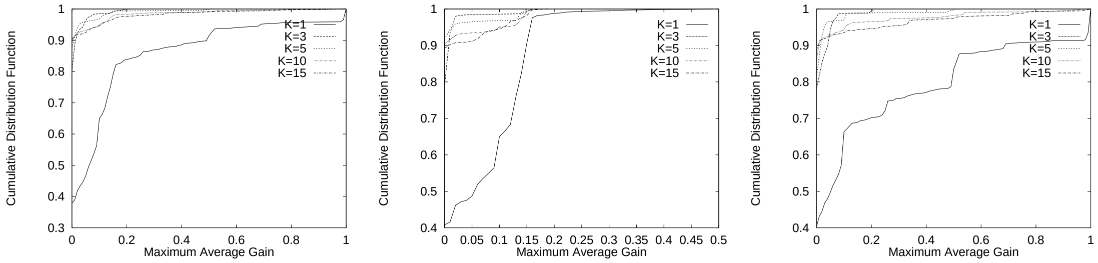


Figure 3.7: Variable size packet networks: Maximum Average gain in (a) EXP1, (b) EXP2, and (c) EXP3

equal to the packet length, i.e., the flows are CBR. Since theoretically the achievable utilization of GR and RCSD should be same when all the flows are CBR, this seems to be a contradictory result. However, this is only an apparent contradiction.

To observe this, recall that we had restricted the GR class to algorithms for which α^i was given as $\alpha_f^i = \frac{l_f}{R_f} + \frac{l_{max}^i}{C^i}$. This restriction included algorithms that allocate only rate (e.g., WFQ, Virtual Clock). However, GR class contains algorithms such as Delay EDD that separate rate and delay allocation and can have $\alpha_f^i < \frac{l_f}{R_f} + \frac{l_{max}^i}{C^i}$, or $\alpha_f^i > \frac{l_f}{R_f} + \frac{l_{max}^i}{C^i}$. Such algorithms have higher achievable utilization than the algorithms that allocate only rate. The unrestricted GR class that contains Delay EDD has the same achievable utilization as the RCSD class when the flows are CBR. Thus, to determine whether the non negligible increase in utilization of RCSD server is due to the restriction that we had placed on GR, in the next section, we repeat the experiments with the unrestricted GR class.

3.1.3.2 Experiments with Unrestricted GR Class

In the unrestricted GR class, the reserved rate $R_f \geq r_f$ that will meet the end-to-end delay requirement of flow f can be derived using:

$$d_f = \frac{\sigma_f - l_f}{R_f} + K\hat{d}_f \tag{3.7}$$

where we have assumed that $\alpha^i_f = \hat{d}_f$. Observe that there are several combinations of R_f and $\hat{d}_f$ that will satisfy (3.7). Hence, to determine both R_f and $\hat{d}_f$, we need one more constraint.

To develop this constraint, we will assume that the scheduling algorithm at each of the servers is Delay EDD. The schedulability conditions for Delay EDD for variable size and fixed size packet networks are given by (3.5) and (3.6), respectively, with l_f and R_f substituted for σ_f and r_f, respectively. From an examination of the schedulability conditions, we conclude that neither very high value of R_f nor a very small value of $\hat{d}_f$ are desirable. Hence, to balance these two tradeoffs we choose minimum value of $R_f \geq r_f$ that will satisfy (3.7) while ensuring that:

$$\hat{d}_f \geq \epsilon \frac{l_f}{R_f}$$

where $\epsilon \leq 1$.

We determine the schedulability region for Delay EDD as follows. Since the set of flows that are schedulable by rate only allocation algorithms are also schedulable by Delay EDD, we first determine a tuple (n_h, n_l) as in Section 3.1.2 for GR. We then choose $\epsilon = 0.1, 0.3, 0.5, 0.7,$ and 0.9 and determine the maximum number of low throughput flows, $\hat{n}_l \geq n_l$, that can be accepted along with n_h high throughput flows. This procedure when repeated for all values of n_h gives the schedulability region for Delay EDD.

We repeat the experiments presented in Section 3.1.3.1 to compare the achievable utilization in GR and RCSD. We find that the average gain becomes less than 0 in all the experiments for $k > 1$. Hence, we only plot the maximum average gain for fixed and variable length packet networks in Figures 3.8 and 3.9, respectively. As the figures demonstrate, the maximum average gain reduces significantly for $k > 1$. It is always less than 0.01 for fixed size packet networks and 0.05 for variable size packet networks.

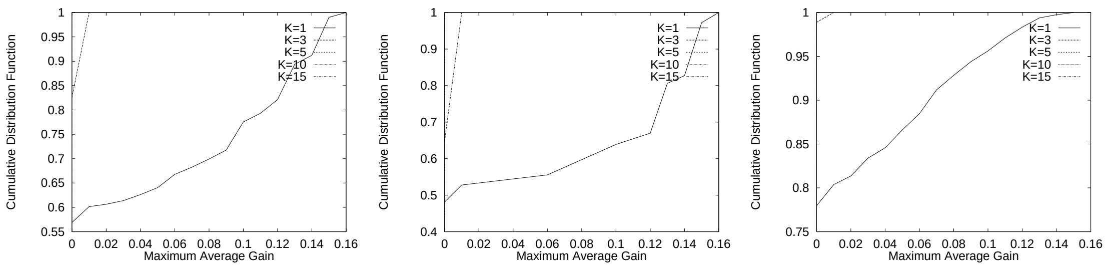


Figure 3.8: Fixed size packet networks: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

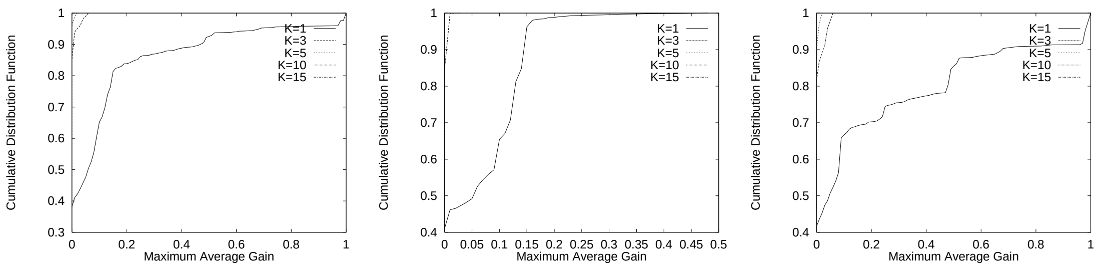


Figure 3.9: Variable size packet networks: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

We repeat all the experiments that we have presented so far by increasing all the delay values by 100ms. Figures 3.10 and 3.11 plot the maximum average gain for fixed size and variable size packet networks, respectively. As the figures demonstrate, there is no significant change in the results.

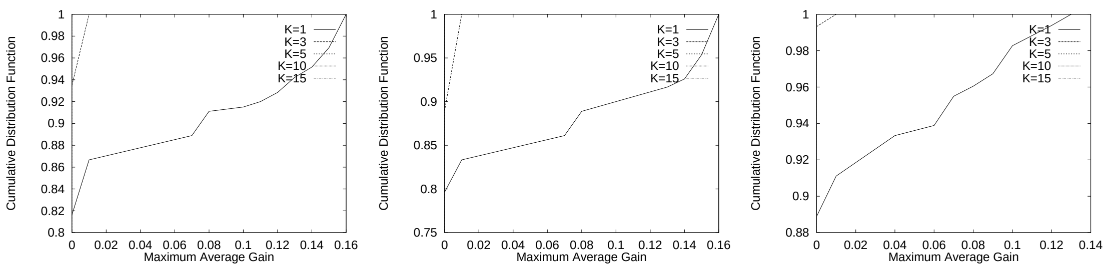


Figure 3.10: Fixed size packet networks with delay increased by 100 ms: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

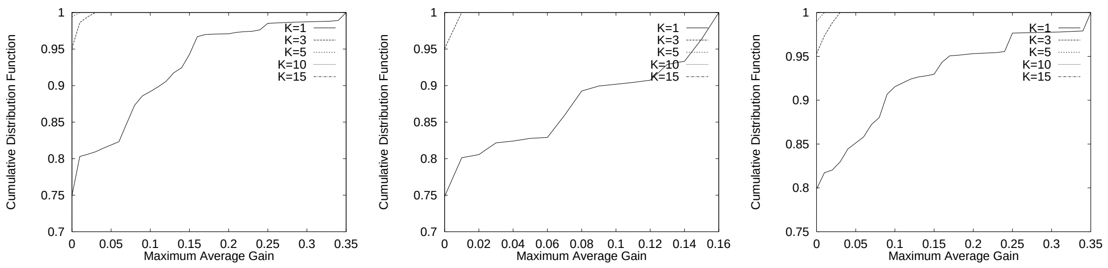


Figure 3.11: Variable size packet networks with delay increased by 100 ms: Maximum Average gain with Delay EDD in (a) EXP1, (b) EXP2, and (c) EXP3

3.1.3.3 Experiments with Randomly Generated Flows

In our experiments so far, we have assumed that there are only two types of flows at a server. In this section, we determine the difference in achievable utilization when the flows have heterogeneous characteristics. To achieve this objective, we generate a list of flows that have heterogeneous characteristics and then determine the subset of flows accepted in RCSD and GR using two methods:

1. We consider flows for acceptance in sequential order and determine the subset of flows that are accepted by a RCSD server using the schedulability conditions presented before. We then determine the subset of flows accepted by a GR server in two steps:

   - We determine the subset of the accepted flows that can be scheduled by a GR server using the schedulability conditions for Delay EDD and $\epsilon = 0.1, 0.3, 0.5, 0.7$, and 0.9.

   - If all the flows accepted by RCSD are accepted by GR, we determine the subset of the remaining flows that can be accepted by GR.

   Let Q^r and Q^g be the set of accepted flows in RCSD and GR, respectively. Define utilization of RCSD and GR servers, denoted by u^r and u^g respectively, as $u^r = \frac{1}{C}\sum_{i \in Q^r} r_i$ and $u^g = \frac{1}{C}\sum_{i \in Q^g} r_i$ where C is the link capacity. Then RCSD utilization gain, U^r is defined as: $U^r = \frac{u^r - u^g}{u^g}$.

2. This method is the same as the previous with the only change that roles of GR and RCSD are reversed. In this case, we define GR utilization gain, U^g as: $U^g = \frac{u^g - u^r}{u^r}$.

In an experiment, we generate a large number of different list of flows, determine U^r and U^g for each list, and determine the cumulative distribution function of both the metrics. The lists of flows are generated as follows. We choose range of burstiness factor $[b_{min}, b_{max}]$, packet lengths $[l_{min}, l_{max}]$, rate $[r_{min}, r_{max}]$, delay $[d_{min}, d_{max}]$, and number of servers on the path $[k_{min}, k_{max}]$. We then generate the quintuple $(\sigma_f, r_f, l_f, d_f, K_f)$ for a flow by choosing r_f, l_f,d_f, and K_f from their respective ranges with uniform probability. We choose burstiness factor b_f uniformly from its range and set $\sigma_f = b_f l_f$. A list of flows consists of such randomly generated flows and has at most one flow more than that required to ensure that the aggregate rate of the flows in the list is at least that of the link capacity.

We conduct 6 experiments and generate $100,000$ lists in each experiment. The ranges for all the factors other than delay are kept same in all the experiments. The ranges that we choose are: $[b_{min}, b_{max}] = [1, 20]$, $[l_{min}, l_{max}] = [512, 8196]$ bits, $[r_{min}, r_{max}] = [32, 2056]$ Kb/s, and $[k_{min}, k_{max}] = [3, 15]$. The six different ranges that we choose for delay are: $[25, 200]$ ms, $[100, 200]$ ms, $[100, 300]$ ms, $[100, 400]$ ms, $[100, 500]$ ms, and $[100, 600]$ ms. Figures 3.12(a) and 3.12(b) plot the metrics U^r and U^g for these experiments, respectively. As both the figures demonstrate, in most cases, RCSD has lower utilization than GR, but the loss decreases with increase in the upper bound on delay. Observe that the distribution of the gain in utilization of RCSD is similar to the experiments conducted in the previous section. Hence, it appears that the results are not correlated with the particular choice of parameters.

3.1.4 Discussion

The experiments in the previous section demonstrate that, in most cases, RCSD networks that employ non-CBR shapers have lower achievable utilization than GR networks. In few cases that they have higher utilization, the increase is small. Hence, from the experiments we conclude that even though GR is a proper subset of RCSD (when fixed rate is allocated to a flow), GR does not have lower achievable utilization.

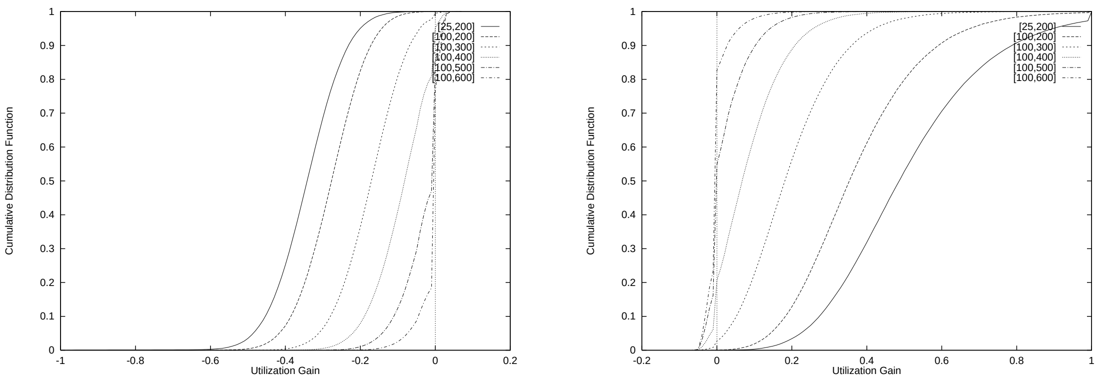


Figure 3.12: Randomly generated flows: (a) U^r (b) U^g

The experimental results also provide insights into the utility of buffers in networks for providing deterministic QoS. From the experimental results presented in the previous section, we know that RCSD networks with non-CBR shapers do not have higher achievable utilization than GR networks. Since RCSD networks with non-CBR shapers have higher buffer requirement than GR networks, we conclude that additional buffer in RCSD networks does not increase the achievable utilization of a network providing deterministic guarantees. Furthermore, contrary to intuition, use of additional buffer may decrease the achievable utilization significantly.

It may be argued that since additional buffer does increase utilization sometimes, although by very small percentage, it may be used only when it increases the utilization and not otherwise. However, an algorithm for determining when to use additional buffers such that utilization always increases is not known. Furthermore, in many of our experiments in the previous section, the schedulability region is similar in structure to the hypothetical example shown in Figure 3.3. In such a case, the utility of additional buffer is dependent not only on the characteristics of the flows, but also the specific combination of flows that have to be supported. Thus, it may be difficult to determine utility of additional buffers in a network where the set of flows may change dynamically.

Though additional buffer at the servers does not increase achievable utilization, it may be desirable for other reasons. If the burstiness of the sources is not smoothed at the edge and additional buffer is employed at the switches to absorb the burstiness, then due to the resultant overall work conserving nature of the system, the packets of sources will experience smaller average delay. Furthermore, a network may be able to achieve better statistical multiplexing of best effort packets and thereby achieve higher overall throughput [46]. Finally, a network that supports deterministic service as well as a service that provides guaranteed throughput, but not delay, may be able to employ additional buffer to increase its overall utilization [18].

3.2 Relative Merits of Algorithm in GR

The GR class contains algorithms with different properties. It includes algorithms : (1) that are work conserving or non-work conserving; (2) that are fair or unfair; and (3) that allocate only rate or achieve separation of rate and delay allocation. Since all these algorithms provide end-to-end QoS guarantees in heterogeneous environments, a natural question that arises is whether some subclasses of GR have advantages over the other. We explore this issue in this section.

The relative advantages of work conserving and non-work conserving scheduling algorithms are well known [34, 33]. Work conserving algorithms provide low average delay to flows but at the expense of increased buffer requirement in the network. Furthermore, they may be able to better multiplex packets from flows requiring best-effort service. On the other hand, as demonstrated in Chapter 2, non-work conserving algorithms reduce delay jitter as well as the buffer requirement in a network without effecting the end-to-end delay bounds. However, they increase the average packet delay for all flows and may effect the statistical multiplexing of best-effort packets.

Work conserving and non-work conserving algorithms are two ends of a spectrum. As we had demonstrated in Chapter 2, work conserving algorithms can employ rate controllers on a per flow basis, to reduce the buffer requirement for flows that prefer low delay jitter over low average delay. Thus, work conserving algorithms by being non-work conserving only on a per flow basis allow tradeoffs between average delay and buffer requirement to be made for each individual flow, i.e., they provide more flexibility than non-work conserving algorithms. Hence, we focus on work conserving algorithms.

The work conserving algorithms can be classified along two dimensions: (1) algorithms that are either fair or unfair; and (2) algorithms that allocate only rate or achieve separation of rate and delay allocation. In the following subsections, we evaluate the tradeoffs between these two classes of algorithms.

3.2.1 Fair and Unfair Algorithms

It was demonstrated in Chapter 2 that fair algorithms (e.g., SCFQ and WFQ) as well as unfair algorithms (e.g., Virtual Clock) belong to the GR class and guarantee that departure time of packet p_f^j is at most $\alpha_f^{j,i}$ more than its expected arrival time at server i, i.e.,

$$L^i(p_f^j) \leq EAT^i(p_f^j, r_f^j) + \alpha^{j,i}$$

We referred to this guarantee as delay guarantee.

Observe that expected arrival time of a packet of a flow is independent of the *actual* service received by a flow. Hence, the delay guarantee does not change even if a flow receives service that is better than what was guaranteed. To illustrate, consider a work conserving server with capacity $10pkt/s$ serving 10 flows each with reserved rate $1pkt/s$. Let flow 1 and flows $2..10$ send infinite packets at time 0 and 1 s, respectively. Since only flow 1 is backlogged from time 0 to time 1, 10 packets of flow 1 will be served by time 1, i.e., flow 1 will receive service at the rate $10pkt/s$ which is better than the reserved rate. However, since the expected arrival time of the 11^{th} packet of flow 1 is 10, the bound on its departure continues to remain $10 + \alpha^{j,i}$.

If the server employs any scheduling algorithm in GR, then it is not possible to improve upon the bound on the departure time of a packet derived from delay guarantee. This is because GR contains unfair algorithms in which a flow may not receive service for arbitrarily long durations. To observe this, let the scheduling algorithm employed by the server in the example be Virtual Clock. In Virtual Clock the priority of a packet of a flow is equal to its departure time in a server with capacity equal to the reserved rate for the flow. Hence, the priority of the 11^{th} packet of flow 1 is 11. Similarly, the priority of the 10^{th} packet of flows $2..10$ is 11. Consequently, the 11^{th} packet of flow 1 can not depart before time 10 s and, thus, the bound on its departure time can not be improved. However, if the server employs a fair algorithm, it may be possible to improve upon the bound.

Fair algorithms, in addition to providing delay guarantee, also ensure that in any time interval in which two flows are backlogged, they receive service in proportion to their reserved rates. Specifically, they ensure that in any time interval $[t_1, t_2]$ in which flows f and m are backlogged, $\mid \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \mid \leq H(f, m)$ where $W_n(t_1, t_2)$ is the service received by flow n in time interval $[t_1, t_2]$. This guarantee means that in any time interval in which a flow is backlogged, it will receive service at least at its reserved rate. Hence, intuitively, if packet p_f^j departs at time t, then a fair algorithm will guarantee that p_f^{j+1} will depart by $t + \frac{l_f}{r_f} + \beta$ where l_f is the length of packets, r_f the reserved rate and β a constant. Thus, fair algorithms also provide the following guarantee:

$$L^i(p_f^{j+1}) \leq \max\{L^i(p_f^j), A^i(p_f^{j+1})\} + \beta_f^{j+1,i} \tag{3.8}$$

We refer to the guarantee on the departure time of a packet based on its arrival time and the departure time of the previous packet as *delay-cum-throughput guarantee*. It can be shown that fair algorithms such as WFQ and SCFQ provide such a guarantee (the $\beta^{j+1,i}$ for WFQ can be derived using a modification of the analysis presented in [4]). Now consider the previous example with a fair scheduling algorithm. Since 10^{th} packet of flow 1 departs at 1 s and the scheduling algorithm is fair, using the delay-cum-throughput guarantee we conclude that the bound on the departure time of 11^{th} packet is $1 + \beta_1^{11}$. If $\beta_1^{11} - \alpha_1^{11} < 9$, then delay-cum-throughput guarantee yields a tighter bound.

To observe the advantages of delay-cum-throughput guarantee over delay guarantee, consider a 10 Mb/s fair server that is serving 10 flows, each with packet size of 200 bytes and reserved rate 1 Mb/s (i.e., for all flows n, $r_n = 1$Mb/s). Let the scheduling algorithm be such that $\alpha_n^j = \beta_n^j = \frac{l}{r_n}$ (this assumption holds for the Start-time Fair Queuing algorithm that we will present in Chapter 5). Let N flows (including flow f) be continuously backlogged and the rest of the flows send no packets. Since only N flows are backlogged and all flows have the same reserved rate flow f receives an effective throughput of $\frac{10}{N}$Mb/s. Hence, let departure time of p_f^j be $j\frac{200\ bytes}{\frac{10}{N}Mb/s}$. Figure 3.13(a) plots the bounds on departure time of packet p_f^{j+1} obtained using delay guarantee and delay-cum-throughput guarantee for $j \geq 1$ and $N = 10$ and $N = 5$. As the figure illustrates, when all the flows are backlogged, i.e., $N = 10$, the bound derived

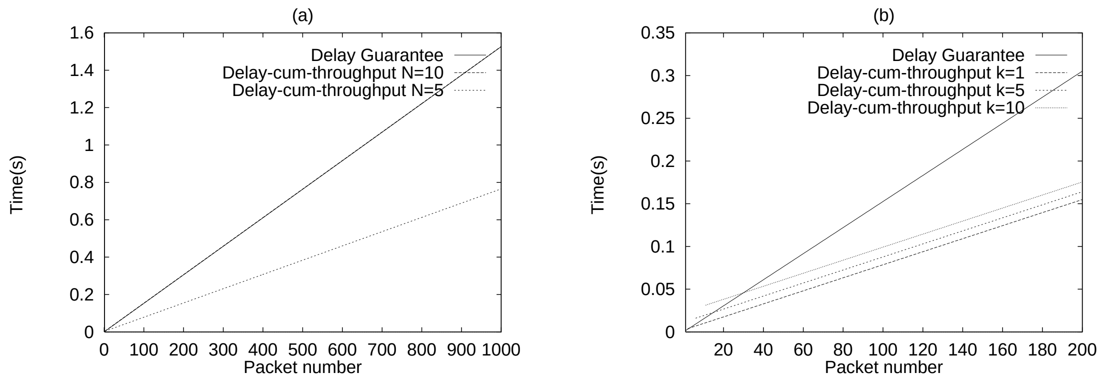


Figure 3.13: Bounds derived using delay and delay-cum-throughput guarantee for: (a) different number of backlogged flows and (b) different values of k.

using delay guarantee is tighter. However, when only five flows are backlogged, i.e., $N = 5$, then the bound derived using delay-cum-throughput guarantee is significantly better. Hence, delay-cum-throughput guarantee improves upon the bounds of delay guarantee when the actual service received by a flow is better than the service that has been guaranteed.

If a flow is continuously backlogged, i.e., $\max\{L^i(p_f^{j-1}), A^i(p_f^j)\} = L^i(p_f^{j-1})$, then (3.8) can be used to determine a bound on the departure time of any packet p_f^{j+k} ($k \geq 1$) given the departure time of packet p_f^j. Specifically, from recursive use of (3.8), we get:

$$L(p_f^{j+k}) \leq L(p_f^j) + \sum_{n=j+1}^{n=j+k} \beta_f^{n,i} \quad k \geq 1$$

Figure 3.13(b) plots the bound on the departure time of packet p_f^{j+k} for $k = 1, 5,$ and 10 and $j \geq 1$ assuming that the departure time of packet p_f^j is given as in previous scenario with $N = 5$ (i.e., 5 backlogged flows). As the figure demonstrates, for high values of k and small values of j, the bound derived using delay guarantee is tighter. However, after the flow has received better service than that was guaranteed for sufficiently long duration, i.e., for higher values of j, the bounds derived using delay-cum-throughput guarantee become significantly tighter.

In networks that carry traffic with multiple time-scale variation (for example, video traffic), many flows will receive service better than that guaranteed by the network. Hence, the improved bounds yielded by delay-cum-throughput guarantee are desirable. To enable a network of servers also to provide such improved bounds, we now derive the delay-cum throughput guarantee of a network of servers.

When a flow is served by a network of servers, a destination knows the departure time of a packet from the last server. Furthermore, from the traffic characteristics of a flow, it may also know the arrival time of a packet at the first server on the path. Hence, the objective is to determine a bound on the departure time of a packet from the last server based on its arrival time at the first server and departure time of the previous packet at the last server. We achieve this objective in Theorem 3.1.

Let τ^i and $\hat{\tau}^i$ denote the upper and lower bound on the propagation delay between servers i and $i+1$, respectively. Henceforth in this section, we will refer to a single flow f, and hence, drop the subscript f from all the variables.

Theorem 3.1 *If there are K servers on the path of a flow, and each server satisfies (3.8), then*

$$L^i(p^{j+1}) \leq \max\{L^K(p^j) - \sum_{k=1}^{k=K-1} \hat{\tau}^k, A^1(p^{j+1})\} + \sum_{k=1}^{k=i-1} \tau^k + \sum_{k=1}^{k=i} \beta^{j+1,i} \quad 1 \leq j \ ; \ 1 \leq i \leq K \tag{3.9}$$

Proof: The proof is by induction on i.

Base Case ($i = 1$)**:** We know that

$$L^1(p^j) \leq L^K(p^j) - \sum_{k=1}^{k=K-1} \hat{\tau}^k \tag{3.10}$$

Substituting (3.10) in (3.8) with $i = 1$ yields (3.9) for $i = 1$.

Induction Hypothesis: Assume (3.9) holds for $1 \leq i \leq m$.

Induction : We need to show (3.9) holds for $1 \leq i \leq m+1$. From (3.8), i.e., the definition of throughput guarantee, we know:

$$L^{m+1}(p^{j+1}) \leq \max\{L^{m+1}(p^j), A^{m+1}(p^{j+1})\} + \beta^{j+1,m+1}$$

Since $A^{m+1}(p^{j+1}) \leq L^m(p^{j+1}) + \tau^m$, using induction hypothesis we get:

$$
\begin{aligned}
L^{m+1}(p^{j+1}) \quad &\leq \quad \max\left\{L^{m+1}(p^j), \max\left\{L^K(p^j) - \sum_{k=1}^{k=K-1}\hat{\tau}^k, A^1(p^{j+1})\right\} + \sum_{k=1}^{k=m}\tau^k + \right. \\
&\quad \left. \sum_{k=1}^{k=m}\beta^{j+1,k}\right\} + \beta^{j+1,m+1}
\end{aligned}
$$

By: (1) moving $\sum_{k=1}^{k=m}\beta^{j+1,i}$ out from the second term of first maximum operator, (2) moving the term $\sum_{k=1}^{k=m-1}\tau^k$ inside the second maximum operator, and (3) then removing the second maximum operator we get:

$$
\begin{aligned}
L^{m+1}(p^{j+1}) \quad &\leq \quad \max\left\{L^{m+1}(p^j), L^K(p^j) - \sum_{k=1}^{k=K-1}\hat{\tau}^k + \sum_{k=1}^{k=m}\tau^k, A^1(p^{j+1}) + \sum_{k=1}^{k=m}\tau^k\right\} + \\
&\quad \sum_{k=1}^{k=m+1}\beta^{j+1,k} \qquad (3.11)
\end{aligned}
$$

Now,

$$
L^{m+1}(p^j) \leq L^K(p^j) - \sum_{k=m+1}^{k=K-1}\hat{\tau}^k \qquad (3.12)
$$

Substituting (3.12) in (3.11) we get:

$$
L^{m+1}(p^{j+1}) \leq \max\left\{L^K(p^j) - \sum_{k=m+1}^{k=K-1}\hat{\tau}^k, L^K(p^j) - \sum_{k=1}^{k=K-1}\hat{\tau}^k + \sum_{k=1}^{k=m}\tau^k, ...\right\} + ... \qquad (3.13)
$$

Since $\sum_{k=1}^{k=m}\tau^k - \sum_{k=1}^{k=m}\hat{\tau}^k \geq 0$,

$$
L^K(p^j) - \sum_{k=m+1}^{k=K-1}\hat{\tau}^k \leq L^K(p^j) - \sum_{k=m+1}^{k=K-1}\hat{\tau}^k + (\sum_{k=1}^{k=m}\tau^k - \sum_{k=1}^{k=m}\hat{\tau}^k)
$$

Hence, the first term in the maximum operator in (3.13) can be removed. Thus,

$$
\begin{aligned}
L^{m+1}(p^{j+1}) \quad &\leq \quad \max\left\{L^K(p^j) - \sum_{k=1}^{k=K-1}\hat{\tau}^k + \sum_{k=1}^{k=m}\tau^k, A^1(p^{j+1}) + \sum_{k=1}^{k=m}\tau^k\right\} + \\
&\quad \sum_{k=1}^{k=m+1}\beta^{j+1,k} \\
&\leq \quad \max\left\{L^K(p^j) - \sum_{k=1}^{k=K-1}\hat{\tau}^k, A^1(p^{j+1})\right\} + \sum_{k=1}^{k=m}\tau^k + \sum_{k=1}^{k=m+1}\beta^{j+1,k}
\end{aligned}
$$

Hence (3.9) holds for $1 \leq i \leq m+1$ and the theorem follows. ∎

If a destination knows the relationship between the arrival time of packet p^{j+1} at the first server and departure time of p^j at the last server (possibly from the traffic characteristics of the source) and the service received by a flow is better than that guaranteed by the network, then just like in the case of a single server, the destination can utilize (3.9) to derive bounds on packet delay better than those determined by end-to-end delay guarantee. In particular, if for all packets p^{j+1}, $L^K(p^j) = \max\{L^K(p^j) - \sum_{n=1}^{n=K-1} \hat{\tau}^k, A^1(p^{j+1})\}$, then by recursive use of (3.9), we get:

$$L^K(p^{j+k}) \leq L^K(p^j) + \sum_{m=1}^{m=k} \left(\sum_{n=1}^{n=K} \beta^{j+m,i} + \sum_{n=1}^{n=K-1} \tau^k \right) \quad 1 \leq k \tag{3.14}$$

To observe the advantage of bounds derived using (3.14), consider a flow that is served by 5 servers. Let the capacity of each server be $C = 10Mb/s$ and for ease of exposition, let there be zero propagation delay between them. Let each server serve $N = 10$ flows, each with reserved rate 1Mb/s and packet size of 200 bytes. Also, let the flow be continuously backlogged at the first server. Let $L^K(p^j)$ be given as:

$$L^K(p^j) = \sum_{n=1}^{n=j-1} \frac{l}{r^j} + K\alpha$$

where $l = 200$ bytes, $K = 5$ and $\alpha = \frac{Nl}{C}$. Let $r^j = 2Mb/s$ for the first 1000 packets, i.e., let them receive service better than that guaranteed by the network, and let $r^j = 1Mb/s$ for $j > 1000$. Figure 3.14 plots the bound on the departure time of packet p^{j+k} ($k = 1, 5, 10$) for different values of j using (3.14) as well as the end-to-end delay guarantee for this scenario. As the figure demonstrates, (3.14) improves upon the bounds of delay guarantee and tracks the actual arrival time of packets much more closely.

We envision the end-to-end delay-cum-throughput guarantee to be useful for at least two classes of applications:

- Flow-controlled data applications: Consider a flow-controlled data source that reserves a minimum rate at each of the servers on the path to the destination. To increase its throughput by taking advantage of statistical multiplexing of various sources, it estimates the bottleneck rate, which is at least the reserved rate, and sends at the estimated bottleneck rate [33]. Due to the fluctuations in the bottleneck rate as well as the inherent delay

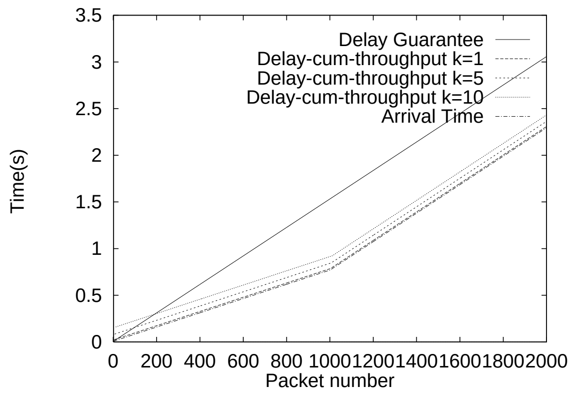


Figure 3.14: Bounds derived using delay and delay-cum-throughput guarantee for different values of k.

and errors in the estimation process, such a source may send at a rate higher than the bottleneck rate. This will lead to queue build up at the bottleneck server and eventually packet losses. Let packets $p^j, ..., p^{j+k}$ be lost due to buffer overflow. In the simplest case, a destination can detect loss of these packets only on arrival of packet p^{j+k+1}. However, if the network provides delay-cum-throughput guarantee, then the destination can use (3.14) to determine a bound on the arrival time of packets $p^j, ..., p^{j+k}$ and declare them lost if they do not arrive by then. It can thus detect packet losses earlier than the arrival of packet p^{j+k+1}. The early detection of packet losses can be used by a destination to "close" the feedback loop between a source and destination faster and thus improve the throughput of the source.

- Adaptive, real-time, playback applications: Audio and video applications that can tolerate discontinuities in playback and adapt their playback point as per the network congestion, may reserve a minimum rate and send packets at a higher rate. In such a scenario, a destination can use the delay-cum-throughput guarantee to determine a bound on the arrival time of future packets and use the bounds to suitably adapt the playback point.

Thus, we conclude that fair algorithms provide better performance bounds than unfair algorithms that can be exploited by flow-controlled and adaptive playback applications. Hence, they are more desirable.

Fair algorithms not only provide better performance bounds, but also lead to a better network architecture. To observe this, consider a source that reserves a rate and is bursty (i.e., generates data at rate higher than reserved rate for short durations). The source can transmit data in two ways:

- It can smooth out burstiness at edge and transmit at the reserved rate. In such a case, since the overall system is non-work conserving, the statistical multiplexing gains achievable by a network may be reduced.
- It can let the bursts into the network. In such a case, since the overall system is work conserving, a network may be able to achieve better statistical multiplexing of best effort packets and thereby achieve higher overall throughput [46].

Now, consider the incentives for a source and a network for the two transmission methods. Since work conserving behaviour is desirable from a networks perspective, a network has incentives to accept bursts. On the other hand, since a network may charge higher price for bursty behaviour, a source has incentives to smooth out the burstiness. But it may have other incentives to not smooth out the bursts, i.e., let the bursts into the network. Consider networks that employ fair or unfair algorithms:

- Fair algorithms: If a source sends bursts in the network and it receives service better than the guaranteed service, its packets will experience low average delay. Furthermore, since fair algorithms also provide delay-cum-throughput guarantee, the delay bound on packet will improve. Consequently, the packets will experience low delay-jitter also. Hence, if a source derives utility from low average delay, delay-jitter, and improved delay bounds, it has incentives to send bursts.
- Unfair algorithms: If a source sends bursts in the network, then average packet delay may be low. However, since unfair algorithms provide only delay guarantee, the delay bounds do not change and consequently the delay-jitter may be high. Hence, if a source desires low average delay and delay-jitter, then it does not have any incentives to send bursts.

	Fair algorithms	Unfair algorithms
Source	+	-
Network	+	+

Table 3.3: Incentive structure in fair and unfair algorithms

	Delay guarantee	Delay-cum-throughput guarantee
Fair algorithms	Yes	Yes
Unfair algorithms	Yes	No

Table 3.4: Comparison of Fair and Unfair algorithms

Thus, whereas in case of fair algorithms a source has incentives to send bursts in the network, in case of unfair algorithms, it has no incentives but only disincentives (unless low average delay and high delay-jitter are acceptable).

From the above examination, we conclude that the incentive structure for sending and accepting bursts is as shown in Table 3.3. In case of fair algorithms a source has incentives to send bursts and the network has incentives to accept it. In case of unfair algorithms, the network continues to have incentives to accept bursts but a source does not. Hence, in case of fair algorithms the incentives will lead to the desired source behaviour. In case of unfair algorithms, the incentives may lead to source behaviour that is undesirable from network perspective. From these considerations, we conclude that fair algorithms lead to a better network architecture.

To summarize, fair algorithms in addition to providing delay guarantee, which is provided by unfair algorithms also, provide delay-cum-throughput guarantee (see Table 3.4). The delay-cum-throughput guarantee yields better bounds on end-to-end packet delay when a source receives service better than that guaranteed by the network. Since in a network that carries traffic with multiple time-scale variations, flows will receive service better than that guaranteed by the network, such improved bounds are desirable. Furthermore, as we illustrated, these bounds can be exploited by flow controlled and real-time playback adaptive applications. Finally, fair algorithms lead to a better network architecture than unfair algorithms. Hence, we conclude that fair algorithms are more desirable than unfair algorithms.

3.2.2 Rate-only and Rate-and-delay Allocation Algorithms

The scheduling algorithms in GR can be classified as either allocating only rate or achieving separation of rate and delay allocation based on the relationship between $\alpha_f^{j,i}$ and the reserved rate R_f for a flow. To understand the classification, observe that $\alpha_f^{j,i}$ is the delay that will be experienced by packet p_f^j at server i if the flow were to transmit at its reserved rate. Hence, the two classes are defined as follows (see Table 3.5):

- Rate-only allocation algorithms: The algorithms belonging to this class have $\alpha_f^{j,i} = \frac{l_f^j}{R_f} + \beta^i$ where β^i is a positive constant. Algorithms such as WFQ, Virtual Clock, and SCFQ belong to this class. Since $\alpha_f^{j,i}$ is inversely proportional to R_f, the algorithms in this class provide high delay to low throughput flows. Thus, if a low throughput flow (such as audio flow) desires low delay, it can achieve it only by increasing its rate reservation; this may lead to loss in achievable utilization of a network. The main advantage of this class of algorithms is that since they only require a server to ensure that the aggregate reserved bandwidth is less than the server capacity, the time-complexity of admission control is $O(1)$.

- Rate-and-delay allocation algorithms: The algorithms is this class can have $\alpha_f^{j,i} < \frac{l_f^j}{R_f}$ or $\alpha_f^{j,i} > \frac{l_f^j}{R_f}$. Algorithms such as Delay EDD belong to this class [13]. Thus, they decouple delay from the reserved rate for a flow and consequently can provide low delay to low throughput flows without increasing rate reservation. Hence, they may lead to higher achievable utilization of a network. On the other hand, the time-complexity of admission control for these algorithms is $O(N)$ where N is the number of flows at a server [64].

The experiments presented in Sections 3.1.3.1 and 3.1.3.2 demonstrate that rate-and-delay allocation algorithms in fact lead to higher utilization of a network. Hence, they are desirable. However, whether the increase in achievable utilization yielded by them is sufficient to warrant the increased complexity in admission control is an open question.

Algorithms	$\alpha_f^{j,i}$
Rate-only	$\frac{l_f^j}{R_f} + \beta^i$
Rate-and-delay allocation	$\alpha_f^{j,i} < \frac{l_f^j}{R_f}$ or $\alpha_f^{j,i} > \frac{l_f^j}{R_f}$

Table 3.5: Rate-only and rate-and-delay allocation algorithms

3.3 Conclusions

In this chapter, we studied two issues:

- Completeness of the GR class: We experimentally demonstrated that even though GR class does not contain all the scheduling algorithms proposed in the literature to provide guaranteed QoS, it does not lead to loss is achievable utilization of a network.

- Relative merits of algorithms in the GR class: We argued that work conserving algorithms are more desirable than non-work conserving ones. We then classified the work conserving algorithms along two dimensions:

  - Fair and unfair algorithms: To evaluate the relative advantages of the two classes, we introduced the concept of *delay-cum-throughput guarantee*. We demonstrated that fair algorithms in addition to providing delay guarantee, which is provided by unfair algorithms also, provide delay-cum-throughput guarantee. The delay-cum-throughput guarantee yields better bounds on end-to-end packet delay when a source receives service that is better than that guaranteed by the network. These improved bounds can be exploited by flow-controlled and real-time playback adaptive applications. Finally, we argued that fair algorithms lead to a better network architecture than unfair algorithms. Hence, we conclude that fair algorithms are more desirable than unfair algorithms.

  - Rate-only and rate-and-delay allocation algorithms: Our experiments demonstrated that rate-and-delay allocation algorithms lead to higher achievable utilization than the rate-only allocation algorithms. Hence, rate-and-delay allocation algorithms are desirable.

Thus, algorithms that are fair and/or achieve separation of rate-and-delay allocation are desirable.

Chapter 4

Requirements for Scheduling Algorithms

In the previous chapters, we presented the Guaranteed Rate (GR) framework for the analysis of packet scheduling algorithms. From our analysis of the algorithms in GR, we concluded that scheduling algorithms that are fair and/or achieve separation of rate and delay allocation are desirable. In this chapter, we determine the requirements for suitable packet scheduling algorithms for integrated services networks. We achieve this objective by determining the applications and network requirements for packet scheduling algorithms.

Our analysis demonstrates that a suitable packet scheduling algorithm for integrated services networks should: (1) achieve low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provide fairness to support variable bit rate video; (3) provide fairness over variable rate servers to support throughput-intensive flow-controlled data applications; (4) provide QoS guarantees over variable rate servers; (5) support hierarchical link bandwidth allocation to enable a network to provide link-sharing service and facilitate coexistence of multiple services and protocols; and (6) be computationally efficient.

The rest of the chapter is organized as follows. We examine the applications and network requirements in Sections 4.1 and 4.2, respectively, and summarize the requirements in Section 4.3. We evaluate the existing algorithms with respect to the requirements in Section

4.4. Finally, Section 4.5 presents the conclusions of this chapter.

4.1 Applications Requirements

Consider the requirements of some of the principal applications envisioned for integrated services networks:

- *Audio applications*: Audio applications are low throughput in nature (for example, voice quality audio is $64Kb/s$) and very sensitive to packet delay. To maintain adequate interactivity, they require low average and maximum packet delay. Hence, a scheduling algorithm should be able to provide low delay to low throughput applications.

- *Video applications*: Variable bit rate (VBR) video sources, which are expected to impose significant requirements on network resources, have unpredictable as well as highly variable bit rate requirement at multiple time-scales [25, 26, 28]. Whereas the variation at time-scale of tens of milliseconds occurs due to use of interframe compression algorithms such as MPEG, variations at the time-scale of seconds or longer occur due to the inherent variation in scene complexity. Since frame and scene complexities vary dynamically, these variations are unpredictable. These features impose two key requirements on network resource management:

  - Due to the difficulty in predicting the bit rate requirement of VBR video sources, video flows may utilize more than the reserved bandwidth. As long as the additional bandwidth used is not at the expense of the other flows (i.e., if the flow utilizes idle bandwidth), it should not be penalized in the future by reducing its bandwidth allocation.

  - Due to multiple time-scale variation in the bit rate requirement of video sources, to achieve efficient utilization of resources, a network may have to over-book available bandwidth. Since such over-booking may yield persistent congestion, a network should provide some QoS guarantees even in the presence of congestion.

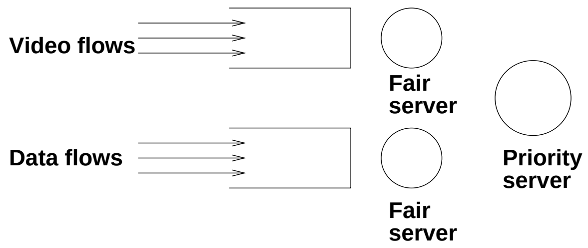


Figure 4.1: Occurrence of variation in server capacity for data flows

Unfair scheduling algorithms, such as Virtual Clock [62], Delay EDD [61], etc., penalize flow for use of idle bandwidth and do not provide bandwidth allocation guarantee in the presence of congestion [42]. Fair scheduling algorithms, on the other hand, guarantee that, regardless of prior usage or congestion, bandwidth would be allocated fairly[42]. Hence, fair scheduling algorithms are desirable for video applications.

- *Data applications*: To support low-throughput, interactive data applications (e.g., telnet), scheduling algorithms must provide low average delay. On the other hand, to support throughput-intensive flow-controlled applications in heterogeneous, large-scale, decentralized networks, scheduling algorithms must allocate bandwidth fairly [12, 33, 45]. Fair allocation of bandwidth at each server enables a network to achieve max-min fair allocation of network bandwidth in heterogeneous environments [12, 44]. Due to the coexistence of VBR video sources and data sources in integrated services networks, the bandwidth available to data applications may vary significantly over time. Consequently, the effective capacity of a server for data applications may vary over time. To observe this variation, consider a server that gives static high priority to real-time flows (such as VBR video flows) over data flows and schedules the packets of data flows using a fair algorithm (see Figure 4.1). In such a case, the capacity of the server for the data flows varies over time. Hence, the fairness property of the scheduling algorithm must hold regardless of variation in a server capacity.

Hence, in summary, a suitable scheduling algorithm should: (1) achieve low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provide fairness to support VBR video; and (3) provide fairness over variable rate servers for throughput-intensive flow-controlled data applications.

4.2 Network Requirements

An integrated services network imposes several requirements on packet scheduling algorithms. Some of the principal requirements that we have identified are:

- *It should provide QoS guarantees over variable rate servers.* Traditionally, scheduling algorithms have been analyzed only for constant bit rate servers that service fixed number of bits per unit time. However, service rate of flow-controlled, broadcast medium, and wireless links may fluctuate over time. Fluctuation in service rate may also occur due to variability in CPU capacity available for processing packets (for example, a CPU constrained IP router may not have sufficient CPU capacity to process packets when routing updates occur). If a server is shared by multiple types of traffic with some traffic types being given priority over the other, then for lower priority traffic, the link appears as a server with fluctuating service rate. Hence, it is desirable that scheduling algorithms provide guarantees when a server is a variable rate server.

- *It should facilitate coexistence of heterogeneous services and protocols.* An integrated services network will provide several services as well as employ multiple protocol families that support different traffic types and/or congestion control mechanisms. For example, as demonstrated in [25, 26] since stored video is less delay sensitive than interactive video, the network protocols employed for stored and interactive video may be different. Similarly, whereas a network will not overbook resources while providing deterministic services, it may overbook resources when providing services with weaker guarantees such as *controlled load* service defined in [54]. Since these services as well as the protocols may be incompatible with each other, a mechanism that provides isolation between different services and protocols and enables their coexistence is required.

  Hierarchical link bandwidth allocation is one possible mechanism for managing heterogeneity in integrated services networks [15, 49]. In hierarchical link bandwidth alloca-

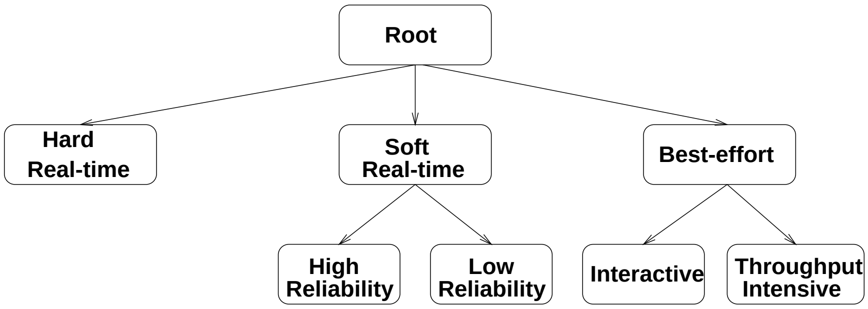


Figure 4.2: An example hierarchical link bandwidth allocation

tion, link bandwidth is allocated to a set of classes. The classes, in turn, recursively allocate the bandwidth among their subclasses. A class ensures that each of its sub-classes receives a minimum requested bandwidth. Such a mechanism can employed by a network to support heterogeneous services as well as multiple protocol families that support different traffic types and/or congestion control mechanisms. For example, a network can support hard and soft real-time as well as best effort services by allocating the link bandwidth between them as per the expected requirements of each of the service (see Figure 4.2). To support high and low reliability soft real-time services, the bandwidth of soft real-time service may be further divided. Similarly, the bandwidth of the best effort services may be further divided between throughput intensive and interactive services.

A key advantage of hierarchical link bandwidth allocation is that it provides isolation between different services while enabling similar services to share resources. Hence, incompatible congestion control algorithms can coexist while compatible algorithms reap the benefits of sharing. For example, while high and low reliability soft real-time services get the benefits of sharing, the hard real-time service is isolated from the overbooking that may occur in soft real-time services, and the congestion control algorithm that may be used by the best effort services. This isolation between services has several advantages:

– Since each service is isolated from the other, different services can be designed independently.

  - Isolation between services allows use of different resource allocation methods for different services. This is desirable as hard real-time service may use a scheduling algorithm that performs well when there is no overbooking; soft real-time services may prefer to use a scheduling algorithm that provides QoS guarantees and/or minimizes deadline violations in presence of overbooking [1]; and best effort services may use a fair scheduler for throughput-intensive flow-controlled data applications.

  Thus, hierarchical link bandwidth allocation is a desirable mechanism for managing heterogeneity in services and protocols and a packet scheduling algorithm should enable it.

- *It should support link-sharing service.* An integrated services network will support a *link-sharing* service in which the bandwidth of a link is allocated among several organizations and the bandwidth of an organization is recursively allocated among its suborganizations [15, 47]. The bandwidth of an organization is allocated among the suborganizations in proportion to the weights of the suborganizations. The weights may be governed by the price paid by the various suborganizations or other policies [15]. Such a link-sharing service has been considered to be the most important service enhancement that is required in current Internet [47].

  Link-sharing can be naturally supported by hierarchical link bandwidth allocation mechanism. Thus, hierarchical link bandwidth allocation is desirable for supporting link-sharing service and a packet scheduling algorithm should enable it.

- *It should be computationally efficient.* Integrated services networks will be high speed in nature and operate at few hundred Mb/s or Gb/s. To facilitate its implementation in such high-speed networks, the scheduling algorithm should be computationally efficient.

Hence, in summary, a scheduling algorithm should: (1) provide QoS guarantees over variable rate servers; (2) support hierarchical link bandwidth allocation to enable a network to provide link-sharing service and facilitate coexistence of multiple services; and (3) be computationally efficient.

4.3 Requirements Summary

From our examination of the application and network requirements, we conclude that a suitable packet scheduling algorithm for integrated services networks should:

1. achieve low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.).
2. provide fairness to support VBR video.
3. provide fairness over variable rate servers to support throughput-intensive flow-controlled data applications.
4. provide QoS guarantees over variable rate servers.
5. support hierarchical link bandwidth allocation to enable a network to provide link-sharing service and facilitate coexistence of multiple services and protocols.
6. be computationally efficient.

Recall that from the analysis of scheduling algorithms in GR we had concluded that scheduling algorithms that are fair and achieve separation of rate and delay allocation are desirable. Since algorithms that achieve separation of rate and delay allocation can provide low delay to low throughput flows, we conclude that the desirable algorithms in GR are also those that are required for integrated services networks.

Since fairness is a central requirement, consider the meaning of fair allocation of link bandwidth. Intuitively, allocation of link bandwidth is fair if in all time intervals equal bandwidth is allocated to all the flows. This concept generalizes to weighted fairness in which the bandwidth must be allocated in proportion to the *weights* of the flows. Formally, if ϕ_f is the weight of flow f and $W_f(t_1, t_2)$ is the aggregate service (in bits) received by it in the interval $[t_1, t_2]$, then an allocation is fair if, for all intervals $[t_1, t_2]$ in which both flows f and m are backlogged, $\frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} = 0$. Clearly, this is an idealized definition of fairness as it assumes that flows can be served in infinitesimally divisible units. The objective of fair *packet* scheduling algorithms is to ensure that $\mid \frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} \mid$ is as close to 0 as possible [1]. However, it has been shown in [22] that if a packet scheduling algorithm guarantees that $\mid \frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} \mid \leq H(f,m)$ for all intervals $[t_1, t_2]$ then $H(f,m) \geq \frac{1}{2}\left(\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m}\right)$, where $H(f,m)$ is a function of the properties of flows f and m, and l_f^{max} and l_m^{max} denote the maximum lengths of packets of flow f and m, respectively. We refer to $H(f,m)$ as the unfairness measure.

Several fair scheduling algorithms that achieve value of $H(f,m)$ close to the lower bound have been proposed in the literature. In the next section, we examine these algorithms to determine if they meet the requirements for a suitable scheduling algorithm.

4.4 Evaluation of Existing Algorithms

The earliest known fair scheduling algorithm is Weighted Fair Queuing (WFQ) [12] (also referred to as Packet-by-Packet Generalized Processor Sharing (PGPS) [42]). WFQ was designed to emulate a hypothetical bit-by-bit weighted round robin server in which the number of bits of a flow served in a round is proportional to the weight of the flow (such a server is also known as GPS server). Since packets can not be serviced a bit at a time, WFQ emulates bit-by-bit round robin by scheduling packets in the increasing order of their departure times in the hypothetical server. To compute this departure order, WFQ associates two tags, a *start tag* and a *finish tag*, with every packet of a flow. Specifically, if p_f^j and l_f^j denote the j^{th} packet of flow f and its length, respectively, and if $A(p_f^j)$ denotes the arrival time of packet p_f^j at the server, then start tag $S(p_f^j)$ and finish tag $F(p_f^j)$ of packet p_f^j are defined as:

$$S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\} \quad j \geq 1$$

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{\phi_f}$$

where $F(p_f^0) = 0$ and $v(t)$ is defined as:

$$\frac{dv(t)}{dt} = \frac{C}{\sum_{j \in B(t)} \phi_j} \tag{4.1}$$

[1]There are other definitions of fairness that have been proposed in the literature. We will elucidate the appropriateness of this definition in Chapter 8.

where C is the capacity of the server and $B(t)$ is the set of backlogged flows at time t in the bit-by-bit round robin server. WFQ then schedules packets in the increasing order of their finish tags.

The implementation of WFQ requires computation of $v(t)$, which in turn requires simulation of bit-by-bit round robin server in real time. This simulation may require processing of $O(Q)$ events in a single packet transmission time, where Q is the number of flows served, and thus is considered computationally expensive [22]. Furthermore, to retain fairness when server rate varies over time, the definition of virtual time will have to be modified. The following examples illustrate that if the definition of virtual time is not modified and is based on the assumption that the capacity of a server is constant, then WFQ becomes unfair over variable rate servers.

Example 4.1 *Let the capacity of the server that WFQ is emulating be C pkts/s, $C > 1$. Let the actual server capacity be 1 pkt/s in [0,1) and C pkt/s in [1,2). Consider two flows f and m both of which have unit length packets and weights of 1 pkt/s. Let flow f send $C+1$ packets at time 0. Hence, for flow f, $F(p_f^j) = j$; $1 \leq j \leq C+1$. Let flow m become backlogged at $t = 1$ and be backlogged during the interval [1,2]. Since only flow f is backloged during [0,1), using (4.1), we get $v(1) = C$. Hence, for flow m, $F(p_m^1) = C+1$. Since WFQ schedules packets in the increasing order of finish tags, we get: $C-1 \leq W_f(1,2) \leq C$ and $W_m(1,2) \leq 1$. However, for fair allocation of bandwidth, $W_f(1,2)$ and $W_m(1,2)$ should both be $C/2$. Since C can be chosen arbitrarily, this example illustrates the unfairness that can result when the actual capacity is lower than the capacity being assumed.*

Example 4.2 *Let the capacity of the server that WFQ is emulating be 1 pkts/s. Let the actual server capacity be* **C** *pkt/s, $C > 1$, in [0,1) and 1 pkt/s in [1,C). Consider two flows f and m both of which have weights of 1 pkt/s. Let flow f send $\frac{3C}{2}$ packets at time 0. Hence, for flow f, $F(p_f^j) = j$; $1 \leq j \leq \frac{3C}{2}$. Let flow m send C packets at time 1. Since only flow f is backloged during [0,1), $v(1) = 1$. Hence, for flow m, $F(p_m^j) = j+1$; $1 \leq j \leq C$. Since WFQ schedules packets in the increasing order of finish tags, while $W_f(1, C+1) \leq 1$, $C-1 \leq W_m(1, C+1) \leq C$. However, for fair allocation of bandwidth, $W_f(1, C+1)$ and $W_m(1, C+1)$ both should be*

$C/2$. Since C can be chosen arbitrarily, this example illustrates the unfairness that can result when the actual capacity is greater than the capacity being assumed.

From the above examples we conclude that to ensure fairness over variable rate servers, the definition of system virtual time should be modified to depend on the time varying server capacity. This can be achieved by defining $v(t)$ as $\frac{dv(t)}{dt} = \frac{C(t)}{\sum_{j \in B(t)} \phi_j}$ where $C(t)$ is the capacity of server at time t. Without a priori knowledge of $C(t)$, computing $v(t)$ based on the new definition requires counting the number of bits transmitted by the server during various intervals as well as continuous evaluation of $v(t)$. The complexity of these operations makes the cost of computing $v(t)$ prohibitive. Thus, we conclude that: (1) if constant rate approximations are employed in WFQ for variable rate servers, then WFQ is unfair, and (2) modified WFQ algorithm that may retain fairness over variable rate servers is computationally prohibitive. Thus, WFQ is unsuitable for achieving fairness over variable rate servers. As we will show in Chapter 7, to be useful for hierarchical link bandwidth allocation [15, 49], a scheduling algorithm must provide fairness over variable rate servers. Consequently, WFQ is unsuitable for meeting two key requirements (i.e., fairness over variable rate servers and support for hierarchical link bandwidth allocation) of a fair scheduling algorithm for integrated services network. Finally, WFQ does not have good fairness properties over constant rate servers. It has been shown in [52] that if all flows are continuously backlogged, then $H(f, m)$ for WFQ is given as:

$$\max \left\{ U_m + \frac{l_{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m}, U_f + \frac{l_{max}}{\phi_m} + \frac{l_f^{max}}{\phi_f} \right\} \tag{4.2}$$

where l_{max} is the maximum packet length served by the server and

$$U_f = \min \left\{ (\mid Q \mid -1) \frac{l_{max}}{\phi_f}, \max_{n \in Q} \{ \frac{l_{max}^n}{\phi_n} \} \right\}$$

where Q is the set of flows served by the server. Since the definition of fairness that we are using does not require flows to be continuously backlogged, (4.2) is a lower bound on $H(f, m)$ for WFQ. Clearly, this is significantly higher than the lower bound on $H(f, m)$. Since in the hypothetical server $\mid \frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} \mid = 0$, this also demonstrates that WFQ does not emulate it "well".

Fair Queuing based on Start-time (FQS), proposed in [27], computes start tag and finish tag of a packet exactly as in WFQ. However, instead of scheduling packets in the increasing order of finish tags, it schedules packets in the increasing order of start tags. Though FQS has advantages for processor scheduling, it is not known to have any advantage over WFQ for scheduling packets in a network. Moreover, since it utilizes $v(t)$ as defined in (4.1), it has disadvantages similar to that of WFQ.

Self Clocked Fair Queuing (SCFQ), originally proposed in [11] and later analyzed in [22], was designed to reduce the computational complexity of fair scheduling algorithms like WFQ. SCFQ also schedules packets in the increasing order of finish tags. However, it achieves efficiency over WFQ by approximating $v(t)$ with the finish tag of the packet in service at time t. It has been shown that the value of $H(f, m)$ for SCFQ is $(\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m})$, which is only a factor of two away from the lower bound [22]. The main limitation of SCFQ is that it increases the maximum delay incurred by the packets significantly. Specifically, if Q is the set of flows served by a server and C its capacity, then as we demonstrated in Chapter 2, packets of flow f may incur $\frac{\sum_{n \in Q \wedge n \neq f} l_n^{max}}{C}$ more delay in SCFQ than in WFQ. This may be unacceptably large in many cases. However, SCFQ is fair over variable rate servers.

Frame-based Fair Queuing (FFQ), proposed in [52], was designed to retain the efficiency of SCFQ in computing the start and finish tags but ensure that the worst-case delay that can be guaranteed to a packet is the same as in WFQ [2]. This objective is achieved in FFQ by: (1) computing start and finish tags as in WFQ, (2) resetting system virtual time $v(t)$, every *frame time* T, to minimum of start tags of the packets that are backlogged, (3) defining $v(t)$ as $v(t) = v(\tau) + (t - \tau)$ where τ is the last time at which system virtual time was reset, and (4) scheduling packets in the increasing order of finish tags. The main limitation of FFQ is that due to its assumption of constant rate servers in computation of $v(t)$, it is unfair over variable rate servers. Furthermore, its $H(f, m)$ value depends on the frame time T, which in turn depends on the minimum rate allocated by a server, and can deviate significantly from the lower bound.

[2] Note that FFQ does not guarantee that the departure time of a packet is the same as in WFQ or WF^{2}Q . It only guarantees that the worst-case delay of a packet that may be incurred when all the flows are continuously backlogged is the same as in WFQ.

Worst-case-fair Fair Queuing (WF^{2}Q), proposed in [4], was designed to improve WFQ's emulation of hypothetical bit-by-bit round robbin server. To achieve this objective, WF^{2}Q : (1) utilizes $v(t)$ as defined in (4.1) and computes start and finish tags as in WFQ, (2) defines a packet to be eligible at time t only if its start tag is at most $v(t)$, and (3) schedules eligible packets in the increasing order of finish tags. It has been shown that WF^{2}Q emulates the hypothetical server well and has a $H(f, m)$ value of $(\frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m})$ [4]. However, since it utilizes $v(t)$ as defined in (4.1), it is computationally inefficient and unfair over variable rate servers.

WF^{2}Q+ has been recently, independent of our work, proposed to reduce the implementation complexity of WF^{2}Q while retaining several of its properties (a similar, but not identical, algorithm termed Starting Potential based Fair Queuing was proposed in [51]) [3]. It defines start tag of packet p_f^j to be the finish tag of packet p_f^{j-1}, i.e., $S(p_f^j) = F(p_f^{j-1})$, if flow f is backlogged on arrival of p_f^j; otherwise, $S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\}$. The finish tag of a packet and the set of eligible packets are defined as in WF^{2}Q but $v(t)$ is defined as $v(t) = \max\{v(\tau) + t - \tau, \min_{n \in B(t)} S(p_n^k)\}$ where τ is the largest time less than t at which a packet finished service, p_n^k is the packet at the head of the queue of flow n at time t, and $B(t)$ is the set of backlogged flows at time t. WF^{2}Q+ , like WF^{2}Q , schedules eligible packets in increasing order of finish tags. Though *worst-case fairness* of WF^{2}Q+ has been derived, its fairness measure has not been derived in[3] [3]. To ensure that properties of WF^{2}Q+ hold over variable rate servers, it has been proposed in [3] that *reference time*, instead of real time, should be used in virtual time computation. Reference time at real time t, $T_R(t)$, is defined as $T_R(t) = \frac{W(0,t)}{C}$ where C is capacity of the server and $W(0, t)$ is the work done by the server in interval $[0, t]$. Given no a priori information regarding variation in server capacity, it appears that determining $W(0, t)$ will require counting the number of bits that have been transmitted by the server in the interval $[0, t]$; this computation can be expensive (the simple implementation presented in [3] is inconsistent with its definition). Furthermore, WF^{2}Q+ has been studied under the assumption that $\sum_{n \in Q} \phi_n \leq C$ where C is the minimum capacity of a server. The following example demonstrates this assumption is necessary to ensure the fairness of WF^{2}Q+

[3]As discussed in Chapter 8, an algorithm that has bounded worst-case-fairness may have unbounded fairness measure.

.

Example 4.3 *Let a server serve packets at a constant rate of $K+1pkt/s$ in $[0,1]$ and then at the constant rate of $2pkt/s$. Thus, C is 2 pkt/s. Let the server serve $K+2$ flows and let each flow be assigned a weight of $1pkt/s$. Let flows $1..K$ terminate after sending one packet each at time 0 and let flow $K+1$ send infinite number of packets. Also let flow $K+2$ send one packet at time $t=1$. Now for all $n \in [1..K]$, $F(p_n^1)=1$. The finish tags of flow $K+1$ packets are given as $F(p_{K+1}^j)=j$. Since $K+1$ packets are served by time 1, $v(1)=T_R(1)=\frac{K+1}{2}$. Thus, $F(p_{K+2}^1)=\frac{K+1}{2}+1$. Since the first packet of flows $1..K+1$ are eligible at time 0 and WF^{2}Q+ schedules packets in the increasing order of finish tags, first packet of flows $1..K+1$ will be served in the time interval $[0,1]$. For ease of exposition of the later part of the schedule, let $q=\lceil\frac{K+1}{2}\rceil$. Then, since $S(p_{K+1}^q)=q-1$ and $v(1)>q-1$, packets $p_{K+1}^2,..p_{K+1}^q$ are eligible for scheduling at time 1. Furthermore, since $F(p_{K+1}^q)<F(p_{K+2}^1)$, in the interval $[1,1+\frac{q-1}{2}]$, $q-1$ packets of flow $K+1$ will be scheduled. Thus in the interval $[1,1+\frac{q-1}{2}]$ even though flows $K+1$ and $K+2$ are backlogged, whereas $q-1$ packets of flow $K+1$ are served, no packet of flow $K+2$ is served. By choosing K, and hence q, appropriately, the difference in the service received by flows $K+1$ and $K+2$ can be made arbitrarily large.*

$\sum_{n\in Q}\phi_i \le C$ may be ensured either by dynamically changing the weight assignments of flows or by performing admission control. An algorithm for dynamically changing the weights or an evaluation of its effects on the fairness properties have not been presented in [3]. On the other hand, it may not be possible to perform admission control for some flow types (for example, best-effort flows). Furthermore, it may not be feasible to employ admission control when minimum server capacity is zero. Hence, a scheduling algorithm that does not require $\sum_{n\in Q}\phi_i \le C$ is desirable.

Recently, Leap Forward Virtual Clock has been proposed in [53]. This algorithm also assumes constant rate servers, and hence may not be fair over variable rate servers. However, it has lower implementation complexity than most of the fair algorithms proposed in the literature.

WFQ, FQS, SCFQ, FFQ, WF^{2}Q , and WF^{2}Q+ sort and schedule packets in the increasing order of finish tags. Hence, per packet computational complexity is $O(logQ)$ where Q is the number of flows served by the server. To reduce this per packet computational complexity, Deficit Round Robin (DRR) was proposed in [50]. It is a derivative of weighted round robin algorithm designed to accommodate variable length packets of a flow. Though the per packet computational complexity of DRR is $O(1)$ per packet and it is fair over variable rate servers, it has the following two limitations:

1. The value of $H(f, m)$ for DRR is $\left(1 + \frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m}\right)$, when $\min_{n \in Q}\{\phi_n\} = 1$, which can deviate arbitrarily away from that of fair scheduling algorithms like SCFQ. For instance, if $\phi_f = \phi_m = 100$ and $l_f^{max} = l_m^{max} = 1$, then $H(f, m)$ for DRR is 1.02, which is 50 times larger than the corresponding 0.02 value for SCFQ. Clearly, appropriate choice of weights can make this factor as high as desired. Since absolute values of $H(f, m)$ have no physical meaning, the relative values are important.

2. The maximum delay incurred by packets serviced by a DRR server can be arbitrarily high. To observe this, consider a weighted round robin server with all packets of size l (DRR is equivalent to weighted round robin server in such a scenario). It is easily observed that a lower bound on maximum delay of a packet of flow f is $\frac{\sum_{n \in Q \wedge n \neq f} l * \phi_n}{C}$. Since weights can be arbitrary, the lower bound can be arbitrarily high.

A common characteristic of all these fair algorithms is that they allocate only rate. The delay of a packet of a flow in these algorithms is inversely related to the rate of a flow. Consequently, they provide high delay to low throughput flows. Since low throughput flows (such as audio and telnet flows) require low delay, this behaviour is undesirable.

In summary, the design of a fair scheduling algorithm that is: (1) computationally efficient, (2) provides fairness over variable rate servers, (3) facilitates hierarchical link bandwidth allocation, and (4) has good delay properties is an open problem.

4.5 Conclusions

In this chapter, we examined the requirements for a suitable packet scheduling algorithm for integrated services network. Our analysis demonstrated that a suitable packet scheduling algorithm should: (1) achieve low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provide fairness for VBR video; (3) provide fairness over variable rate servers for throughput-intensive flow-controlled data applications; (4) provide QoS guarantees over variable rate servers; (5) support hierarchical link bandwidth allocation to enable a network to provide link-sharing service and facilitate coexistence of multiple services and protocols; and (6) be computationally efficient. We demonstrated that none of the existing algorithms meet these requirements.

In the following chapters, we design Start-time Fair Queuing (SFQ) and Fair Airport (FA) algorithms and employ them to design Hierarchical SFQ and FA schedulers. Our analysis demonstrates that these algorithms meet the requirements for suitable scheduling algorithms for integrated services networks.

Chapter 5

Start-time Fair Queuing

In this chapter, we design Start-time Fair Queuing (SFQ) algorithm to meet the requirements outlined in Chapter 4. We demonstrate that SFQ meets the requirements since it: (1) achieves low average as well as maximum delay for low-throughput applications (e.g., interactive audio, telnet, etc.); (2) provides fairness which is desirable for VBR video; (3) provides fairness, regardless of variation in server capacity, for throughput-intensive, flow-controlled data applications; (4) provides QoS guarantees over variable rate servers; (5) enables hierarchical link bandwidth allocation; and (6) is computationally efficient.

The rest of the chapter is structured as follows. We design SFQ algorithm in Section 5.1. We analyze its fairness and throughput properties in Sections 5.2 and 5.3, respectively. We derive single server deadline properties in Section 5.4 and extend them to multiple servers in Section 5.5. Section 5.6 discusses various aspects of SFQ. Finally, the results of this chapter are summarized in Section 5.7.

5.1 Design of Start-time Fair Queuing

The design of SFQ is based on two intuitive observations:

- To efficiently achieve fairness regardless of variation in a server capacity, an algorithm should not make any assumptions about server capacity. If the algorithm assumes fixed server capacity, then as shown in Chapter 4, it may become unfair over variable rate servers. On the other hand, if it assumes that instantaneous server capacity is known, then due to the complexity of tracking the capacity, the implementation may be inefficient.

- To provide low delay to low throughput flows, an algorithm should not schedule packets in increasing order of finish tags. This is because the finish tag of a packet is inversely proportional to its weight. Since low throughput flows have smaller weights, their packets have larger finish tags than packets of high throughput flows. Consequently, if the algorithm schedules packets in increasing order of finish tags, then it will give priority to high throughput flows. This will result in low delay for high throughput and high delay for low througput flows. This is just the opposite of the requirement that a scheduling algorithm should provide low delay to low throughput flows.

  One way of addressing this limitation is to schedule packets in increasing order of *start tags*. Since start tag of a packet is not inversely related to the througput of the flow, low throughput flows can be provided lower delays.

Recall that SCFQ does not make any assumptions regarding server capacity. Hence, it is a candidate algorithm for achieving fairness regardless of variation in a server capacity. However, it schedules packets in increasing order of finish tags. Consequently, it will provide high delay to low throughput flows. We can potentially remedy this drawback by using the same start tag and finish tag computations as in SCFQ but scheduling packets in increasing order of start tags. However, in such a case since the virtual time function is defined as the finish tag of packet in service, it may become non-monotonic in time. To observe this, consider the following example:

Example 5.1 *Consider two flows f and m with weights $1pkt/s$ and $2pkt/s$, respectively. Let both flows send one packet at time 0. Then, $S(p_f^1) = S(p_m^1) = 0$, $F(p_f^1) = 1$, and $F(p_m^1) = 0.5$. Let p_f^1 be scheduled before p_m^1. Then, the virtual time is 1 when packet p_f^1 is in service and 0.5 when p_m^1 begins service.*

To make virtual time function monotonic, we can define it to be the *start tag*, rather than finish tag, of packet in service. This results into the Start-time Fair Queuing (SFQ) algorithm. SFQ schedules packets in increasing of start tags of packets and defines virtual time to be the start tag of packet in service. The formal specification of the algorithm is as follows:

1. On arrival, a packet p_f^j is stamped with start tag $S(p_f^j)$, computed as:

   $$S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\} \quad j \geq 1 \tag{5.1}$$

   where $F(p_f^j)$, the finish tag of packet p_f^j, is defined as:

   $$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{\phi_f} \quad j \geq 1 \tag{5.2}$$

   where $F(p_f^0) = 0$ and ϕ_f is the weight of flow f.

2. Initially the server virtual time is 0. During a busy period, the server virtual time at time t, $v(t)$, is defined to be equal to the start tag of the packet in service at time t. At the end of a busy period, $v(t)$ is set to the maximum of finish tag assigned to any packets that have been serviced by then[1].

3. Packets are serviced in increasing order of the start tags; ties are broken arbitrarily (some tie breaking rules may be more desirable than others and are discussed in Section 5.4).

As is evident from the definition, the computation of $v(t)$ in SFQ is inexpensive since it only involves examining the start tag of packet in service. Consequently, computation of finish and start tags is inexpensive. Hence, the main complexity of SFQ lies in scheduling packets in increasing order of start tags. Since SFQ schedules packets of flow in FIFO order, the scheduling order can be determined by a priority queue that contains only the packets that are at the head of their respective flow queues. Hence, if there are Q flows at the server, all the priority queue operations can be performed in time $O(logQ)$ [7]. Thus, per packet complexity of SFQ is $O(logQ)$.

[1]Observe that server virtual time changes only when a packet finishes service. Also, we set $v(t)$ to the maximum of finish tags of the packets at the end of busy period only for clarity of proofs; all the start tags as well as the server virtual time can be equivalently set to zero. Resetting start tags and virtual time to 0 may be desirable from an implementation perspective.

One of the requirements for scheduling algorithms that we had identified was that they should provide QoS guarantees over variable rate servers. Clearly, a server can not provide bound on the departure time of a packet if unbounded variation in server capacity can occur. Hence, we analyze for servers with bounded fluctuation in service rate.

Two server models, termed *Fluctuation Constrained (FC)* server and *Exponentially Bounded Fluctuation (EBF)* server, that have bounded fluctuation in service rate and are suitable for modeling many variable rate servers have been introduced in [37] [2]. A FC server has two parameters; average rate C (bits/s) and burstiness $\delta(C)$ (s). Intuitively, in a FC server the time taken to serve packets of aggregate length w in a busy period, exceeds the time taken in an equivalent constant rate server by at most $\delta(C)$. Formally,

Definition 5.1 *A server is a Fluctuation Constrained (FC) server with parameters* $(C, \delta(C))$*, if the time taken to serve packets of aggregate length* w *in a busy period, denoted by* $T(w)$*, satisfies:*

$$T(w) \leq \frac{w}{C} + \delta(C) \tag{5.3}$$

EBF server is a stochastic relaxation of FC server. Intuitively, in an EBF server the probability of the time taken to serve packets of aggregate length w in a busy period deviating by more than γ from that in an equivalent constant rate server, decreases exponentially with γ. Formally,

Definition 5.2 *A server is an Exponentially Bounded Fluctuation (EBF) server with parameters* $(C, B, \alpha, \delta(C))$*, if the time taken to serve packets of aggregate length* w *in a busy period, denoted by random variable* $T(w)$*, satisfies:*

$$P(T(w) > \frac{w}{C} + \delta(C) + \gamma) \leq Be^{-\alpha\gamma} \quad 0 \leq \gamma \tag{5.4}$$

In what follows, we analyze the fairness of SFQ for any variable rate server, and its throughput and deadline guarantees for FC and EBF servers. Since a $(C, 0)$ FC server is a constant rate server, the following analysis is also valid for constant rate servers.

[2] The definitions of FC and EBF servers as presented here are different from that in [37]. Specifically, whereas [37] characterizes the servers by the work done in a busy period, we characterize the servers by the time taken to serve packets of length w in a busy period.

5.2 Fairness Guarantee

To derive fairness guarantee of SFQ, we need to prove a bound on $\mid \frac{W_f(t_1,t_2)}{\phi_f} - \frac{W_m(t_1,t_2)}{\phi_m} \mid$ for any interval in which both flows f and m are backlogged. We achieve this objective by establishing a lower and an upper bound on $W_f(t_1, t_2)$ in Lemmas 5.1 and 5.2, respectively.

Lemma 5.1 *If flow f is backlogged throughout the interval $[t_1, t_2]$, then in a SFQ server:*

$$\phi_f(v_2 - v_1) - l_f^{max} \leq W_f(t_1, t_2) \tag{5.5}$$

where $v_1 = v(t_1)$ and $v_2 = v(t_2)$.

Proof: Since $W_f(t_1, t_2) \geq 0$, if $\phi_f(v_2 - v_1) - l_f^{max} \leq 0$, (5.5) holds trivially. Hence, consider the case where $\phi_f(v_2 - v_1) - l_f^{max} > 0$, i.e., $v_2 > v_1 + \frac{l_f^{max}}{\phi_f}$. Let packet p_f^k be the first packet of flow f, that receives service in the open interval (v_1, v_2). To observe that such a packet exists, consider the following two cases:

- Packet p_f^n such that $S(p_f^n) < v_1$ and $F(p_f^n) > v_1$ exists: Since flow f is backlogged in $[t_1, t_2]$, we conclude $v(A(p_f^{n+1})) \leq v_1$. From (5.1) and (5.2), we get:

$$S(p_f^{n+1}) = F(p_f^n)$$

  Since $F(p_f^n) \leq S(p_f^n) + \frac{l_f^{max}}{\phi_f}$ and $S(p_f^n) < v_1$, we get:

$$S(p_f^{n+1}) < v_1 + \frac{l_f^{max}}{\phi_f} \tag{5.6}$$

$$< v_2 \tag{5.7}$$

  Since $S(p_f^{n+1}) = F(p_f^n) > v_1$, using (5.7), we conclude $S(p_f^{n+1}) \in (v_1, v_2)$.

- Packet p_f^n such that $S(p_f^n) = v_1$ exists: p_f^n may finish service at time $t < t_1$ or $t \geq t_1$. If it finishes service at $t < t_1$, then since flow f is backlogged at time t_1, $v(A(p_f^{n+1})) \leq v_1$. If it finishes service at $t \geq t_1$, then since since flow f is backlogged at time t, p_f^{n+1} arrives before p_f^n finishes service and hence $v(A(p_f^{n+1})) \leq v_1$. Hence, in either case,

$S(p_f^{n+1}) = F(p_f^n)$. Since $F(p_f^n) \leq S(p_f^n) + \frac{l_f^{max}}{\phi_f}$ and $S(p_f^n) = v_1$, we get:

$$\begin{aligned} S(p_f^{n+1}) &\leq v_1 + \frac{l_f^{max}}{\phi_f} && (5.8) \\ &< v_2 && (5.9) \end{aligned}$$

Since $S(p_f^{n+1}) = F(p_f^n) > v_1$, using (5.9), we conclude $S(p_f^{n+1}) \in (v_1, v_2)$.

Since either of the two cases always holds, we conclude that packet p_f^k such that $S(p_f^k) \in (v_1, v_2)$ exists. Furthermore, from (5.6) and (5.8), we get:

$$S(p_f^k) \leq v_1 + \frac{l_f^{max}}{\phi_f} \tag{5.10}$$

Let p_f^{k+m} be the last packet to receive service in the virtual time interval (v_1, v_2). Hence,

$$F(p_f^{k+m}) \geq v_2 \tag{5.11}$$

From (5.10) and (5.11), we conclude:

$$F(p_f^{k+m}) - S(p_f^k) \geq (v_2 - v_1) - \frac{l_f^{max}}{\phi_f} \tag{5.12}$$

But since flow f is backlogged in the interval (v_1, v_2), from (5.1) and (5.2) we know:

$$\begin{aligned} F(p_f^{k+m}) &= S(p_f^k) + \sum_{n=0}^{n=m} \frac{l_f^{k+n}}{\phi_f} \\ F(p_f^{k+m}) - S(p_f^k) &= \sum_{n=0}^{n=m} \frac{l_f^{k+n}}{\phi_f} && (5.13) \end{aligned}$$

Hence, from (5.13) and (5.12), we get:

$$\begin{aligned} \sum_{n=0}^{n=m} \frac{l_f^{k+n}}{\phi_f} &\geq (v_2 - v_1) - \frac{l_f^{max}}{\phi_f} \\ \sum_{n=0}^{n=m} l_f^{k+n} &\geq \phi_f (v_2 - v_1) - l_f^{max} \end{aligned}$$

Since $S(p_f^{k+m}) < v_2$, packet p_f^{k+m} is guaranteed to have been transmitted by t_2. Hence, $W_f(t_1, t_2) \geq \sum_{n=0}^{n=m} l_f^{k+n}$, and the lemma follows. ■

Lemma 5.2 *In a SFQ server, during any interval* $[t_1, t_2]$*:*

$$W_f(t_1, t_2) \leq \phi_f(v_2 - v_1) + l_f^{max} \tag{5.14}$$

where $v_1 = v(t_1)$ *and* $v_2 = v(t_2)$.

Proof: From the definition of SFQ, the set of flow f packets served in the interval $[v_1, v_2]$ have start tag at least v_1 and at most v_2. Hence, the set can be partitioned into two sets:

- Set D consisting of packets that have start tag at least v_1 and finish tag at most v_2. Formally,

  $$D = \{k | v_1 \leq S(p_f^k) \leq v_2 \wedge F(p_f^k) \leq v_2\}$$

  From (5.1) and (5.2), we conclude:

  $$\sum_{k \in D} l_f^k \leq \phi_f(v_2 - v_1) \tag{5.15}$$

- Set E consisting of packets that have start tag at most v_2 and finish tag greater than v_2. Formally,

  $$E = \{k | v_1 \leq S(p_f^k) \leq v_2 \wedge F(p_f^k) > v_2\} \tag{5.16}$$

  Clearly, at most one packet can belong to this set. Hence,

  $$\sum_{k \in E} l_f^k \leq l_f^{max} \tag{5.17}$$

From (5.15) and (5.17) we conclude that (5.14) holds. ■

Since unfairness between two flows in any interval is maximum when one flow receives maximum possible service and the other minimum service, Theorem 5.1 follows directly from Lemmas 5.1 and 5.2.

Theorem 5.1 *For any interval* $[t_1, t_2]$ *in which flows* f *and* m *are backlogged during the entire interval, the difference in the service received by two flows at a SFQ server is given as:*

$$\left| \frac{W_f(t_1, t_2)}{\phi_f} - \frac{W_m(t_1, t_2)}{\phi_m} \right| \leq \frac{l_f^{max}}{\phi_f} + \frac{l_m^{max}}{\phi_m}$$

Using Theorem 5.1 and the analysis presented in [52] we conclude that SFQ has unfairness measure that is at least as good as that of all the known scheduling algorithms.

There are two important aspects of Theorem 5.1:

- To establish it, we did not make any assumptions about the service rate of the server. Hence, it holds regardless of the characteristics of the server. This demonstrates that SFQ achieves fair allocation of bandwidth over variable rate servers, and thus meets a fundamental requirement of fair scheduling algorithms for integrated services networks.

- To establish it, we did not make any assumptions about the weights; weights are just uninterpreted numbers. In particular, we did not require any admission control such as $\sum_{n \in Q} \phi_n \leq C$. Since for variable rate servers, C may not always be defined, as well as it may not be possible to perform admission control for best-effort flows, this is an important difference between SFQ and algorithms such as $\mathrm{WF}^2\mathrm{Q}$+ and FFQ that require admission control to ensure fairness.

To experimentally evaluate the relative performance of SFQ and WFQ over variable rate servers, we simulated 3 flows with the same destination traversing a single switch using REAL network simulator (see Figure 5.1). Source 1 transmitted a MPEG compressed VBR video sequence with average rate 1.21 Mb/s using 50 bytes packets [3]. Sources 2 and 3 were TCP Reno sources and used 200 bytes packets. The link capacity between the switch and the destination was 2.5 Mb/s. The scheduling algorithm at the switch gave higher priority to source 1 packets and scheduled source 2 and 3 packets using either WFQ or SFQ. Consequently, the output link at the switch appeared as a variable rate server to sources 2 and 3. The WFQ implementation used the link capacity to compute the finish tags. Source 3 was made active 500 ms after sources 1 and 2, and the network was simulated for one second. Figure 5.2 plots the sequence number of packets of sources 2 and 3 received by the destination when WFQ and SFQ were used. As the figure demonstrates, source 2 received unfair advantage over source 3 when WFQ was used. Specifically, when WFQ was used, during the 500ms interval after source 3 became active, whereas the destination received only 48 packets from source 3, it received

[3]The video sequence was derived by digitizing and compressing television serial Frasier.

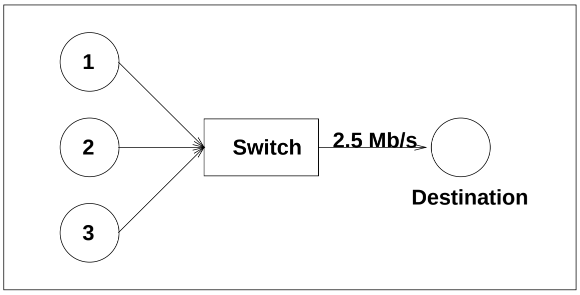


Figure 5.1: The network topology used for the simulations

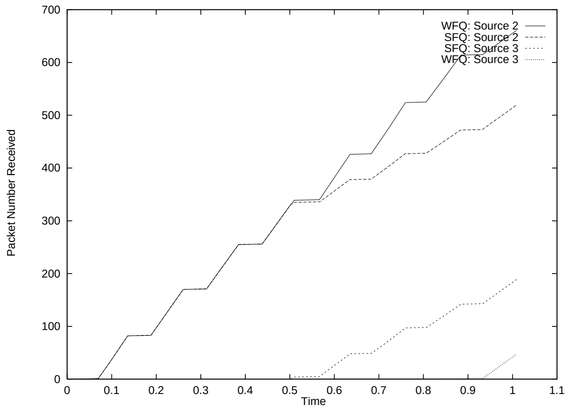


Figure 5.2: Comparison of the number of packets received by sources 2 and 3 in WFQ and SFQ

331 packets from source 2. On the other hand, when SFQ was used, it received 189 packets and 190 packets from sources 2 and 3, respectively. Furthermore, during the first 435 ms after source 3 was started, the number of source 3 packets received by the destination in WFQ and SFQ was 2 and 145, respectively.

5.3 Throughput Guarantee

In the previous sections, we have not assigned any interpretation to the weight of a flow. To establish the throughput and deadline guarantee of a flow, we will henceforth interpret ϕ_f as the rate assigned to flow f and denote ϕ_f by r_f. Theorems 5.2 and 5.3 establish the throughput guaranteed to a flow by a SFQ FC and EBF server, respectively, when appropriate admission control procedures are used.

Theorem 5.2 *If Q is the set of flows served by a SFQ FC server with parameters $(C, \delta(C))$, and $\sum_{n \in Q} r_n \leq C$, then for all intervals $[t_1, t_2]$ in which flow f is backlogged throughout the interval, $W_f(t_1, t_2)$ is given as:*

$$W_f(t_1, t_2) \geq r_f(t_2 - t_1) - r_f \frac{\sum_{n \in Q} l_n^{max}}{C} - r_f \delta(C) - l_f^{max} \tag{5.18}$$

Proof: Let $v(t_1) = v_1$ and let $\widehat{W}(v_1, v_2)$ denote the aggregate length of packets served by the server in the virtual time interval $[v_1, v_2]$. Then, from Lemma 5.2 we conclude:

$$\widehat{W}(v_1, v_2) \leq \sum_{n \in Q} r_n(v_2 - v_1) + \sum_{n \in Q} l_n^{max}$$

Since $\sum_{n \in Q} r_n \leq C$,

$$\widehat{W}(v_1, v_2) \leq C(v_2 - v_1) + \sum_{n \in Q} l_n^{max} \tag{5.19}$$

Define v_2 as:

$$v_2 = v_1 + t_2 - t_1 - \frac{\sum_{n \in Q} l_n^{max}}{C} - \delta(C) \tag{5.20}$$

Then from (5.19), we conclude:

$$\begin{aligned}
\widehat{W}(v_1, v_2) &\leq C(v_1 + t_2 - t_1 - \frac{\sum_{n \in Q} l_n^{max}}{C} - \delta(C) - v_1) + \sum_{n \in Q} l_n^{max} \\
&\leq C(t_2 - t_1 - \delta(C))
\end{aligned}$$

Let $\widehat{t_2}$ be such that $v(\widehat{t_2}) = v_2$. Also, let $T(w)$ be the time taken by server to serve packets with aggregate length w in its busy period. Then,

$$\begin{aligned}
\widehat{t_2} &\leq t_1 + T(\widehat{W}(v_1, v_2)) \\
&\leq t_1 + T(C(t_2 - t_1 - \delta(C))) \qquad (5.21)
\end{aligned}$$

From the definition of Fluctuation Constrained server, we know:

$$T(w) \leq \frac{w}{C} + \delta(C) \tag{5.22}$$

From (5.21) and (5.22) we get:

$$\begin{aligned}\widehat{t_2} &\leq t_1 + \frac{C(t_2 - t_1 - \delta(C))}{C} + \delta(C) \\ &\leq t_2 \qquad (5.23)\end{aligned}$$

From Lemma 5.1 we know that:

$$W_f(t_1, \widehat{t_2}) \geq r_f(v_2 - v_1) - l_f^{max}$$

Since $\widehat{t_2} \leq t_2$, using (5.20) we get:

$$W_f(t_1, t_2) \geq r_f(t_2 - t_1) - r_f \frac{\sum_{n \in Q} l_n^{max}}{C} - r_f \delta(C) - l_f^{max}$$

■

Theorem 5.3 *If Q is the set of flows served by a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$, and $\sum_{n \in Q} r_n \leq C$, then for all intervals $[t_1, t_2]$ in which flow f is backlogged throughout the interval, $W_f(t_1, t_2)$ is given as:*

$$P\left(W_f(t_1, t_2) < r_f(t_2 - t_1) - r_f \frac{\sum_{n \in Q} l_n^{max}}{C} - r_f \delta(C) - r_f \gamma - l_f^{max}\right) \leq B e^{-\alpha \gamma}$$

Proof: Let $v(t_1) = v_1$ and let $\widehat{W}(v_1, v_2)$ denote the aggregate length of packets served by the server in the virtual time interval $[v_1, v_2]$. Then, from Lemma 5.2 we conclude:

$$\widehat{W}(v_1, v_2) \leq \sum_{n \in Q} r_n(v_2 - v_1) + \sum_{n \in Q} l_n^{max}$$

Since $\sum_{n \in Q} r_n \leq C$,

$$\widehat{W}(v_1, v_2) \leq C(v_2 - v_1) + \sum_{n \in Q} l_n^{max} \qquad (5.24)$$

Define v_2 as:

$$v_2 = v_1 + t_2 - t_1 - \frac{\sum_{n \in Q} l_n^{max}}{C} - \delta(C) - \eta \qquad (5.25)$$

Then from (5.24), we conclude:

$$\begin{aligned}\widehat{W}(v_1, v_2) &\leq C(v_1 + t_2 - t_1 - \frac{\sum_{n \in Q} l_n^{max}}{C} - \delta(C) - \eta - v_1) + \sum_{n \in Q} l_n^{max} \\ &\leq C(t_2 - t_1 - \delta(C) - \eta)\end{aligned}$$

Let $\hat{t_2}$ be such that $v(\hat{t_2}) = v_2$. Also, let $T(w)$ be random variable denoting the time taken by server to serve packets with aggregate length w in its busy period. Then:

$$
\begin{aligned}
P(\hat{t_2} \leq t_2) &\geq P(t_1 + T(\widehat{W}(v_1, v_2)) \leq t_2) \\
&\geq P(T(\widehat{W}(v_1, v_2)) \leq (t_2 - t_1))
\end{aligned}
$$

From the definition of EBF server, we know:

$$P(T(C(t_2 - t_1 - \delta(C) - \eta)) \leq (t_2 - t_1)) \geq 1 - Be^{-\alpha\eta}$$

Hence,

$$P(\hat{t_2} \leq t_2) \geq 1 - Be^{-\alpha\eta} \tag{5.26}$$

From Lemma 5.1 we know that $P\left(W_f(t_1, t_2) \geq r_f(v_2 - v_1) - l_f^{max}\right) \geq P(\hat{t_2} \leq t_2)$. Hence, using (5.26) we get:

$$P\left(W_f(t_1, t_2) \geq r_f(v_2 - v_1) - l_f^{max}\right) \geq 1 - Be^{-\alpha\eta}$$

From (5.25) (the definition of v_2) we get:

$$
\begin{aligned}
P\left(W_f(t_1, t_2) \geq r_f(t_2 - t_1) - r_f\frac{\sum_{n \in Q} l_n^{max}}{C} - r_f\delta(C) - r_f\eta - l_f^{max}\right) &\geq 1 - Be^{-\alpha\eta} \\
P\left(W_f(t_1, t_2) < r_f(t_2 - t_1) - r_f\frac{\sum_{n \in Q} l_n^{max}}{C} - r_f\delta(C) - r_f\eta - l_f^{max}\right) &\leq Be^{-\alpha\eta}
\end{aligned}
$$

■

The throughput guarantee established in Theorem 5.2 is the same as Bit Worst-case Fairness Index property defined in [3]. It can be shown that for constant rate servers, when similar admission control procedure is used, the throughput guarantee of WFQ and SCFQ is similar to that of SFQ.

5.4 Deadline Guarantee

SFQ algorithm, as defined so far, only allocates constant rate to the packets of a flow. However, due to the multiple time-scale variation of VBR video, to achieve efficient utilization of network resources, a server may be required to allocate variable rate to packets of a video flow. To support variable rate allocation, we generalize SFQ by extending the definition of the tags. Let r_f^j be the rate assigned to packet p_f^j. Then finish tag of packet p_f^j, $F(p_f^j)$, is defined as:

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{r_f^j} \quad j \geq 1 \tag{5.27}$$

Start tag of a packet and the system virtual time are defined as before.

We show in Sections 5.4.1 and 5.4.2 that the generalized SFQ algorithm provides two types of deadline guarantees to a packet:

- *Delay guarantee*: This is a deadline guarantee for a packet based on a packet based on its *expected arrival time* which we had introduced in Chapter 2. Specifically, SFQ guarantees that:

  $$L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \alpha_f^j$$

  where $L_{SFQ}(p_f^j)$ is the departure time of packet p_f^j in a SFQ server, α_f^j depends on l_f^j and the properties of the server as well as the other flows at the server, and $EAT(p_f^j, r_f^j)$ is the expected arrival time of packet p_f^j that has been assigned rate r_f^j. $EAT(p_f^j, r_f^j)$ is defined as in Chapter 2, i.e.,:

  $$EAT(p_f^j, r_f^j) = \max\{A(p_f^j), EAT(p_f^{j-1}, r_f^{j-1}) + \frac{l_f^{j-1}}{r_f^{j-1}}\} \quad j \geq 1 \tag{5.28}$$

  where $EAT(p_f^0, r_f^0) = -\infty$. Hence, SFQ belongs to the GR class and has all the desirable properties of the GR scheduling algorithms.

- *Delay-cum-throughput guarantee*: This is a deadline to a packet based on its arrival time and the departure time of the previous packet which we had introduced in Chapter 3. Specifically, SFQ guarantees that:

  $$L_{SFQ}(p_f^j) \leq \max\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\} + \beta_f^j$$

SFQ provides these deadline guarantees when the aggregate rate reserved for all the flows is less than the server capacity. To derive the deadline guarantee, let us first define the intuitive meaning of the term “capacity is not exceeded”. Let rate function for flow f at virtual time v, denoted by $R_f(v)$, be defined as the rate assigned to the packet that has start tag less than v and finish tag greater than v. Formally,

$$R_f(v) = \begin{cases} r_f^j & \text{if } \exists j \ni \left(S(p_f^j) \leq v < F(p_f^j)\right) \\ 0 & \text{otherwise} \end{cases}$$

Let Q be the set of flows served by the server. Then the capacity of a FC or EBF server with average rate C, is not exceeded if:

$$\sum_{n \in Q} R_n(v) \leq C \quad v \geq 0$$

To derive the delay as well as delay-cum-throughput guarantee of FC and EBF SFQ servers, we first derive a bound on the work done by a SFQ server in virtual time interval $[v_1, v_2]$ in Lemma 5.3. The proof of Lemma 5.3 is very similar to that of Lemma 5.2.

Lemma 5.3 *If the capacity of a FC or EBF server with parameters $(C, \delta(C))$ or $(C, B, \alpha, \delta(C))$, respectively, is not exceeded, then the aggregate length of packets that have start tag at least v_1 and at most v_2 and are served in the same busy period, denoted by $\widehat{W}(v_1, v_2)$, is given by:*

$$\widehat{W}(v_1, v_2) \leq C \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j$$

whenever $v_1 = S(p_f^k, r_f^k)$, $v_2 = S(p_f^j, r_f^j)$ and $v_2 - v_1 = \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}}$

Proof: The set of packets that have start tag at least v_1 and at most v_2 can be partitioned into two sets:

- A set consisting of packets that have start tag at least v_1 and finish tag at most v_2. Formally, the set of packets of flow n, denoted by D_n, in this set is:

$$D_n = \{m | v_1 \leq S(p_n^m) \leq v_2 \wedge F(p_n^m) \leq v_2\} \tag{5.29}$$

  Then, from the definition of $R_n(v)$ and $F(p_n^m, r_f^m)$, we know that the cumulative length of such flow n packets served by the server in the virtual time interval $[v_1, v_2]$, denoted by $AP_n(v_1, v_2)$, is given as:

$$AP_n(v_1, v_2) \leq \int_{v_1}^{v_2} R_n(v)dv$$

Hence, aggregate length of packets in this set, $\sum_{n \in Q} AP_n(v_1, v_2)$, is given as:

$$\begin{aligned}
\sum_{n \in Q} AP_n(v_1, v_2) &\leq \sum_{n \in Q} \int_{v_1}^{v_2} R_n(v)dv \\
&\leq \int_{v_1}^{v_2} \sum_{n \in Q} R_n(v)dv \\
&\leq \int_{v_1}^{v_2} Cdv \\
&\leq C(v_2 - v_1)
\end{aligned}$$

But $v_2 - v_1 = \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}}$. Hence,

$$\sum_{n \in Q} AP_n(v_1, v_2) \leq C \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}}$$

- A set consisting of packets that have start tag at most v_2 and finish tag greater than v_2. Formally, the set of packets of flow n, denoted by E_n, in this set is:

$$E_n = \{m | v_1 \leq S(p_n^m) \leq v_2 \wedge F(p_n^m) > v_2\}$$

Clearly, at most one packet of flow n can belong to this set. Furthermore, $E_f = \{j\}$. Hence, the maximum aggregate length of packets in this set is:

$$\sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j$$

Hence, $\widehat{W}(v_1, v_2)$, is:

$$\widehat{W}(v_1, v_2) \leq C \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j$$

■

5.4.1 Delay Guarantee

Theorems 5.4 and 5.5 establish the delay guarantee of SFQ for FC and EBF servers, respectively.

Theorem 5.4 *If the capacity of a SFQ FC server with parameters $(C, \delta(C))$ is not exceeded, then:*

$$L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C)$$

Proof: Let set H consist of packets that have start tag equal to virtual arrival time. Formally,

$$H = \{m | m > 0 \wedge S(p_f^m) = v(A(p_f^m))\}$$

Since $S(p_f^1) = v(A(p_f^1))$, $\mid H \mid \geq 1$. Let $k \leq j$ be largest integer in H. Also, let $v_1 = v(A(p_f^k))$ and $v_2 = S(p_f^j)$. From the definition of SFQ, we know that the set of packets that are served in the interval $[A(p_f^k), L_{SFQ}(p_f^j)]$ have start tag at least v_1 and at most v_2. Hence,

$$L_{SFQ}(p_f^j) \leq A(p_f^k) + T(\widehat{W}(v_1, v_2)) \tag{5.30}$$

where $T(w)$ denotes the time taken to serve packets of aggregate length w.

Now observe from the definition of k that as the server virtual time is set to the maximum finish tag assigned to any packet at the end of a busy period, packet p_f^k and p_f^j are served in the same busy period of a server. Furthermore, from the definition of k, $v_2 - v_1 = \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}}$. Hence, using Lemma 5.3 we get:

$$\widehat{W}(v_1, v_2) \leq C \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j \tag{5.31}$$

From the definition of Fluctuation Constrained server, we know:

$$T(w) \leq \frac{w}{C} + \delta(C) \tag{5.32}$$

Substituting (5.31) and (5.32) in (5.30) we get:

$$L_{SFQ}(p_f^j) \leq A(p_f^k) + \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C)$$

From (5.28) we know, $A(p_f^k) + \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} \leq EAT(p_f^j, r_f^j)$. Hence,

$$L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C)$$

■

Theorem 5.5 *If the capacity of a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$ is not exceeded, then:*

$$P\left(L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma} \quad (5.33)$$

Proof: Let $T(w)$ be the random variable denoting the time taken by server to serve packets of aggregate length w in a busy period. From the definition of Exponentially Bounded Fluctuation server, we get:

$$P\left(T(C(t_2 - t_1)) \leq (t_2 - t_1) + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma} \quad (5.34)$$

Define k, v_1 and v_2 as in the proof of Theorem 5.4. Then:

$$L_{SFQ}(p_f^j) \leq A(p_f^k) + T(\widehat{W}(v_1, v_2))$$

$$\Rightarrow P\left(L_{SFQ}(p_f^j) \leq A(p_f^k) + \frac{\widehat{W}(v_1, v_2)}{C} + \delta(C) + \gamma\right) \geq$$
$$P\left(T(\widehat{W}(v_1, v_2)) \leq \frac{\widehat{W}(v_1, v_2)}{C} + \delta(C) + \gamma\right)$$

From (5.34) we get:

$$P\left(L_{SFQ}(p_f^j) \leq A(p_f^k) + \frac{\widehat{W}(v_1, v_2)}{C} + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma}$$

Substituting the upper bound on $\widehat{W}(v_1, v_2)$ from (5.31), we get:

$$P\left(L_{SFQ}(p_f^j) \leq A(p_f^k) + \sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma}$$

From (5.28) we know, $A(p_f^k)+\sum_{n=0}^{n=j-k-1} \frac{l_f^{k+n}}{r_f^{k+n}} \leq EAT(p_f^j, r_f^j)$. Substituting for $EAT(p_f^j, r_f^j)$ yields (5.33). ■

Theorem 5.4 demonstrates that SFQ guarantees a deadline to a packet based on its expected arrival time. Hence, it belongs to the GR class and has all the desirable properties of the algorithms in GR.

The delay guarantee derived in Theorems 5.4 and 5.5 is independent of a tie breaking rule that a SFQ server may use when more than one packet have the same start tag. Though a tie breaking rule does not effect the delay guarantee, it can be used by a server to achieve different objectives. For example, a tie-breaking rule may give higher priority to interactive, low-throughput applications to reduce the average delay.

Theorems 5.4 and 5.5 can be used to determine delay guarantee even when a server has flows with different priorities and services them in the priority order (such a scenario may occur in an integrated services network with different traffic types). To illustrate, consider a server that services flows with two priorities and uses SFQ to schedule the packets of lower priority flows. If the bandwidth requirement of higher priority flows can be characterized by a leaky bucket with average rate ρ and burstiness σ (such a characterization may be enforced by shaping the higher priority flows through a leaky bucket), then the residual bandwidth available to the lower priority flows can be modeled as being fluctuation constrained with parameters $(C - \rho, \frac{\sigma}{C})$. Hence, Theorem 5.4 can be used to determine the delay guarantee of the lower priority flows. Similarly, if the aggregate arrival process of the high priority flows can be modeled as poisson process, then the residual bandwidth can be modeled as EBF server [37] and Theorem 5.5 can be used to determine the delay guarantee.

Theorem 5.4 demonstrates that maximum delay of a packet in SFQ is smaller than in SCFQ. Specifically, a tight bound on the departure time of a packet at a constant rate server employing SCFQ, derived in Chapter 2,is:

$$L_{SCFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{r_f^j} \tag{5.35}$$

Since $\delta(C) = 0$ for a constant rate server, the difference in maximum delay that a packet may incur at servers employing SCFQ and SFQ is:

$$\frac{l_f^j}{r_f^j} - \frac{l_f^j}{C}$$

Clearly, maximum delay in SFQ is smaller than in SCFQ. To illustrate numerically, when r_f^j =64Kb/s, $l_f^j = 200$ bytes and C=100Mb/s, the difference is 24.4ms. If there are K servers on the path of a flow, this difference increases by a factor of K. Similarly, the difference increases linearly with the packet size.

Theorem 5.4 also shows that, unlike WFQ, the maximum delay of a packet in SFQ depends on the maximum packet length of all the flows at the server. However, in spite of this dependence, SFQ provides lower maximum delay, as compared to WFQ, to low-throughput flows. To observe this, consider the difference in the maximum delay experienced by packet p_f^j, denoted by $\Delta(p_f^j)$, in WFQ and SFQ. Since WFQ guarantees that packet p_f^j will be transmitted by $EAT(p_f^j, r_f^j) + \frac{l_f^j}{r_f^j} + \frac{l_{max}}{C}$ where l_{max} is the maximum packet length at the server (see Chapter 2), we get:

$$\Delta(p_f^j) = \frac{l_f^j}{r_f^j} + \frac{l_{max}}{C} - \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} - \frac{l_f^j}{C}$$

Hence, $\Delta(p_f^j) \geq 0$ if:

$$r_f^j \leq \frac{C \quad l_f^j}{\sum_{n \in Q \wedge n \neq f} l_n^{max} + l_f^j - l_{max}} \tag{5.36}$$

To gain a qualitative understanding of (5.36), let $l_f^j = l_{max} = l_n^{max} = l$ and $r_f^j = r_f$. Then, $\Delta(p_f^j) \geq 0$ if $r_f \leq \frac{C}{|Q|-1}$. That is, maximum delay of packets of a flow in SFQ is smaller than in WFQ if the link bandwidth used by the flow is at most $\frac{C}{|Q|-1}$; such a flow is referred to as a low-throughput flow. This is also illustrated by Figure 5.3(a), which plots the reduction in delay in SFQ for different number of flows and flow rates, assuming 200 bytes packet and link capacity of 100Mb/s. As the figure shows, whereas the delay reduces for flows with rate $r_f < \frac{C}{|Q|-1}$, i.e., low throughput flows, it increases for flows with rate $r_f \geq \frac{C}{|Q|-1}$, i.e., high throughput flows. To compare the delay performance of WFQ and SFQ in an example scenario, consider a network link that is servicing 70 flows (possibly video flows) with rate 1Mb/s and

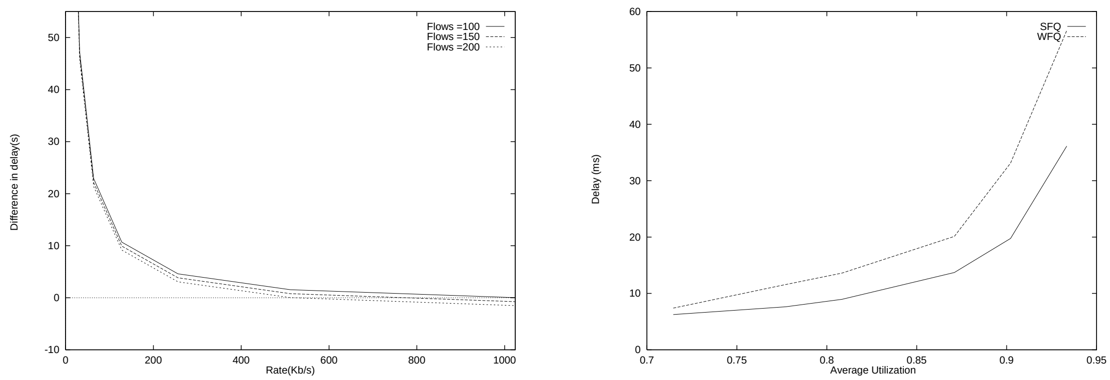


Figure 5.3: (a) Difference in maximum delay in WFQ and SFQ (b) Comparison of average delay in WFQ and SFQ

200 flows (possibly audio flows) with rate 64Kb/s. In such a scenario, whereas the maximum delay of the packets of flow with rate 64 Kb/s reduces by 20.39ms in SFQ, the maximum delay of 1Mb/s flows increases by 2.48 ms.

SFQ is also expected to lower the average delay of low-throughput applications while increasing the average delay of high-throughput ones. This is because whereas SFQ schedules packets in the increasing order of start tags, and thereby schedules packets at the earliest possible instant; WFQ schedules packets in increasing order of finish tag, and thus delays a packet as long as possible. To validate this hypothesis, we simulated a switch that was shared by high and low throughput flows carrying poisson traffic. The link capacity was 1Mb/s and the packet size was 200 bytes. Seven high-throughput flows with average rate 100Kb/s shared the switch with varying number of low-throughput flows with average rate 32Kb/s. The number of low-throughput flows was varied from 2 to 10 and the switch was simulated for 1000 seconds. Figure 5.3(b) compares the average packet delay of low-throughput flows in WFQ and SFQ at varying levels of link utilization. As the figure illustrates, the average delay of low-throughput flows is higher in WFQ than in SFQ; at 80.81% link utilization, the average delay is $4.7ms$ higher in WFQ than in SFQ. Hence, SFQ provides lower average delay to low-throughput applications as compared to WFQ.

5.4.2 Delay-cum-throughput Guarantee

We first establish a general property of SFQ FC and EBF servers in Theorems 5.6 and 5.7, respectively, and then derive their delay-cum-throughput guarantees in Corollaries 5.1 and 5.2.

Theorem 5.6 *If the capacity of a SFQ FC server with parameters $(C, \delta(C))$ is not exceeded, then:*

$$L_{SFQ}(p_f^j) \leq t + \sum_{n=k-1}^{n=j-1} \frac{l_f^n}{r_f^n} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C) \tag{5.37}$$

where $t \geq A(p_f^j)$ and packet p_f^k is the first packet in the queue of flow f at time t.

Proof: Let $S(p_f^0) = F(p_f^0) = -\infty$. If $t \geq L_{SFQ}(p_f^j)$, the theorem holds trivially. Thus let $t < L_{SFQ}(p_f^j)$. There are two cases to consider:

- $S(p_f^{k-1}) \leq v(t) < F(p_f^{k-1})$: Define $v_1 = S(p_f^{k-1})$ and $v_2 = S(p_f^j)$. Since all packets that are served by the server in the interval $[t, L_{SFQ}(p_f^j)]$ have start tag at least v_1 and at most v_2, we get:

  $$L_{SFQ}(p_f^j) \leq t + T(\widehat{W}(v_1, v_2))$$

  where $T(w)$ and $\widehat{W}(v_1, v_2)$ are as defined in the proof of Theorem 5.4. From the definition of v_1 and v_2, we get $v_2 - v_1 = \sum_{n=k-1}^{n=j-1} \frac{l_f^n}{r_f^n}$.

- $v(t) \geq F(p_f^{k-1})$: Define $\hat{v_1} = S(p_f^k)$ and $\hat{v_2} = S(p_f^j)$. Since packet p_f^k is backlogged, $v(A(p_f^k)) \leq v(t)$. Furthermore, $\hat{v_1} = \max\{v(A(p_f^k)), F(p_f^{k-1})\}$. Hence, $\hat{v_1} \leq v(t)$. But since packet p_f^k is backlogged, $\hat{v_1} \geq v(t)$. Hence, $\hat{v_1} = v(t)$. Since $v(t) = \hat{v_1}$ all packets that are served by the server in the interval $[t, L_{SFQ}(p_f^j)]$ have start tag at least $\hat{v_1}$ and at most $\hat{v_2}$, we get:

  $$L_{SFQ}(p_f^j) \leq t + T(\widehat{W}(\hat{v_1}, \hat{v_2}))$$

  From the definition of $\hat{v_2} - \hat{v_1}$, we get $\hat{v_2} - \hat{v_1} = \sum_{n=k}^{n=j-1} \frac{l_f^n}{r_f^n}$.

Since $(\hat{v_2} - \hat{v_1}) \leq (v_2 - v_1)$, $T(\widehat{W}(\hat{v_1}, \hat{v_2})) \leq T(\widehat{W}(v_1, v_2))$. Hence, in either of the above cases:

$$L_{SFQ}(p_f^j) \leq t + T(\widehat{W}(v_1, v_2))$$

Using Lemma 5.3 for $\widehat{W}(v_1, v_2)$ and $T(w) \leq \frac{w}{c} + \delta(C)$ we get (5.37) . ∎

Theorem 5.7 *If the capacity of a SFQ EBF server with parameters* $(C, B, \alpha, \delta(C))$ *is not exceeded, then:*

$$P\left(L_{SFQ}(p_f^j) \leq t + \sum_{n=k-1}^{n=j-1} \frac{l_f^n}{r_f^n} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma}$$

where $t \geq A(p_f^j)$ *and packet* p_f^k *is the first packet in the queue of flow* f *at time* t*.*

Proof: The proof is similar to the proof of Theorem 5.6. The only difference is that the probabilistic interpretation of $T(w)$ for EBF servers, as in the proof of Theorem 5.5, is used. ■

Corollaries 5.1 and 5.2 use Theorems 5.6 and 5.7, respectively, to derive the delay-cum-throughput guarantees of SFQ FC and EBF servers, respectively.

Corollary 5.1 *If the capacity of a SFQ FC server with parameters* $(C, \delta(C))$ *is not exceeded, then:*

$$L_{SFQ}(p_f^j) \leq \max\left\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\right\} + \frac{l_f^{j-1}}{r_f^{j-1}} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C)$$

where $L_{SFQ}(p_f^0) = 0$.

Proof: Let $t = \max\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\}$. Then packet p_f^j is the first packet in the queue of flow f at time t. Hence, in Theorem 5.6, $p_f^k = p_f^j$ and the corollary follows. ■

Corollary 5.2 *If the capacity of a SFQ EBF server with parameters* $(C, B, \alpha, \delta(C))$ *is not exceeded, then*

$$P\left(L_{SFQ}(p_f^j) \leq \max\left\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\right\} + \frac{l_f^{j-1}}{r_f^{j-1}} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma}$$

where $L_{SFQ}(p_f^0) = 0$.

Proof: Let $t = \max\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\}$. Then packet p_f^j is the first packet in the queue of flow f at time t. Hence, in Theorem 5.7, $p_f^k = p_f^j$ and the corollary follows. ■

The property of SFQ established in Theorem 5.6 is similar to the worst-case fairness property of an algorithm that was proposed in [4]. An algorithm is *worst-case* fair if it guarantees that departure time of packet p_f^j, denoted by $L(p_f^j)$, is given as:

$$L(p_f^j) \leq t + \frac{Q_f(t)}{r_f} + \beta \tag{5.38}$$

where $t = A(p_f^j)$, β is a constant, $Q_f(t)$ is the queue length of the flow at time t, and $r_f = r_f^j$ is the rate assigned to the packets of the flow. The main difference in the worst-case fairness and the property established in Theorem 5.6 is that whereas (5.38) holds only for $t = A(p_f^j)$, (5.37) holds for any $t \geq A(p_f^j)$. This difference is crucial as Corollary 5.1 can be derived only because t can be greater than $A(p_f^j)$ in Theorem 5.6.

5.5 End-to-End Deadline Guarantee

In this section, we utilize the single server deadline guarantee to derive delay and delay-cum-throughput guarantee of a network of servers.

First consider a network of FC SFQ servers. Since a FC SFQ server provides deterministic delay and delay-cum-throughput guarantees, the end-to-end delay and delay-cum-throughput guarantees are directly known from the analysis presented in Chapters 2 and 3. Specifically, end-to-end delay guarantee is known from Theorem 2.9 in Chapter 2 and delay-cum-throughput guarantee from Theorem 3.1 in Chapter 3. However, since these theorems require that each server on the path provide a deterministic deadline guarantees, they can not be employed to determine the deadline guarantees for a network where some of the servers may be EBF SFQ servers. Hence, in the following sections, we derive the deadline guarantees when some of the servers may be EBF SFQ servers.

5.5.1 End-to-End Delay Guarantee

The objective is to determine the deadline guarantee of a network of servers based on the expected arrival time of a packet at the first server on the path of a flow. To do so, let the i^{th} server along the path of a flow be denoted as server i. Also, let there be K servers on the path of a flow and let each of the server guarantee a deadline to a packet based on its expected arrival time. Then, the network guarantees a deadline to a packet based on its expected arrival time at the K^{th} server. Observe that the expected arrival time of a packet at server K is dependent on departure time of packet at server $K-1$, which, in turn, is dependent on expected arrival time of the packet at server $K-1$. Using this argument recursively, a network of servers can guarantee a deadline to a packet based on the expected arrival time of the packet at the first server. We employed this method to derive end-to-end delay guarantee of a network of servers in Chapter 2 when each of of the server provides a deterministic bound on the departure time of a packet. We now generalize it to a network of servers where some servers may be EBF servers and consequently provide only probabilistic bounds on the departure time.

Observe that SFQ delay guarantee for both FC and EBF servers when the server capacity is not exceeded can be rewritten as:

$$P\left(L_{SFQ}(p_f^j) \leq EAT(p_f^j, r_f^j) + \beta_f^j + \gamma\right) \geq 1 - Be^{-\lambda\gamma} \tag{5.39}$$

Substituting $\beta_f^j = \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C)$, $B = 0$ and $\lambda = \infty$, yields the delay guarantee for FC server. Substituting $\beta_f^j = \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^j}{C} + \delta(C)$ and $\lambda = \alpha$, yields the delay guarantee for EBF servers. Hence, we will use (5.39) to derive the end-to-end delay guarantee. Furthermore, to facilitate interoperability with other scheduling algorithms, we will only require each server on the path of a flow to guarantee a deadline which is similar to (5.39). We first relate the expected arrival time of a packet at adjacent servers in Theorem 5.8 and then use it to derive end-to-end delay guarantee in Corollary 5.3.

Let τ^i be an upper bound on the propagation delay between servers i and $i+1$. Also, let all the variables of server i be identified by superscript i, i.e., β_f^j and r_f^j are identified as $\beta_f^{j,i}$ and $r_f^{j,i}$, respectively. Henceforth in this section, we will refer to a single flow f, and hence, drop the subscript f from all the variables.

Theorem 5.8 *If scheduling algorithm at server i guarantees that:*

$$P\left(L^i(p^j) \leq EAT^i(p^j, r^{j,i}) + \beta^{j,i} + \gamma\right) \geq 1 - B^i e^{-\lambda^i\gamma} \tag{5.40}$$

where $L^i(p^j)$ is the time at which packet p^j departs server i, then:

$$P\left(EAT^{i+1}(p^j,\hat{r}^{j,i}) \leq EAT^i(p^j,\hat{r}^{j,i}) + \max_{n\in[1..j]}\{\beta^{n,i}\} + \tau^i + \gamma\right) \geq 1 - B^i e^{-\lambda^i\gamma} \quad (5.41)$$

where $\hat{r}^{j,i} \leq \min\{r^{j,i}, r^{j,i+1}\}$.

Proof: The proof is by induction on j.

Base Case: $j = 1$

$$EAT^{i+1}(p^1,\hat{r}^{1,i}) = A^{i+1}(p^1)$$

From (5.40) we know:

$$\begin{aligned}
P\left(L^i(p^1) + \tau^i \leq EAT^i(p^1,\hat{r}^{1,i}) + \beta^{1,i} + \tau^i + \gamma\right) &\geq 1 - B^i e^{-\lambda^i\gamma} \\
P\left(A^{i+1}(p^1) \leq EAT^i(p^1,\hat{r}^{1,i}) + \beta^{1,i} + \tau^i + \gamma\right) &\geq \\
P\left(EAT^{i+1}(p^1,\hat{r}^{1,i}) \leq EAT^i(p^1,\hat{r}^{1,i}) + \max_{n\in[1..1]}\{\beta^{n,i}\} + \tau^i + \gamma\right) &\geq
\end{aligned}$$

Hence (5.41) holds for $j = 1$.

Induction Hypothesis: Assume (5.41) holds for $1 \leq j \leq m$.

Induction: We need to show (5.41) holds for $1 \leq j \leq m+1$.

$$EAT^{i+1}(p^{m+1},\hat{r}^{m+1,i}) = \max\{A^{i+1}(p^{m+1}), EAT^{i+1}(p^m,\hat{r}^{m,i}) + \frac{l^m}{\hat{r}^{m,i}}\} \quad (5.42)$$

There are two cases to consider:

- $A^{i+1}(p^{m+1}) > EAT^{i+1}(p^m,\hat{r}^{m,i}) + \frac{l^m}{\hat{r}^{m,i}}$: From (5.42) we get:

  $$EAT^{i+1}(p^{m+1},\hat{r}^{m+1,i}) = A^{i+1}(p^{m+1})$$

  From (5.40) we know:

  $$\begin{aligned}
  P\left(L^i(p^{m+1}) + \tau^i \leq EAT^i(p^{m+1},\hat{r}^{m+1,i}) + \beta^{m+1,i} + \tau^i + \gamma\right) &\geq 1 - B^i e^{-\lambda^i\gamma} \\
  P\left(A^{i+1}(p^{m+1}) \leq EAT^i(p^{m+1},\hat{r}^{m+1,i}) + \beta^{m+1,i} + \tau^i + \gamma\right) &\geq \quad (5.43)
  \end{aligned}$$

  Since $EAT^{i+1}(p^{m+1},\hat{r}^{m+1,i}) = A^{i+1}(p^{m+1})$, we get

  $$\begin{aligned}
  P\left(EAT^{i+1}(p^{m+1},\hat{r}^{m+1,i}) \leq EAT^i(p^{m+1},\hat{r}^{m+1,i}) + \max_{n\in[1..m+1]}\{\beta^{n,i}\} + \tau^i + \gamma\right) \\
  \geq 1 - B^i e^{-\lambda^i\gamma} \quad (5.44)
  \end{aligned}$$

- $A^{i+1}(p^{m+1}) \leq EAT^{i+1}(p^m, \widehat{r}^{m,i}) + \frac{l^m}{\widehat{r}^{m,i}}$: From (5.42) we get:

$$EAT^{i+1}(p^{m+1}, \widehat{r}^{m+1,i}) = EAT^{i+1}(p^m, \widehat{r}^{m,i}) + \frac{l^m}{\widehat{r}^{m,i}} \tag{5.45}$$

From the induction hypothesis, we know:

$$P\left(EAT^{i+1}(p^m, \widehat{r}^{m,i}) \leq EAT^i(p^m, \widehat{r}^{m,i}) + \max_{n\in[1..m]}\{\beta^{n,i}\} + \tau^i + \gamma\right) \geq 1 - B^i e^{-\lambda^i \gamma}$$

$$P\left(EAT^{i+1}(p^m, \widehat{r}^{m,i}) + \frac{l^m}{\widehat{r}^{m,i}} \leq EAT^i(p^m, \widehat{r}^{m,i}) + \frac{l^m}{\widehat{r}^{m,i}} + \max_{n\in[1..m]}\{\beta^{n,i}\} + \tau^i + \gamma\right) \geq 1 - B^i e^{-\lambda^i \gamma}$$

Since $EAT^i(p^m, \widehat{r}^{m,i}) + \frac{l^m}{\widehat{r}^{m,i}} \leq EAT^i(p^{m+1}, \widehat{r}^{m+1,i})$, using (5.45) we get:

$$P\left(EAT^{i+1}(p^{m+1}, \widehat{r}^{m+1,i}) \leq EAT^i(p^{m+1}, \widehat{r}^{m+1,i}) + \max_{n\in[1..m+1]}\{\beta^{n,i}\} + \tau^i + \gamma\right) \geq 1 - B^i e^{-\lambda^i \gamma} \tag{5.46}$$

From (5.44), (5.46) and the induction hypothesis, we conclude that (5.41) holds for $1 \leq j \leq m+1$. Hence, the theorem follows. ■

Corollary 5.3 If scheduling algorithm at each server on the path of a flow satisfies (5.40), and there are K servers on the path of the flow, then:

$$P\left(L^K(p^j) \leq EAT^1(p^j, \widehat{r}^j) + \sum_{n=1}^{n=K} \max_{m\in[1..j]}\{\beta^{m,n}\} + \sum_{n=1}^{n=K-1} \tau^n + \gamma\right) \geq 1 - \left(\sum_{n=1}^{n=K} B^n\right) e^{-\gamma \frac{1}{\sum_{n=1}^{n=K} \frac{1}{\lambda^n}}}$$

where $L^K(p^j)$ is the time at which packet p^j leaves server K, and $\widehat{r}^j = \min_{n\in[1..K]} \widehat{r}^{j,n}$.

Proof: Define:

$$D^{n+1}(p^j) = EAT^{n+1}(p^j, \widehat{r}^j) - EAT^n(p^j, \widehat{r}^j) - \max_{m\in[1..j]} \beta^{m,n} - \tau^n \quad 1 \leq n \leq K-1$$

Also let:

$$D^{K+1}(p^j) = L^K(p^j) - EAT^K(p^j, \hat{r}^j) - \max_{m \in [1..j]} \beta^{m,K}$$

Since $\hat{r}^j \leq \hat{r}^{j,n-1}$, from Theorem 5.8 we get:

$$P(D^n(p^j) > \gamma) \leq B^{n-1} e^{-\gamma \lambda^{n-1}} \qquad 2 \leq n \leq K$$

Hence D^n is an Exponentially bounded random variable with parameters B^{n-1} and λ^{n-1} [37]. Similarly from Theorem 5.5, we know:

$$P\left(D^{K+1}(p^j) > \gamma\right) \leq B^K e^{-\gamma \lambda^K}$$

Hence, $D^{K+1}(p^j)$ is an Exponentially Bounded variable with parameters B^K and λ^K. From [56] we know that sum of Exponentially Bounded variables $D_1, ..., D_K$ with parameters $B_1, ..., B_K$ and $\lambda_1, ..., \lambda_K$ is also an Exponentially Bounded variable with parameters $\sum_{n=1}^{n=K} B_n$ and $\frac{1}{\sum n=K_{n=1} \frac{1}{\lambda^n}}$. Hence, $\sum_{n=2}^{n=K+1} D^n(p^j)$ is an Exponentially Bounded variable. Therefore,

$$\begin{aligned} P\left(\sum_{n=2}^{n=K+1} D^n(p^j) > \gamma\right) &\leq (\sum_{n=1}^{n=K} B^n) e^{-\gamma \frac{1}{\sum_{n=1}^{n=K} \frac{1}{\lambda^n}}} \\ P\left(\sum_{n=2}^{n=K+1} D^n(p^j) \leq \gamma\right) &\leq 1 - (\sum_{n=1}^{n=K} B^n) e^{-\gamma \frac{1}{\sum_{n=1}^{n=K} \frac{1}{\lambda^n}}} \end{aligned} \qquad (5.47)$$

Now,

$$\sum_{n=2}^{n=K+1} D^n(p^j) = L^K(p^j) - EAT^1(p^j, \hat{r}^j) - \sum_{n=1}^{n=K} \max_{m \in [1..j]} \{\beta^{m,n}\} - \sum_{n=1}^{n=K-1} \tau^n \qquad (5.48)$$

Hence, the corollary follows from substituting (5.48) in (5.47). ■

To derive Corollary 5.3, we have only required the scheduling algorithm at each server to satisfy (5.40). Hence, any scheduling algorithm that satisfies (5.40) can interoperate to provide end-to-end guarantee. Furthermore, Corollary 5.3 can be used for an internetwork of FC and EBF servers. Finally, the proof method of Theorem 5.8 and Corollary 5.3 can be generalized as in Chapter 2 to derive end-to-end delay guarantee even when packet may be fragmented and reassembled in the network. Hence, SFQ can provide guarantees in heterogeneous internetworking environments.

5.5.2 End-to-End Delay-cum-throughput Guarantee

When a flow is served by a network of servers, a destination knows the departure time of a packet from the last server. Furthermore, from the traffic characteristics of a flow, it may also know the arrival time of a packet at the first server on the path. Hence, the objective is to determine a bound on the departure time of a packet from the last server based on its arrival time at the first server and departure time of the previous packet at the last server. We had achieved this objective in Theorem 3.1 presented in Chapter 3 when all the servers on the path of a flow provided a deterministic bound on the departure time of a packet. We now generalize the proof methodology to derive the end-to-end delay-cum-throughput guarantee for a network of servers in which some servers may be EBF servers and provide probabilistic bound on departure time of a packet.

Observe that SFQ delay-cum-throughput guarantee for both FC and EBF servers when the server capacity is not exceeded can be rewritten as:

$$P\left(L_{SFQ}(p_f^{j+1}) \leq \max\{L_{SFQ}(p_f^j, A(p_f^{j+1})\} + \beta_f^{j+1} + \gamma\right) \geq 1 - Be^{-\lambda\gamma} \tag{5.49}$$

Substituting $\beta_f^{j+1} = \frac{l_f^j}{r_f^j} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^{j+1}}{C} + \delta(C)$, $B = 0$ and $\lambda = \infty$, yields the delay-cum-throughput guarantee for FC server. Substituting $\beta_f^{j+1} = \frac{l_f^j}{r_f^j} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \frac{l_f^{j+1}}{C} + \delta(C)$ and $\lambda = \alpha$ yields the delay-cum-throughput guarantee for EBF servers. Hence, we will use (5.49) to derive the end-to-end delay-cum-throughput guarantee. Furthermore, to facilitate interoperability with other scheduling algorithms, we will only require each server on the path of a flow to guarantee a deadline which is similar to (5.39).

Let $\hat{\tau}^i$ denote the lower bound on the propagation delay between servers i and $i+1$. As in the previous section, we drop the subscript f from all the variables and identify all variables of i^{th} server by superscript i. Theorem 5.9 establishes the end-to-end delay-cum-throughput guarantee.

Theorem 5.9 *If there are K servers on the path of a flow, and each server i guarantees that:*

$$P\left(L^i(p^{j+1}) \leq \max\{L^i(p^j), A^i(p^{j+1})\} + \beta^{j+1,i} + \gamma\right) \geq 1 - B^i e^{-\lambda^i\gamma} \tag{5.50}$$

where $L^i(p^j)$ is the time at which packet p^j departs server i, then:

$$P\left(L^i(p^{j+1}) \leq \max\{L^K(p^j) - \sum_{n=1}^{n=K-1} \hat{\tau}^n, A^1(p^{j+1})\} + \sum_{n=1}^{n=i-1} \tau^n + \sum_{n=1}^{n=i} \beta^{j+1,n} + \gamma\right) \geq 1 - (\sum_{n=1}^{n=i} B^n) e^{-\gamma \frac{1}{\sum_{n=1}^{n=i} \frac{1}{\lambda^n}}} \; ; \; 1 \leq i \leq K \tag{5.51}$$

Proof: The proof is by induction on i.

Base Case ($i = 1$)**:** Since $\sum_{n=1}^{n=K-1} \hat{\tau}^n$ is the minimum propagation delay between the first and the K^{th} server, we get:

$$L^1(p^j) \leq L^K(p^j) - \sum_{n=1}^{n=K-1} \hat{\tau}^n \tag{5.52}$$

Substituting (5.52) in (5.50) with $i = 1$ yields (5.51) for $i = 1$.

Induction Hypothesis: Assume (5.51) holds for $1 \leq i \leq m$.

Induction : We need to show (5.51) holds for $1 \leq i \leq m + 1$. From (5.50), i.e., the definition of throughput guarantee, we know:

$$L^{m+1}(p^{j+1}) \leq \max\{L^{m+1}(p^j), A^{m+1}(p^{j+1})\} + \beta^{j+1,m+1} + \Gamma^{m+1}$$

where Γ^{m+1} is an exponentially bounded random variable with parameters B^{m+1} and λ^{m+1},i.e., $P(\Gamma^{m+1} > \gamma) \leq B^{m+1} e^{-\gamma\lambda^{m+1}}$. Since $A^{m+1}(p^{j+1}) \leq L^m(p^{j+1}) + \tau^m$, we get:

$$L^{m+1}(p^{j+1}) \leq \max\{L^{m+1}(p^j), L^m(p^{j+1}) + \tau^m\} + \beta^{j+1,m+1} + \Gamma^{m+1} \tag{5.53}$$

From the induction hypothesis, we know:

$$L^m(p^{j+1}) \leq \max\{L^K(p^j) - \sum_{n=1}^{n=K-1} \hat{\tau}^n, A^1(p^{j+1})\} + \sum_{n=1}^{n=m-1} \tau^n + \sum_{n=1}^{n=m} \beta^{j+1,n} + \Gamma^m \tag{5.54}$$

where Γ^m is an exponentially bounded random variable with parameters $\sum_{n=1}^{n=m} B^n$ and $\frac{1}{\sum_{n=1}^{n=m} \frac{1}{\lambda^n}}$. Substituting (5.54) in (5.53), we get:

$$\begin{aligned} L^{m+1}(p^{j+1}) \quad \leq \quad & \max\left\{L^{m+1}(p^j), \max\left\{L^K(p^j) - \sum_{n=1}^{n=K-1} \hat{\tau}^n, A^1(p^{j+1})\right\} + \right. \\ & \left. \sum_{n=1}^{n=m} \tau^n + \sum_{n=1}^{n=m} \beta^{j+1,n} + \Gamma^m \right\} + \beta^{j+1,m+1} + \Gamma^{m+1} \end{aligned}$$

By: (1) moving $\sum_{n=1}^{n=m}\beta^{j+1,n}$ and Γ^m out from the second term of first maximum operator, (2) moving the term $\sum_{n=1}^{n=m-1}\tau^n$ inside the second maximum operator, and (3) then removing the second maximum operator we get:

$$\begin{aligned} L^{m+1}(p^{j+1}) \quad &\leq \quad \max\left\{L^{m+1}(p^j), L^K(p^j) - \sum_{n=1}^{n=K-1}\hat{\tau}^n + \sum_{n=1}^{n=m}\tau^n, A^1(p^{j+1}) + \sum_{n=1}^{n=m}\tau^n\right\} + \\ &\sum_{n=1}^{n=m+1}\beta^{j+1,n} + \Gamma^m + \Gamma^{m+1} \end{aligned} \tag{5.55}$$

Now,

$$L^{m+1}(p^j) \leq L^K(p^j) - \sum_{n=m+1}^{n=K-1}\hat{\tau}^n \tag{5.56}$$

Substituting (5.56) in (5.55) we get:

$$L^{m+1}(p^{j+1}) \leq \max\{L^K(p^j) - \sum_{n=m+1}^{n=K-1}\hat{\tau}^n, L^K(p^j) - \sum_{n=1}^{n=K-1}\hat{\tau}^n + \sum_{n=1}^{n=m}\tau^n, ...\} + ... \tag{5.57}$$

Since $\sum_{n=1}^{n=m}\tau^n - \sum_{n=1}^{n=m}\hat{\tau}^n \geq 0$,

$$L^K(p^j) - \sum_{n=m+1}^{n=K-1}\hat{\tau}^n \leq L^K(p^j) - \sum_{n=m+1}^{n=K-1}\hat{\tau}^n + (\sum_{n=1}^{n=m}\tau^n - \sum_{n=1}^{n=m}\hat{\tau}^n)$$

Hence, the first term in the maximum operator in (5.57) can be removed. Thus,

$$\begin{aligned} L^{m+1}(p^{j+1}) \quad &\leq \quad \max\left\{L^K(p^j) - \sum_{n=1}^{n=K-1}\hat{\tau}^n + \sum_{n=1}^{n=m}\tau^n, A^1(p^{j+1}) + \sum_{n=1}^{n=m}\tau^n\right\} \\ &\quad + \sum_{n=1}^{n=m+1}\beta^{j+1,n} + \Gamma^m + \Gamma^{m+1} \\ &\leq \quad \max\left\{L^K(p^j) - \sum_{n=1}^{n=K-1}\hat{\tau}^n, A^1(p^{j+1})\right\} + \sum_{n=1}^{n=m}\tau^n + \\ &\quad \sum_{n=1}^{n=m+1}\beta^{j+1,n} + \Gamma^m + \Gamma^{m+1} \end{aligned} \tag{5.58}$$

Let $\Gamma = \Gamma^m + \Gamma^{m+1}$. From [56] we know that sum of Exponentially Bounded variables D_1 and D_2 with parameters B_1, B_2 and λ_1, λ_2 is also an Exponentially Bounded variable, regardless of any statistical dependence between D_1 and D_2, with parameters $B_1 + B_2$ and $\frac{1}{\frac{1}{\lambda^1}+\frac{1}{\lambda^2}}$. Since Γ^m and Γ^{m+1} are exponentially bounded random variables, we conclude that Γ is also an exponentially bounded random variable with parameters $(\sum_{n=1}^{n=m+1}B^n)$ and $\frac{1}{\sum_{n=1}^{n=m+1}\frac{1}{\lambda^n}}$.

Hence, using the definition of exponentially bounded random variable in (5.58), we conclude that (5.51) holds for $1 \leq i \leq m+1$. ■

5.6 Discussion

SFQ borrows the concept of "self-clocking" and scheduling packets in the increasing order of start tags from SCFQ and FQS, respectively. However, it leads to better performance than either of the two. SFQ has the same fairness measure and implementation complexity as SCFQ but has smaller delay guarantee. Similarly, whereas FQS is unfair over variable rate servers and has high implementation complexity, SFQ is fair over variable rate servers and has lower implementation complexity. Furthermore, since in FQS, all Q flows can become active simultaneously, and consequently Q packets can have the same start tag, the bound on the departure time of a packet in FQS is at least that in SFQ.

The delay guarantee of SFQ depends on the maximum packet length of all the flows at the server. In contrast, the delay guarantee of WFQ depends only on the flow's properties. Thus, WFQ provides better isolation of delay guarantee of a flow. However, as we will show in Chapter 8, the delay guarantee of SFQ is similar to that of an online algorithm [4] that minimizes unfairness. Furthermore, as we demonstrated in Section 5.4.1 it is the lack of isolation of delay guarantee that enables SFQ to provide lower delay to low throughput flows. However, if SFQ is employed to provide a priori specified bounds on packet delay, then the maximum number of flows as well as their packet sizes would have to be estimated. In some networking environments, such an estimate may be large and consequently SFQ may not be able to provide lower a priori delay to low throughput applications. For networking environments where the primary objective of a scheduling algorithm is to provide a priori bounds, we have designed a class of Fair Airport (FA) scheduling algorithms. We present the design and analysis of FA in the next chapter.

[4] An online scheduler is one which does not use the length of packet p_f^{k+1} in making a scheduling decision for packet p_f^k.

5.7 Conclusions

In this chapter, we presented Start-time Fair Queuing (SFQ) algorithm that is computationally efficient, achieves fairness regardless of variation in a server capacity, and has fairness guarantee that is at least as good as all the known scheduling algorithms. We analyzed its throughput, single server deadline, and end-to-end deadline guarantee for variable rate Fluctuation Constrained (FC) and Exponentially Bounded Fluctuation (EBF) servers. To the best of our knowledge, this is the first analysis of any fair or real-time scheduling algorithm for such servers.

We demonstrated that SFQ: (1) achieves low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provides fairness which is desirable for VBR video; (3) provides fairness, regardless of variation in server capacity, for throughput-intensive, flow-controlled data applications; (4) provides QoS guarantees over variable rate servers; and (5) is computationally efficient. In Chapter 7, we show that it enables hierarchical link bandwidth allocation, and thus meets all the requirements of a scheduling algorithm for integrated services networks.

Chapter 6

Fair Airport Scheduling Algorithms

In the previous chapter, we presented Start-time Fair Queuing (SFQ) algorithm. As Theorem 5.4 demonstrated, the maximum delay of packets in SFQ depends on the maximum packet length of all the flows at a server. Consequently, if SFQ is employed to provide *a priori* bounds on packet delay of flows, the maximum packet length and the number of flows at a server would have to be estimated. This estimate may be large and hence the bounds on packet delay may be large. Hence, in networking environments where one of the main objectives of a scheduling algorithm is to provide pre-specified bounds on packet delay, it may be desirable to employ:

- Scheduling algorithms that allocate only rate and have delay guarantee similar to Weighted Fair Queuing (WFQ): Several fair algorithms that have delay guarantee similar to WFQ are known (for example, WFQ [12, 42], FFQ [52], WF^2Q [4], WF^2Q+ [3], Leap Forward Virtual Clock [53], etc.). As demonstrated in Chapter 4, most of these algorithms are either unfair over variable rate servers or require the number of bits transmitted to be counted to achieve fairness over variable rate servers. Since counting the number of bits transmitted may be expensive, algorithms that do not have this requirement for achieving fairness over variable rate servers are desirable.

- Scheduling algorithms that achieve separation of rate and delay allocation: The algorithms that have delay guarantee similar to WFQ provide low delay to low throughput flows by reserving higher rate; this may result into low utilization of network resources.

Hence, algorithms that achieve separation of rate and delay allocation and provide lower maximum delay to packets of a flow without increasing the rate reservation are desirable. Though unfair algorithms that achieve separation of rate and delay allocation are known (for example, Delay EDD), fair algorithms that achieve the same objective are not.

Thus, efficient algorithms that either allocate only rate or achieve separation of rate and delay allocation and are fair over variable rate servers are not known. We address this limitation in this chapter by designing a class of Fair Airport (FA) algorithms. An algorithm in FA class combines SFQ with a non-work conserving algorithm in Rate Controlled Service Discipline (RCSD) class [19, 60]. We derive fairness and deadline guarantees for FA servers and demonstrate that by appropriately choosing an algorithm from RCSD class, algorithms that either allocate only rate or achieve separation of rate and delay allocation and are fair over Fluctuation Constrained variable rate servers can be designed. This method leads to the design of the *first* fair algorithm that achieves separation of rate and delay allocation. We also show that the FA class contains algorithms with different implementation complexity and performance guarantees and thus enables tradeoffs between the two. Finally, the FA algorithms can be generalized to achieve other objectives such as prioritized fair allocation of residual bandwidth.

The rest of the chapter is structured as follows. We present the design of FA algorithms in Section 6.1. We then present the implementation complexity, fairness guarantee, and deadline properties in Sections 6.2, 6.3, and 6.4, respectively. Two instantiations of FA algorithms are presented in Section 6.5 and salient features of the FA algorithms are summarized in Section 6.6. Finally, Section 6.7 summarizes the results of this chapter.

6.1 Design of Fair Airport Scheduling Algorithms

To gain an intuitive understand the design of Fair Airport[1] scheduling algorithms, consider the class of Rate Controlled Service Disciplines (RCSD) [19, 60]. A RCSD scheduling algorithm employs a rate controller (also referred to as a shaper) for each flow (see Figure 6.1). A packet

[1]The class of Fair Airport scheduling algorithms derives its name from the class of Airport scheduling algorithms presented in [8].

on arrival joins the rate controller for its flow. The rate controller delays packets to ensure that the output traffic from the flow conforms to specified constraints. Once a packet passes through the rate controller, it joins a queue that we refer to as Guaranteed Service Queue (GSQ) [8]. Packets are serviced from GSQ using a work conserving GSQ scheduling algorithm.

Observe that the rate controller limits the service received by a flow from the GSQ in any time interval. Hence, intuitively, the difference in service received by any two flows in any time interval from the GSQ is bounded, i.e., the service received from GSQ is fair (we will demonstrate this formally in Section 6.3). Furthermore, since different scheduling algorithms can be employed for GSQ, different delay guarantees can be provided to flows. Thus, RCSD scheduling algorithms are fair and can provide different delay guarantees. They are, however, non-work conserving since GSQ may be empty and consequently the server may idle even when packets are backlogged at the rate controller. Hence, the challenge is to make the algorithm work conserving while ensuring that it remains fair.

RCSD can be made work conserving by introducing an Auxiliary Service Queue (ASQ) (this approach was first introduced in [8] and later employed in [19]). In such a case, packets on arrival not only join a rate controller, but also ASQ. When there are no packets in the GSQ, packets are scheduled from the ASQ. We assume that the packets of a flow are scheduled from ASQ in sequence. Since each packet joins both ASQ and GSQ, this ensures that packets of a flow will not be reordered. This makes RCSD work conserving while ensuring that packets of a flow are not reordered. We now need to ensure that it continues to remain fair. We can achieve this objective by employing a *fair scheduling algorithm* for the ASQ. In such a case, the difference in service received by two flows from the ASQ will be bounded. Since the difference in service received by flows from GSQ is bounded, we conclude that the difference in aggregate service received by two flows from the server will be bounded, i.e., the resulting algorithm will be fair. Furthermore, since packets are transmitted only earlier from the ASQ than they are guaranteed to depart from the GSQ, its delay guarantee for flows will be the same as the non-work conserving algorithm. Since non-work conserving RCSD can provide different delay guarantees to a flow, we conclude that we can design fair algorithms with different delay guarantees.

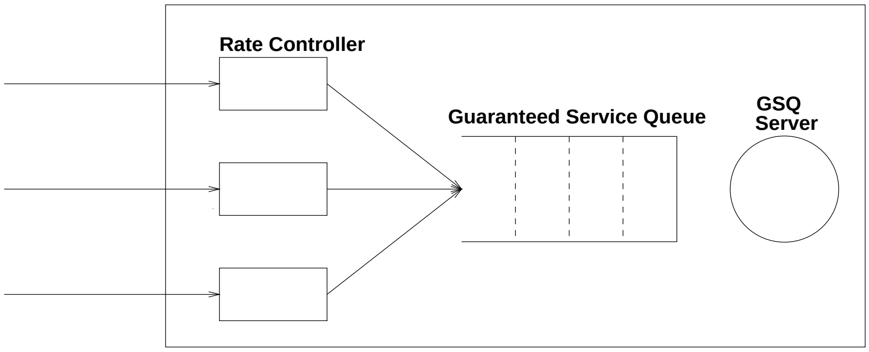


Figure 6.1: Non-work conserving RCSD

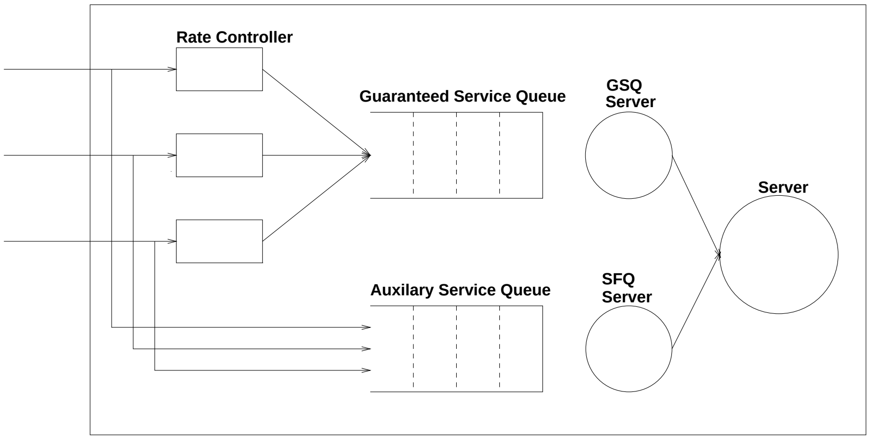


Figure 6.2: Fair Airport Scheduling Algorithms

Thus, Fair Airport (FA) scheduling algorithms, informally, are defined as fair work conserving RCSD algorithms (see Figure 6.2). We now present the detailed design of FA scheduling algorithms.

RCSD algorithms may employ rate controllers that enforce different traffic constraints. In FA, we restrict our focus to rate controllers that ensure that the output traffic for a flow is at most at its reserved rate. In such rate controllers, a packet departs at its *expected arrival time*. Let r_f be the rate reserved for flow f. Then, departure time departure time of packet p_f^j from the rate controller, denoted by $L_{RC}(p_f^j)$, is given as:

$$L_{RC}(p_f^j) = EAT(p_f^j, r_f)$$

where $EAT(p_f^j, r_f)$ is the expected arrival time of packet p_f^j and is defined as in Chapter 2, i.e.,:

$$EAT(p_f^j, r_f) = \max\left\{A(p_f^j), EAT(p_f^{j-1}, r_f) + \frac{l_f^{j-1}}{r_f}\right\} \tag{6.1}$$

where $EAT(p_f^0, r_f) + \frac{l_f^0}{r_f} = 0$. There are three main components of FA: rate controller, GSQ scheduling algorithm, and ASQ scheduling algorithm. Consider the design of each of the three components:

- *Rate controller*: In FA, a flow receives service through GSQ and ASQ. The service received by a flow from ASQ should not effect the service received from GSQ. Hence, if a packet is serviced from ASQ, it should not effect the departure time of the next packet from the rate controller. We achieve this objective by changing the departure time of a packet from the rate controller to its expected arrival time computed using only the subsequence of packets served through GSQ.

  Let $EAT^{RC}(p_f^j, r_f)$ denote the expected arrival time of packet p_f^j computed at the rate controller using only the subsequence of packets of flow f that were serviced through GSQ prior to p_f^j. Hence, if $p_f^{\theta(j)}$ is the packet prior to p_f^j that was served via GSQ, then using (6.1) we get:

  $$EAT^{RC}(p_f^j, r_f) = \max\{A(p_f^j), EAT^{RC}(p_f^{\theta(j)}, r_f) + \frac{l_f^{\theta(j)}}{r_f}\}$$

  We define the departure time of p_f^j from the rate controller as follows:

  $$EAT^{RC}(p_f^j, r_f) - \epsilon \leq L_{RC}(p_f^j) \leq EAT^{RC}(p_f^j, r_f) \tag{6.2}$$

  To enable efficient implementation of the rate controller, we have allowed the departure time from the rate controller to be at most ϵ before the expected arrival time.

- *GSQ scheduling algorithm*: We had informally argued that the delay guarantee of FA algorithm will be the same as that of the GSQ scheduling algorithm. Hence, to enable design of FA algorithms with different delay guarantees, we do not specify the GSQ scheduling algorithm. Any scheduling algorithm can be employed as long as it guarantees that the departure time of packet p_f^j from GSQ, denoted by $L_{GSQ}(p_f^j)$, is:

$$L_{GSQ}(p_f^j, r_f) \leq EAT^{GSQ}(p_f^j, r_f) + \beta_f^j \tag{6.3}$$

where $EAT^{GSQ}(p_f^j, r_f)$ is the expected arrival time of p_f^j computed at the GSQ.

- *ASQ scheduling algorithm*: The ASQ scheduling algorithm should have two characteristics:
  - *It must be fair over variable rate servers.* Since GSQ is given priority over ASQ, the bandwidth of the ASQ server varies over time. Hence, the ASQ scheduling algorithm must be fair over variable rate servers.
  - *It should not utilize the length of a packet in determining when the packet is scheduled.* In FA, a packet that is in the ASQ may be served from the GSQ. Such a packet should not effect the service received by a flow from the ASQ. This can be achieved by scheduling the next packet of the flow for the time the previous packet was scheduled. Since the packets of a flow may have variable length, this will be feasible if the ASQ scheduling algorithm does not utilize the length of a packet in determining when the packet is scheduled.

  SFQ is fair over variable rate servers. Furthermore, since it schedules packets in increasing order of start tags, it does not utilize the length of a packet in determining when it is scheduled. Hence, SFQ meets these two requirements and we employ it as the ASQ scheduling algorithm.

  The SFQ algorithm is as defined in the previous chapter. To ensure that packets that are serviced from GSQ do not effect the service received by a flow from the ASQ, the only change is that the start tags and finish tags of packets of a flow are computed using only the subsequence of packets served from the ASQ.

Hence, the class of FA scheduling algorithms is defined as follows:

1. On arrival, a packet of a flow joins the rate controller for the flow and ASQ. The departure time of packet p_f^j from the rate controller is given by (6.2).

2. Any algorithm that satisfies (6.3) can be used as the GSQ scheduling algorithm.

3. The ASQ scheduling algorithm is SFQ.

4. A packet is removed from the rate controller when it starts service in the ASQ.

5. A packet that joins GSQ is dequeued from ASQ after it has been serviced by the GSQ server. On removal of such a packet, the start tag of the next packet of that flow in the ASQ (if such a packet exists) is set to the start tag of the packet being removed. (This, in effect, computes the start and finish tags based on the subsequence of packets served from ASQ.)

6. The server gives priority to GSQ over ASQ. However, it does not preempt the transmission of a packet from ASQ.

To summarize, FA algorithms are work conserving realizations of RCSD algorithms that employ SFQ as the ASQ scheduler. To the best of our knowledge, SFQ is the only scheduling algorithm that meets the requirements of the ASQ scheduling algorithm that we had outlined earlier. Hence, SFQ is central to the design of FA algorithms.

In the following sections, we determine the implementation complexity and derive the fairness and deadline guarantees of the FA algorithms.

6.2 Implementation Complexity

Fair Airport algorithms can be implemented in two ways: rate controllers and GSQ and ASQ scheduling algorithms can be implemented separately or by a single data structure. First consider the implementation in which all three components are implemented separately. To determine the implementation complexity in such a case, consider the complexity of rate controller and the scheduling algorithms for GSQ and ASQ:

- Rate controller: If a packet is to be made eligible in GSQ by time t, then as per (6.2), it can be made eligible in the interval $[t - \epsilon, t]$. Since ϵ is non-zero, the rate controllers for all the flows can be implemented by a single calendar queue with the clock tick value of

ϵ [5]. The per packet complexity of calendar queue, and hence rate controller, is $O(1)$. Observe that if ϵ was 0, then implementing rate controller for all the flows would require a priority queue of eligible times of the first packets of the flows which would increase the rate controller complexity to $O(logQ)$.

- GSQ and ASQ scheduling algorithms: SFQ requires only the packets at the head of each flow queue to be sorted. Hence, it can be implemented by using a priority queue consisting only of the packets that are at the head of each flow queue. Consequently, the per packet computational complexity is $O(logQ)$ for the ASQ scheduling algorithm. The complexity of the GSQ scheduler, on the other hand, depends on the the choice of the algorithm. If the GSQ scheduler is a static priority scheduler, then its complexity would be $O(1)$ per packet and hence the aggregate complexity would be $O(logQ)$. If the GSQ scheduler is a dynamic priority scheduler that employs a priority queue consisting only of the packets that are at the head of each flow queue, then its complexity would be $O(logQ)$ and the aggregate complexity would continue to be $O(logQ)$. An important point to observe is that if GSQ scheduler employs a priority queue, then even though there are two priority queues, every packet is not inserted into the priority queue of both SFQ and GSQ servers. If n packets of a flow arrive in the flows busy period, a total of at most $n + 1$ insertions occur in either of the priority queues. This is because: (1) every packet that is inserted into the priority queue of the GSQ server is served by it, and (2) due to rule 5 of the algorithm, if k packets of a flow are inserted into SFQ priority queue during the flows busy period, then at least $k - 1$ packets are served by SFQ server. Consequently, at most $n + 1$ insertions occur.

Hence, if the GSQ scheduler per packet complexity is either $O(1)$ or $O(logQ)$, the per packet complexity of Fair Airport algorithms is $O(logQ)$.

A disadvantage of implementing rate controllers and GSQ and ASQ scheduling algorithms separately is that $O(Q)$ computation may have to be performed within a single packet transmission time. To observe this, note that it is possible that the expected arrival time (at the rate controller) of all the packets that are at the head of each flow queue may be same. Consequently, Q packets may join the GSQ during a single packet transmission time. Hence, in the worst-case, $O(Q)$ computation may have to be performed in a single packet transmission time. This worst-case complexity can be reduced to $O(logQ)$ by implementing the rate controllers and the GSQ and ASQ scheduling algorithms using a single augmented binary search tree [2, 7, 39]. We illustrate this implementation using an augmented red-black tree which is a balanced binary search tree [7].

To implement FA using a red-black tree, we assume that the GSQ scheduling algorithm assigns a priority value to each packet and schedules packets in increasing order of priority. Furthermore, the priority value of a packet in GSQ can be determined when the packet becomes the first packet in its flow queue. Finally, we assume that GSQ scheduler schedules packets of a flow in FIFO order. Since SFQ (ASQ scheduler) also schedules packets of a flow in FIFO order, to determine the packet to be scheduled, it suffices to determine the flow to be scheduled. Hence, for each flow, let there be the following three variables:

- `eligibility_time`: This is the time at which the packet at the head of the flow queue will depart from the rate controller.
- `gsq_priority`: This is the priority value assigned by GSQ scheduler to the packet at the head of the flow queue.
- `asq_priority`: This is the start tag of the packet at the head of its flow queue.

Let all backlogged flows be inserted in a red-black tree with `eligibility_time` as the search key. Operations such as deletion/insertion of a node and finding the node (flow) with the minimum `eligibility_time` value can be performed in time $O(logQ)$ [7].

In FA at time t, the packet that has the smallest `gsq_priority` value among the set of packets that have eligibility time at most t is scheduled. If such a packet does not exist, then the packet with the minimum `asq_priority` is scheduled. To perform this search operation in time $O(logQ)$ we augment the nodes of the red-balck tree with three fields:

- `min_eligibility_time_node`: This is the node with the minimum `eligibility_time` value in the subtree rooted at the node.

- `min_gsq_priority_node`: This is the node with the minimum `gsq_priority` value in the subtree rooted at the node.
- `min_asq_priority_node`: This is the node with the minimum `asq_priority` value in the subtree rooted at the node.

Figure 6.3 presents the algorithm for determining the flow to be scheduled at time t that uses these additional fields (R is the root node of the tree and left and right denote the left and right child of a node, respectively). The essential idea in the algorithm is to traverse the tree structure to determine the node with the smallest eligibility time greater than t. During the tree traversal, we keep track of the node with minimum GSQ priority that has eligibility time at most t. The tree traversal is performed by the while loop in the algorithm and the else clause of the if statement in the while loop tracks the node with minimum GSQ priority.

To determine the complexity of the algorithm, observe that the while loop in the algorithm is executed at most N times, where N is the depth of the tree. Since $N = O(logQ)$ for a red-black tree [7], we conclude that the complexity of the algorithm is $O(logQ)$. The additional fields in the red-black tree may increase the complexity of the insertion and deletion operations. However, since these fields for a node can be computed using only the fields of the node and its immediate children, from Theorem 15.1 of [7] we know that the complexity of the insertion/deletion operations continues to remain $O(logQ)$. Since within a single packet transmission time the number of packets that can arrive at the server, and consequently the number of insertion operations, is independent of the number of flows, we conclude that $O(logQ)$ computation is performed within a single packet transmission time.

To summarize, the per packet complexity of FA algorithm in either implementations is $O(logQ)$. However, the maximum computation that may have to be performed within a single packet transmission time is $O(Q)$ and $O(logQ)$ for separate and combined implementations, respectively.

```
Schedule(R,t)
{
    if ( t > R.eligibility_time )
        return R.min_asq_priority_node
    else
    {
        min = ∞
        min_node = null
        cur = R
        while (cur ≠ null)
        {
            if (cur.eligibility_time > t)
                cur= cur.left
            else
            {
                if ( cur.gsq_priority < min )
                {
                    min = cur.gsq_priority
                    min_node = cur
                }
                if ( cur.left ≠ null)
                {
                tmp = cur.left.min_gsq_priority_node
                if (tmp.gsq_priority < min)
                {
                    min = tmp.gsq_priority
                    min_node = tmp
                }
                cur = cur.right
            }
        }
    }
    return min_node
    }
}
```

Figure 6.3: An $O(logQ)$ algorithm to determine the packet to be scheduled in FA

6.3 Fairness Guarantee

To prove fairness guarantee of Fair Airport algorithm, we first derive lower and upper bounds on the service received by a flow through GSQ in Lemmas 6.1 and 6.2, respectively. We then prove the fairness guarantee in Theorem 6.1. In what follows, a flow will be considered backlogged, unless stated otherwise, if it is backlogged at the FA server. Observe that if a flow is backlogged, it may or may not be backlogged at the GSQ server. Let $W_f^{GSQ}(t_1, t_2)$ and $W_f^{ASQ}(t_1, t_2)$ be the service received by flow f during $[t_1, t_2]$ from GSQ and ASQ, respectively. Let $\beta_f^{max} = \max\{\beta_f^j\}$ and $l_f^{max} = \max\{l_f^j\}$.

Lemma 6.1 *If the GSQ scheduler ensures (6.3), then for all intervals* $[t_1, t_2]$ *in which the flow is backlogged throughout the interval:*

$$r_f(t_2 - t_1) - r_f(\beta_f^{max} + \epsilon) - l_f^{max} \leq W_f^{GSQ}(t_1, t_2)$$

Proof: Let packet p_f^j refer to the j^{th} packet of flow f serviced by GSQ. Since flow f is backlogged during $[t_1, t_2]$ and the rate controller lets a packet into the GSQ at most ϵ before its expected arrival time, the maximum value of the expected arrival time of a packet that is served before time t is $t + \epsilon$. Hence, the maximum value of the expected arrival time of the first packet that is served via GSQ in the interval $[t_1, t_2]$, denoted by p_f^m, is:

$$EAT^{GSQ}(p_f^m, r_f) \leq t_1 + \epsilon + \frac{l_f^{max}}{r_f} \tag{6.4}$$

Let packet p_f^n be such that:

$$EAT^{GSQ}(p_f^n, r_f) + \beta_f^n \leq t_2 \tag{6.5}$$

$$t_2 \leq EAT^{GSQ}(p_f^n, r_f) + \frac{l_f^n}{r_f} + \beta_f^{n+1} \tag{6.6}$$

Since flow f is backlogged, packet p_f^n exists. Furthermore, since GSQ scheduler guarantees that packet p_f^n will depart by $EAT^{GSQ}(p_f^n, r_f) + \beta_f^n$, from (6.5) we conclude that packet

p_f^n is guaranteed to be served by time t_2. Hence, $W_f^{GSQ}(t_1,t_2) \geq \sum_{k=0}^{k=m-n} l_f^{m+k}$. Since flow is continuously backlogged in $[t_1, t_2]$,

$$EAT^{GSQ}(p_f^n, r_f) + \frac{l_f^n}{r_f} = EAT^{GSQ}(p_f^m, r_f) + \frac{\sum_{k=0}^{k=m-n-1} l_f^{m+k}}{r_f} + \frac{l_f^n}{r_f}$$

Hence, from (6.6) we get:

$$t_2 \leq EAT^{GSQ}(p_f^m, r_f) + \frac{\sum_{k=0}^{k=m-n} l_f^{m+k}}{r_f} + \beta_f^{n+1}$$

$$t_2 - \beta_f^{n+1} - EAT^{GSQ}(p_f^m, r_f) \leq W_f^{GSQ}(t_1, t_2)$$

Using (6.4) and $l_f^{n+1} \leq l_f^{max}$, we get

$$(t_2 - t_1) - (\beta_f^{max} + \epsilon) - \frac{l_f^{max}}{r_f} \leq W_f^{GSQ}(t_1, t_2)$$

Hence the lemma follows. ■

Lemma 6.2 *If the GSQ scheduler ensures (6.3), then for any interval* $[t_1, t_2]$ *in which the flow is backlogged throughout the interval:*

$$W_f^{GSQ}(t_1, t_2) \leq r_f(t_2 - t_1) + (\beta_f^{max} + \epsilon) + l_f^{max}$$

Proof: Let packet p_f^j refer to the j^{th} packet of flow f serviced by GSQ. Since GSQ scheduler ensures that packet p_f^j departs by $EAT^{GSQ}(p_f^j, r_f) + \beta_f^j$. Hence, the first packet to be serviced within $[t_1, t_2]$, p_f^m, must be such that:

$$EAT^{GSQ}(p_f^m, r_f) + \beta_f^m \geq t_1$$

Since the rate controller ensures that at all time t, the expected arrival time of the packets in GSQ is at most $t + \epsilon$, we know that last packet that is served in $[t_1, t_2]$, denoted by p_f^n, is such that:

$$EAT^{GSQ}(p_f^n, r_f) < t_2 + \epsilon$$

From the definition of expected arrival time, we know:

$$EAT^{GSQ}(p_f^n, r_f) - EAT^{GSQ}(p_f^m, r_f) \geq \sum_{k=0}^{k=m-n-1} \frac{l_f^{m+k}}{r_f}$$

$$\Rightarrow EAT^{GSQ}(p_f^n, r_f) - EAT^{GSQ}(p_f^m, r_f) + \frac{l_f^n}{r_f} \geq \sum_{k=0}^{k=m-n} \frac{l_f^{m+k}}{r_f}$$

Using the bounds on $EAT^{GSQ}(p_f^n, r_f)$ and $EAT^{GSQ}(p_f^m, r_f)$, we get:

$$(t_2 - t_1) + \epsilon + \beta_f^m + \frac{l_f^n}{r_f} \geq \sum_{k=0}^{k=m-n} \frac{l_f^{m+k}}{r_f}$$

Since $\sum_{k=0}^{k=m-n} l_f^{m+k} \geq W_f^{GSQ}(t_1, t_2)$,

$$r_f(t_2 - t_1) + r_f(\beta_f^{max} + \epsilon) + l_f^{max} \geq W_f^{GSQ}(t_1, t_2)$$

■

Theorem 6.1 *If GSQ scheduler ensure (6.3), then for any interval $[t_1, t_2]$ such that flows f and m are backlogged during the entire interval, the difference in the service received by two flows is given as:*

$$\left| \frac{W_f(t_1, t_2)}{r_f} - \frac{W_m(t_1, t_2)}{r_m} \right| \leq 2\left(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}\right) + (\beta_f^{max} + \beta_m^{max}) + 2\epsilon$$

Proof: To derive a bound on $W_f^{ASQ}(t_1, t_2)$ observe that

- Since a packet is removed from the ASQ only when it has been service either through ASQ or GSQ, whenever a flow is backlogged in the system, it is also backlogged in the ASQ.
- Whenever a packet is serviced from the GSQ, the start tag of the next packet in the ASQ is set to the start tag of the packet. Hence Lemmas 5.1 and 5.2 continue to hold.

Since the flow is backlogged in in the interval $[t_1, t_2]$, from Lemmas 5.1 and 5.2 , we conclude:

$$r_f(v_2 - t_1) - l_f^{max} \leq W_f^{ASQ}(t_1, t_2) \leq r_f(v_2 - v_1) + l_f^{max} \tag{6.7}$$

where $v_2 = v(t_2)$ and $v_1 = v(t_1)$. The theorem follows from (6.7) and Lemmas 6.1 and 6.2. ■

This fairness guarantee will hold over variable rate servers also if the GSQ scheduling algorithm ensures that (6.3) holds when the server rate varies. In Section 6.5 we will present two GSQ scheduling algorithms that ensure that (6.3) holds over variable rate servers. Thus, FA contains algorithms that are fair over variable rate servers.

6.4 Deadline Guarantees

A FA server, just like a SFQ server, provides two types of deadline guarantees: delay guarantee and delay-cum-throughput guarantee. Theorems 6.2 and 6.3 establish the delay and delay-cum-throughput guarantees, respectively.

Theorem 6.2 *The departure time of packet p_f^j in a FA server, denoted by $L_{FA}(p_f^j)$, is given as:*

$$L_{FA}(p_f^j) \leq EAT(p_f^j, r_f) + +\beta_f^j \tag{6.8}$$

Proof: Consider packet p_f^j. It may be served through ASQ or GSQ. Let us consider the two cases:

- Packet p_f^j is serviced through GSQ: Since GSQ ensures (6.3):

  $$EAT^{GSQ}(p_f^j, r_f) + +\beta_f^j \tag{6.9}$$

  It has been shown in [24] that:

  $$EAT^{GSQ}(p_f^j, r_f) = EAT^{RC}(p_f^j, r_f) \tag{6.10}$$

  when the rate controller satisfies (6.2). Since $EAT^{RC}(p_f^j, r_f) \leq EAT(p_f^j, r_f)$, from (6.9) and (6.10) we conclude that (6.8) holds for packet p_f^j.

- Packet p_f^j is serviced through ASQ: Packet p_f^j starts service from ASQ at time t only if $t < EAT^{RC}(p_f^j, r_f)$. Hence, packet p_f^j departs by $t + \beta_f^j$. Since $EAT^{RC}(p_f^j, r_f) \leq EAT(p_f^j, r_f)$, we conclude that (6.8) holds for packet p_f^j.

■

Theorem 6.3 *In a FA server the departure time of packet p_f^j is given as:*

$$L_{FA}(p_f^j) \leq \max\{L_{FA}(p_f^{j-1}), A(p_f^j)\} + \frac{l_f^{max}}{r_f} + \beta_f^j + \epsilon \tag{6.11}$$

where $L_{FA}(p_f^0) = 0$.

Proof: Let t be the time at which packet p_f^j becomes the first flow f packet to be scheduled, i.e.,

$$t = \max\{L_{FA}(p_f^{j-1}), A(p_f^j)\} \tag{6.12}$$

Then, since the last packet to be scheduled through GSQ could have a maximum packet length of l_f^{max} and expected time at the rate controller of at most $t + \epsilon$, using (6.2) we conclude:

$$EAT^{RC}(p_f^j) \leq t + \epsilon + \frac{l_f^{max}}{r_f}$$

Packet p_f^j may be served through the GSQ or ASQ. Let the packet be served through GSQ. Using (6.10) we get:

$$EAT^{GSQ}(p_f^j) \leq t + \epsilon + \frac{l_f^{max}}{r_f} \tag{6.13}$$

Since GSQ server guarantees that p_f^j will depart by $EAT^{GSQ}(p_f^j) + \beta_f^j$:

$$L_{FA}(p_f^j) \leq EAT^{GSQ}(p_f^j) + \beta_f^j \tag{6.14}$$

Using (6.13) and then (6.12) to substitute for t yields (6.11). Since a packet can be served from ASQ only earlier than it is guaranteed to depart from GSQ, the theorem follows. ■

Since SFQ guarantees a deadline to a packet based on its expected arrival time, it belongs to the GR class. Hence, its end-to-end delay guarantee is known from Theorem 2.9 in Chapter 2. Also, since the delay-cum-throughput guarantee of FA satisfies the assumptions of Theorem 3.1 in Chapter 3, its end-to-end delay-cum-throughput guarantee is also known.

6.5 Two Instantiations of Fair Airport

In this section, we instantiate the GSQ scheduling algorithm to Virtual Clock and Delay EDD to demonstrate that FA algorithms can achieve the delay guarantee of WFQ as well as separation of rate and delay allocation, respectively.

To demonstrate that Virtual Clock and Delay EDD as GSQ scheduling algorithms satisfy (6.3) we will assume that the server is a Fluctuation Constrained (FC) server defined in Chapter 5. This will enable FA server to provide deadline as well as fairness guarantees when the server rate varies over time.

Virtual Clock and Delay EDD and their delay guarantee when the servers are FC are as follows:

- Virtual Clock: The Virtual Clock scheduling algorithm is defined as follows:

  1. Packet p_f^j on its arrival is stamped with its virtual clock value, denoted by $VC(p_f^j, r_f)$, which is defined as:
  $$VC(p_f^j, r_f) = EAT(p_f^j, r_f) + \frac{l_f^j}{r_f} \tag{6.15}$$
  2. Packets are scheduled in increasing order of virtual clock values.

  In Chapter 2 we had derived the delay guarantee of Virtual Clock for constant rate servers. We extend that proof methodology to derive the delay guarantee for FC servers. To do so, define rate function of flow f, denoted by $R_f(t)$, as:
  $$R_f(t) = \begin{cases} r_f & \text{if } \exists j \ni \Big(A(p_f^j) \leq t\Big) \wedge \Big(VC(p_f^{j-1}, r_f) < t \leq VC(p_f^j, r_f)\Big) \\ 0 & \text{otherwise} \end{cases} \tag{6.16}$$
  Theorem 6.4 proves the delay guarantee of a FC Virtual Clock server.

  Theorem 6.4 *If Q is the set of flows served by a FC VC server with parameters $(C, \delta(C))$ and $\sum_{n \in Q} R_n(t) \leq C$ for all t, then the departure time of packet p_f^j at the server, denoted by $L_{VC}(p_f^j)$, is given as:*
  $$L_{VC}(p_f^j) \leq EAT(p_f^j, r_f) + \frac{l_f^j}{r_f} + \frac{l_{max}}{C} + \delta(C) \tag{6.17}$$

Proof: It is easily observed from (6.15), i.e., the definition of virtual clock value and (6.16) that the cumulative length of all flow n packets that arrive in interval $[t_1, t_2]$ and have virtual clock value no greater than t_2, denoted by $AP_n(t_1, t_2)$, is given as:

$$AP_n(t_1, t_2) \leq \int_{t_1}^{t_2} R_n(t)dt$$

Consider packet p_f^j served in the busy period beginning at time t_0. Let $t_2 = VC(p_f^j, r_f)$ and let t_1 be the largest time less than t_2 at which a packet that has virtual clock value greater than t_2 is scheduled in the busy period. If such a packet does not exist, set $t_1 = t_0^-$. Then, since packets are served in the increasing order of virtual clock values, all the packets scheduled in the interval $(t_1, L_{VC}(p_f^j)]$ arrive in the interval $(t_1, t_2]$ and have virtual clock value at most t_2. Hence, the aggregate length of packets scheduled in the interval $(t_1, L_{VC}(p_f^j)]$ is $\sum_{n \in Q} AP_n(t_1^+, t_2)$. Consequently, the aggregate length of packets served in the interval $[t_1, L_{VC}(p_f^j)]$ is at most $l_{max} + \sum_{n \in Q} AP_n(t_1^+, t_2)$. Since the server is FC, we get:

$$\begin{aligned}
L_{VC}(p_f^j) &\leq t_1 + \frac{l_{max} + \sum_{n \in Q} AP_n(t_1^+, t_2)}{C} + \delta(C) \\
&\leq t_1 + \frac{\int_{t_1^+}^{t_2} \sum_{n \in Q} R_n(t)dt}{C} + \frac{l_{max}}{C} + \delta(C) \\
&\leq t_1 + \frac{C(t_2 - t_1)}{C} + \frac{l_{max}}{C} + \delta(C) \\
&\leq t_2 + \frac{l_{max}}{C} + \delta(C)
\end{aligned}$$

Since $t_2 = VC(p_f^j, r_f) = EAT(p_f^j, r_f) + \frac{l_f^j}{r_f}$, the theorem follows. ■

We observe that this proof method can also be employed to derive the delay guarantee of generalized Virtual Clock scheduling algorithm, presented in Chapter 2, for FC servers. Thus, Virtual Clock satisfies (6.3) with $\beta_f^j = \frac{l_f^j}{r_f} + \frac{l_{max}}{C} + \delta(C)$. Hence, using Theorem (6.2) we conclude that the departure time of packet in the FA server is given as:

$$L_{FA}(p_f^j) \leq EAT(p_f^j, r_f) + \frac{l_f^j}{r_f} + \frac{l_{max}}{C} + \delta(C)$$

This is the same delay guarantee as WFQ when $\delta(C) = 0$. Since the admission control equation requires only checking that aggregate bandwidth does not exceed the capacity, we conclude that a FA algorithm with Virtual Clock as the GSQ scheduler achieves the delay guarantee of WFQ with $O(1)$ complexity admission control algorithm.

- Delay EDD: Delay EDD is defined only when packet lengths of a flow do not vary. Hence, we assume that $l_f = l_f^j$. Then Delay-EDD is defined as follows:

  1. If d_f is the delay bound for flow f at server i, then on arrival packet p_f^j is stamped with a deadline, denoted by $D(p_f^j)$, which is computed as follows:

  $$D(p_f^j) = EAT(p_f^j, r_f) + d_f \tag{6.18}$$

  2. Packets are served in the increasing order of deadline.

Theorem 6.5 *If Q is the set of flows serviced by the server and*

$$\forall t > 0 : \quad \sum_{n \in Q} \max\{0, \left\lceil \frac{(t - d_n) r_n}{l_n} \right\rceil \frac{l_n}{C}\} \leq t \tag{6.19}$$

and the server is a $(C, \delta(C))$ Fluctuation Constrained server, then the time at which the transmission of packet p_f^j is completed in Delay EDD, denoted by $L_{EDD}(p_f^j)$ is:

$$L_{EDD}(p_f^j) \leq EAT(p_f^j, r_f) + d_f + \frac{l_{max}}{C} + \delta(C) \tag{6.20}$$

where l_{max} is the maximum length of packet served by the server.

Proof: From (6.18) and (6.1), we conclude that the cumulative length of packets of flow n that arrive in the interval $[t_1, t_2]$ and have deadline less than or equal to t_2, denoted by $AP_n(t_1, t_2)$, is given as:

$$AP_n(t_1, t_2) \leq \left\lceil \frac{(t_2 - t_1 - d_n) r_n}{l_n} \right\rceil l_n \tag{6.21}$$

We now prove the theorem by contradiction. Let (6.20) not hold for packet p_f^j. Also, let t_0 be the beginning of the busy period in which p_f^j is served and $t_2 = D(p_f^j)$. Let

t_1 be the greatest time less than t_2 during the busy period at which a packet with deadline greater than t_2 is scheduled. If such a t_1 does not exist, set $t_1 = t_0^-$. Since the packets are scheduled in the earliest deadline first order, all the packets scheduled in the interval $(t_1, L_{EDD}(p_f^j)]$ arrive in the interval $(t_1, t_2]$ and have deadline at most t_2. Hence, the aggregate length of packets served in the interval $[t_1, L_{EDD}(p_f^j)]$ is at most $l_{max} + \sum_{n \in Q} AP_n(t_1^+, t_2)$. Since the server is fluctuation constrained, we get:

$$L_{EDD}(p_f^j) \leq t_1 + \frac{l_{max} + \sum_{n \in Q} AP_n(t_1^+, t_2)}{C} + \delta(C)$$

Using (6.21) and (6.19) with $t = t_2 - t_1^+$, we get:

$$\begin{aligned} L_{EDD}(p_f^j) &\leq t_1 + \frac{l_{max}}{C} + t_2 - t_1^+ + \delta(C) \\ &\leq t_2 + \frac{l_{max}}{C} + \delta(C) \end{aligned}$$

This contradicts the assumption that $L_{EDD}(p_f^j) > t_2 + \frac{l_{max}}{C} + \frac{\delta(C)}{C}$. Hence such a packet p_f^j does not exist. Since $t_2 = D(p_f^j) = EAT(p_f^j, r_f) + d_f$, the theorem follows. ∎

Thus, Delay EDD satisfies (6.3) with $\beta_f^j = d_f + \frac{l_{max}}{C} + \delta(C)$. Hence, using Theorem (6.2) we conclude that the departure time of packet in the FA server is given as:

$$L_{FA}(p_f^j) \leq EAT(p_f^j, r_f) + d_f + \frac{l_{max}}{C} + \delta(C)$$

Since d_f can be chosen to be less than $\frac{l_f}{r_f}$, equal to $\frac{l_f}{r_f}$, or more than $\frac{l_f}{r_f}$, separation of rate and delay allocation in FA with Delay EDD as the GSQ scheduler is achieved.

Due to high computational complexity, it may not be feasible to employ (6.19) as the schedulability test. Hence, conditions stronger than (6.19) that have lower computational complexity have been developed in [64]. The delay guarantee holds under the stronger conditions as well. The computational complexity of the stronger conditions is $O(Q)$.

6.6 Salient Features of Fair Airport

Some of the salient features of the class of Fair Airport algorithms are:

- As demonstrated in Section 6.5, it contains algorithms that have the delay guarantee of WFQ as well as those that achieve separation of rate and delay allocation.

- As demonstrated in Section 6.5, Virtual Clock and Delay EDD ensure (6.3) when the server is Fluctuation Constrained. Hence, FA contains scheduling algorithms that are fair over variable rate servers.

- It contains algorithms with different implementation complexity and performance guarantees. For example, a static priority GSQ scheduler is more efficient than a Delay EDD scheduler. However, whereas the former can support only predetermined set of delay vectors, the latter can support any [59].

- The concept of Fair Airport algorithm is general. The rate controller can be generalized as in [19] to enforce different traffic constraints. It can be shown that the generalized FA class continues to remain fair.

- The FA algorithms can be generalized to achieve prioritized fair allocation of residual bandwidth which may be desirable to support high throughput flows in max-min fair networks [38]. To achieve this objective, let the set of flows served by a server be partitioned into various priority classes. Then, FA is generalized by changing the ASQ scheduler to an algorithm that provides non-preemptive static priority to the flows of higher priority classes and schedules packets belonging to the same priority class using SFQ (see Figure 6.4). It can be observed from the proof of the fairness guarantee of FA algorithms that the fairness guarantee continues to hold for any two flows that belong to the same priority class. Furthermore, the deadline guarantees do not change. Thus, the generalized FA algorithm enables prioritized fair allocation of residual bandwidth without affecting the deadline guarantees.

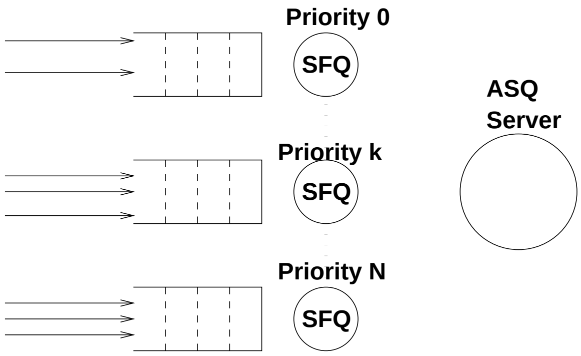


Figure 6.4: Generalization of Fair Airport ASQ scheduler to achieve prioritized fair allocation of residual bandwidth

6.7 Conclusions

In this chapter, we designed a class of Fair Airport algorithms that combines SFQ with a non-work conserving algorithm in RCSD class. We derived fairness and deadline guarantees of FA servers and demonstrated that by appropriately choosing an algorithm from RCSD class, algorithms that either allocate only rate or achieve separation of rate and delay allocation and are fair over FC variable rate servers can be designed. This method led to the design of the *first* fair algorithm that achieves separation of rate and delay allocation. We also demonstrated that the FA class contains algorithms with different implementation complexity and performance guarantees and thus enables tradeoffs between the two. Finally, we demonstrated that the FA algorithms can be generalized to achieve other objectives such as prioritized fair allocation of residual bandwidth.

In summary, we demonstrated that FA class is fair and contains algorithms that: (1) can achieve separation of rate and delay allocation and thus provide low delay to low throughput applications; (2) are fair over variable rate FC servers; (3) provides QoS guarantees over variable rate servers; (4) are computationally efficient. In Chapter 7, we show that it enables hierarchical link bandwidth allocation, and thus meets the requirements that we had outlined in Chapter 4.

Chapter 7

Hierarchical Link Bandwidth Allocation

An integrated services network will provide several services as well as employ multiple protocol families that support different traffic types and/or congestion control mechanisms. Since these services as well as the protocols may be incompatible with each other, a mechanism that provides isolation between different services and protocols and enables their coexistence is required. An integrated services network will also support a *link-sharing* service in which the bandwidth of a link is allocated to several organizations and the bandwidth of an organization is recursively allocated to its suborganizations. As argued in Chapter 4, hierarchical link bandwidth allocation is a suitable mechanism for enabling coexistence of heterogeneous services as well as providing a *link-sharing* service.

In this chapter, we present scheduling algorithms for achieving hierarchical allocation of link bandwidth. We employ SFQ and FA to design *Hierarchical SFQ and FA* algorithms. We demonstrate that the hierarchical scheduling algorithms achieve the objectives of hierarchical link bandwidth allocation. We then analyze the performance guarantees that can be provided to flows when a link bandwidth is hierarchically allocated. Our analysis technique is general and applicable for other hierarchical schedulers constructed using algorithms that provide guarantees similar to SFQ and FA. Furthermore, this analysis technique is the first in the literature to yield tight performance bounds for hierarchical scheduling algorithms. Finally, we present an implementation of Hierarchical SFQ as a streams module for scheduling packets in an IPv6 router.

The rest of the chapter is organized as follows. We present the requirements of hierarchical link bandwidth allocation in Section 7.1. Hierarchical SFQ and FA are presented and analyzed in Sections 7.2 and 7.3, respectively. The performance guarantees for a flow in hierarchical schedulers are derived in Section 7.4. We present the implementation of Hierarchical SFQ in Section 7.5. Finally, Section 7.6 summarizes the results of this chapter.

7.1 Requirements

The requirements of hierarchical link bandwidth allocation are specified by a tree. We refer to a node in the tree as a class. A class may correspond to a service class, an organization, or an aggregation of flows. Each flow at a server belongs to one leaf class. Packets of a flow are scheduled by the scheduler for the leaf class to which the flow belongs. Each leaf class may employ a different scheduling algorithm. Each class in the tree has a weight assigned to it. The objective of a mechanism implementing hierarchical link bandwidth allocation is to distribute the bandwidth allocated to a class among its subclasses in proportion to the weights, i.e., fairly.

To illustrate, consider the tree shown in Figure 7.1. Whereas classes labeled "Agency X" and "Agency Y" correspond to organizations, classes labeled "Real-time" and "Best-effort" correspond to services. Also, the "Real-time" classes in agencies X and Y are shown to employ SFQ and FA as the scheduling algorithms, respectively. The weights of agencies X and Y are shown in parenthesis and are 2 and 1, respectively. Hence, the link bandwidth is allocated in the ratio 2:1 among agencies X and Y.

The objective of hierarchical link bandwidth allocation can be achieved by a *hierarchical scheduler* that considers each class, other than the leaf classes, as a virtual server and uses a *fair* scheduler to schedule the virtual servers. The following example illustrates that the scheduler used must allocate bandwidth fairly over variable rate servers.

Example 7.1 *Consider the tree shown in Figure 7.1. Initially, let no flows be backlogged in*

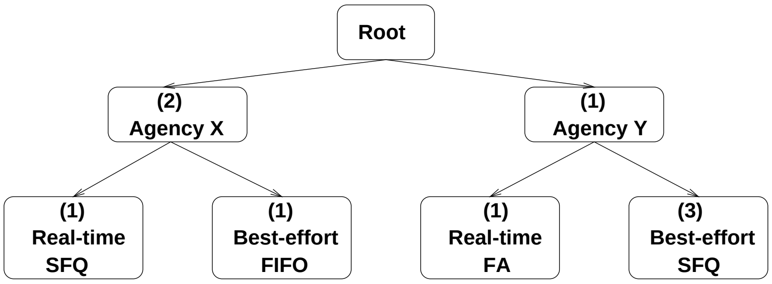


Figure 7.1: An example specification of hierarchical link bandwidth allocation

agency Y. Hence, agency X gets the full link bandwidth. When flows become backlogged in agency Y, the bandwidth available to agency X (and hence to its subclasses) reduces to $\frac{2}{3}$ of the link bandwidth. Consequently, to fairly allocate the bandwidth of agency X between its real-time and best-effor subclasses, the scheduler must be able to allocate bandwidth fairly over variable rate servers.

Since SFQ allocates bandwidth fairly regardless of variation in server capacity, a Hierarchical SFQ scheduler achieves the objectives of hierarchical link bandwidth allocation. Also, since FA is fair over Fluctuation Constrained variable rate servers, Hierarchical FA can also achieve the desired objectives. In the following sections, we present Hierarchical SFQ and FA scheduling algorithms and demonstrate that they achieve the desired objectives.

7.2 Hierarchical SFQ

Hierarchical SFQ scheduler is a recursive scheduler. It uses SFQ to schedule each class; treating each subclass as a flow. The scheduling of packets occurs recursively: the scheduler for root class schedules the subclasses; the scheduler of subclasses in turn schedule their subclasses. The leaf classes are scheduled by employing a leaf class dependent scheduler.

Since SFQ fairly allocates bandwidth regardless of a server behavior, Hierarchical SFQ ensures that the bandwidth of a class is allocated to its subclasses in proportion to their weights. Hence, Hierarchical SFQ achieves the objectives of hierarchical link bandwidth allocation.

The fairness guarantee of Hierarchical SFQ ensures that a class will receive a fair allocation of its parent class's bandwidth. However, it is not sufficient to characterize the bandwidth received by a class. Since the objectives of some of the classes may be to provide various performance guarantees to flows, a characterization of the bandwidth received is desirable. To achieve this objective, we first prove the following corollaries of Theorems 5.6 and 5.7.

To derive the corollaries, we will interpret the weight of a flow as the rate assigned to the flow. For example, let the link bandwidth in Figure 7.1 be $90Mb/s$. Then since the the weights of agencies X and Y are 2 and 1, they are guaranteed to received $\frac{2}{3}$ and $\frac{1}{3}$ of the link bandwidth, respectively. Hence, weights of agencies X and Y are interpreted as $60Mb/s$ and $30Mb/s$, respectively. The rates of the other classes can be similarly derived. Let $T_f(w)$ denote the time taken to serve flow f packets of aggregate length w which are served in the same backlogged period of the flow. The symbols used here are the same as those in Chapter 5.

Corollary 7.1 *If the capacity of a SFQ FC server with parameters $(C, \delta(C))$ is not exceeded and $r_f^j = r_f$ for all packets, then $T_f(w)$ is given as:*

$$T_f(w) \leq \frac{w}{r_f} + \frac{l_f^{max}}{r_f} - \alpha_f + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \delta(C) \tag{7.1}$$

where $\alpha_f = \min\{\frac{l_f^j}{r_f} - \frac{l_f^j}{C}\}$ and the minimum is over all the flow f packets.

Proof: Let $w = \sum_{n=k}^{n=j} l_f^n$. Also, let $A(p_f^j) = \max\{L_{SFQ}(p_f^{k-1}), A(p_f^k)\}$. Then using $t = \max\{L_{SFQ}(p_f^{k-1}), A(p_f^k)\}$ and $r_f^j = r_f$ in Theorem 5.6, we get:

$$L_{SFQ}(p_f^j) \leq t + \frac{l_f^{k-1}}{r_f} + \frac{\sum_{n=k}^{n=j-1} l_f^n}{r_f} + \frac{l_f^j}{C} + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \delta(C)$$

Adding $\frac{l_f^j}{r_f} - \frac{l_f^j}{r_f}$ on the right hand side and using $l_f^{k-1} \leq l_f^{max}$, we get:

$$L_{SFQ}(p_f^j) - t \leq \frac{l_f^{max}}{r_f} + \frac{\sum_{n=k}^{n=j} l_f^n}{r_f} - (\frac{l_f^j}{r_f} - \frac{l_f^j}{C}) + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \delta(C)$$

Since $L_{SFQ}(p_f^j) - t = T_f(w)$, $w = \sum_{n=k}^{n=j} l_f^n$, and $\alpha_f = \min\{\frac{l_f^j}{r_f} - \frac{l_f^j}{C}\}$, we get (7.1) when $A(p_f^j) = t$. Now, when a flow is backlogged, the start tags assigned to packets of the flow, and consequently the bound on the departure time of a packet, is the same regardless of the actual arrival time of the packets. Hence, we conclude (7.1) holds even when $A(p_f^j) \neq t$. Thus the corollary follows. ■

Corollary 7.2 *If the capacity of a SFQ EBF server with parameters $(C, B, \alpha, \delta(C))$ is not exceeded and $r_f^j = r_f$ for all packets, then random variable $T_f(w)$ is given as:*

$$P\left(T_f(w) \leq \frac{w}{r_f} + \frac{l_f^{max}}{r_f} - \alpha_f + \sum_{n \in Q \wedge n \neq f} \frac{l_n^{max}}{C} + \delta(C) + \gamma\right) \geq 1 - Be^{-\alpha\gamma} \tag{7.2}$$

where $\alpha_f = \min\{\frac{l_f^j}{r_f} - \frac{l_f^j}{C}\}$ and the minimum is over all the flow f packets.

Proof: The proof is similar to that of Corollary 7.1 with the only change that Theorem 5.7 is used for $L_{SFQ}(p_f^j)$. ■

Now, consider a class f that is a subclass of the root class. Let the link be a FC server with parameters $(C, \delta(C))$ and let the set of the subclasses of the root class be denoted by Q. Then, if class f has been assigned rate r_f, from Corollary 7.1 we conclude that the virtual server corresponding to f is a FC server with parameters:

$$\left(r_f, \frac{l_f^{max}}{r_f} - \alpha_f + \frac{\sum_{n \in Q \wedge n \neq f} l_n^{max}}{C} + \delta(C)\right) \tag{7.3}$$

Similarly, using Corollary 7.2, we conclude that if the link is an EBF server, then the virtual server corresponding to f is a EBF server. Using the argument recursively, we conclude that if the link is a FC or EBF server, then each of the virtual server in the tree structure is a FC or EBF server, respectively. Thus, the objective of characterizing the bandwidth of each class has been achieved.

To summarize, Hierarchical SFQ ensures that the bandwidth of each class is allocated among its subclasses in proportion to the weights of the subclasses. Furthermore, when the link can be characterized as a FC or EBF server, it guarantees that the virtual servers corresponding to the classes will be FC or EBF servers, respectively.

7.3 Hierarchical FA

Hierarchical FA is similar to Hierarchical SFQ; the only difference is that it uses FA instead of SFQ to schedule the subclasses.

To demonstrate that Hierarchical FA achieves the objectives of hierarchical link bandwidth allocation, we will need to make an assumption. Recall from Chapter 6 that algorithms in FA class were demonstrated be fair only when a server is a FC server. Hence, a class will receive bandwidth fairly from its parent class only if the parent class can be modeled as a FC server. Consequently, Hierarchical FA will achieve the desired objectives only if each of the virtual servers in the tree can be characterized as FC servers. To demonstrate that each of the virtual servers are FC servers, we assume that the link is a FC server.

Consider a class f that is a subclass of the root class. Then, since the link is a FC server, from Lemma 6.1 presented in Chapter 6 we know that in any time interval $[t_1, t_2]$ in which class f is continuously backlogged, the service received by it in $[t_1, t_2]$, denoted by $W_f(t_1, t_2)$, is given as:

$$r_f(t_2 - t_1) - r_f(\beta_f^{max} + \epsilon) - l_f^{max} \leq W_f(t_1, t_2)$$

Hence, we conclude that the time taken to serve packets of length w is given as:

$$T_f(w) \leq \frac{w}{r_f} + \beta_f^{max} + \epsilon + \frac{l_f^{max}}{r_f} \tag{7.4}$$

From (7.4) we conclude that if the link is a FC server, then each of the subclasses of the root class are FC servers. Using this argument recursively, we conclude that each of the virtual servers in the tree are FC servers. Thus, we conclude that Hierarchical FA achieves the objectives of hierarchical link bandwidth allocation. Furthermore, (7.4) characterizes the bandwidth of each of the virtual servers.

To summarize, Hierarchical FA achieves the objectives of hierarchical link bandwidth allocation when the link is a FC server. Furthermore, it guarantees that each of the virtual servers in the hierarchy are FC servers.

7.4 Performance Guarantees

In the previous sections, we presented Hierarchical SFQ and FA and demonstrated that they achieve the objectives of hierarchical link bandwidth allocation. In this section, we determine the performance guarantees that can be provided to flows by such scheduling algorithms.

In case of hierarchical link bandwidth allocation, a flow belongs to a leaf class and is scheduled by a leaf class dependent scheduler. Hence, to determine the performance guarantees for a flow, we need to:

- characterize the virtual server corresponding to the leaf class; and
- determine the performance guarantees of the leaf class scheduling algorithm for the server characterization.

As we demonstrated in Section 7.2, Hierarchical SFQ ensures that if the root class is either a FC or EBF server, then each of the virtual servers in the hierarchy, and consequently the leaf virtual servers, are FC or EBF servers, respectively. The parameters for the virtual servers are determined using Corollaries 7.1 and 7.2. Similarly, Hierarchical FA guarantees that if the root class is FC, then each of the leaf virtual servers are FC (the parameters are determined using (7.4). Hence, the leaf virtual servers can be characterized as either FC or EBF servers. Thus, to determine the performance guarantees for a flow, it suffices to determine the guarantees offered by a leaf class scheduler when a server is either FC or EBF.

We now determine the performance guarantees for a flow when SFQ, FA, Virtual Clock, or Delay EDD are employed as the leaf class scheduling algorithm.

- SFQ: Theorem 5.2, Theorem 5.4, and Corollary 5.1 yield the throughput guarantee, delay guarantee, and delay-cum-throughput guarantee for FC servers, respectively. Theorem 5.3, Theorem 5.5, and Corollary 5.2 yield the corresponding results for EBF servers.
- FA: Theorems 6.2 and 6.3 yield the single server delay and delay-cum-throughput guarantee for FC servers, respectively.
- Virtual Clock: Theorem 6.4 yields the single server delay guarantee for FC servers.
- Delay EDD: Theorem 6.5 yields the the single server delay guarantee for FC servers.

The end-to-end delay and delay-cum-throughput guarantees when a server is either FC or EBF are derived using Corollary 5.3 and Theorem 5.9, respectively.

The above analysis method is general and can be employed for any hierarchical scheduling algorithm that provides guarantees similar to Hierarchical SFQ and FA, i.e., ensures that the virtual servers are FC or EBF servers. Furthermore, this analysis is tighter than the analysis presented in [3]. To observe this, consider a tree with three classes; two leaf classes and a root class. Let the rate of leaf classes 1 and 2 be r_1 and r_2, respectively, and let each of them contain 2 flows with equal weights. Let both the leaf classes be scheduled by SFQ and the length of all packets be l. Then, it can be shown that the best bound on delay of packet p_f^j for flow f in leaf class 1 using the analysis in [3] is:

$$EAT(p_f^j, \frac{r_1}{2}) + (\frac{l}{r_1} + \frac{2l}{C}) + \frac{2l}{r_1} \tag{7.5}$$

In contrast using our analysis, we get:

$$EAT(p_f^j, \frac{r_1}{2}) + (\frac{2l}{C}) + \frac{2l}{r_1} \tag{7.6}$$

Clearly, (7.6) is smaller than (7.5). Hence, our analysis is the first in the literature to yield tight performance bounds for hierarchical scheduling algorithms.

7.5 Implementation and Experimental Evaluation

We have implemented Hierarchical SFQ in a streams module for scheduling Internet Protocol version 6 (IPv6) packets in a Pentium computer that is running Solaris 5.5 operating system and is configured as a router. We first describe our implementation in Section 7.5.1 and then present experimental results from the implementation in Section 7.5.2.

7.5.1 Implementation

To understand our implementation, first briefly consider a networking subsystem implemented using the streams architecture [43]. In streams architecture, a packet is processed by a sequence of modules (see Figure 7.2). The terminating module is called a driver. Hence, the module for a network interface card is a driver. The interaction between modules occurs through queues. Each module has two queues: a write queue and a read queue. The write and read queues are

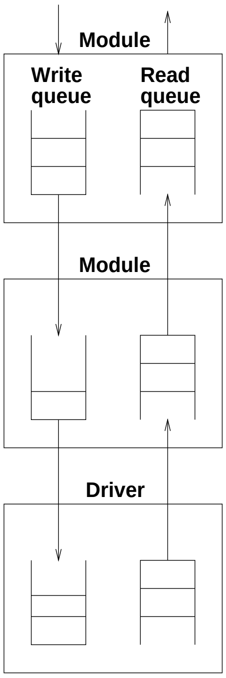


Figure 7.2: Streams architecture

used for transmit and receive side of packet processing, respectively. When a packet is to be transmitted, it is put into the write queue of the first module. The first module processes the packet and puts it into the next module's write queue. This procedure is recursively carried out until the packet reaches the driver. The driver transmits the packet on the network. Packet processing is carried out similarly using the read queues when a packet is received by a driver. We are now ready to explain our implementation.

Hierarchical SFQ is implemented in a streams module (HSFQ module) that is inserted between the IPv6 module and a network device driver (see Figure 7.3). On receiving a packet from the device driver, HSFQ module passes on the packet to the IPv6 module without processing it. However, on receiving a packet from the IPv6 module, HSFQ module enqueues it in its local queue that is used to implement Hierarchical SFQ. Ideally, HSFQ module should ensure that there is at most one packet in the network device driver and schedule a packet using Hierarchical SFQ whenever the packet in the device driver finishes service. However, some device drivers yield a very low throughput when such an approach is used. This is because they intro-

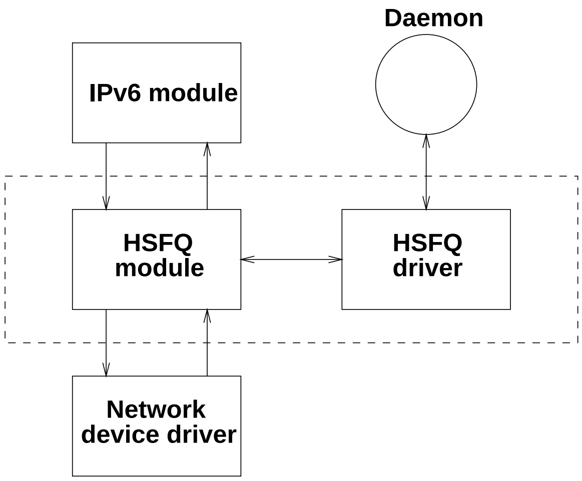


Figure 7.3: Hierarchical SFQ implementation

duce a delay between the time a packet finishes transmission and the upper module is notified of that event. To ensure that there is no throughput loss in such drivers, HSFQ module ensures that there are at most W packets in the network driver where W is a tunable parameter. When a new packet can be enqueued in the driver, it determines the packet to be enqueued using the Hierarchical SFQ scheduling algorithm. In Hierarchical SFQ, leaf classes can employ different scheduling algorithms. However, in our implementation all the leaf classes employ SFQ. We now present the data structures and the main algorithms in our implementation.

Hierarchical SFQ is implemented using a tree. The leaf nodes in the tree are flows and the non-leaf nodes are classes. To create the tree structure, we have implemented a HSFQ streams driver. HSFQ streams driver accepts control messages (sent via $ioctl()$ system calls) to create and delete classes and flows, set their weights, and change their weights. At the system boot time, a daemon process creates a tree structure specified using a configuration file.

There are two types of nodes in the tree: flow and class nodes. Let us look at the structure of the two types of nodes. The important fields that we maintain for a flow node are:

- `weight`: this is the weight of the flow.
- `finish_tag`: this is the finish tag of the last packet that was scheduled from the flow.
- `fifo_queue`: this is a bounded FIFO queue of the packets of the flow. We have imple-

mented it using a circular array.

- `parent`: this is a reference to the parent class of the flow.

A class has two roles: (1) it is a server for its children nodes; and (2) it is a flow from the perspective of its parent node. We maintain different fields for these two roles. The fields that are maintained for its role as a server are:

- `virtual_time`: this is the virtual time of the server.
- `max_finish_tag`: this is the maximum finish tag assigned to any of the immediate children of the class. This field is required since in SFQ the virtual time of a server is set to the maximum of the finish tags of the flows when the server becomes idle.
- `priority_queue`: this is a priority queue of logical packets. A logical packet is a representation of the packet that is queued at the server. It has the following fields:
  - `flow`: this is a reference to the flow or class to which the logical packet belongs. In a leaf class (a class that has only flows as its children), this field is a reference to a flow; otherwise it is a reference to a class.
  - `length`: this is the length of the logical packet.
  - `start_tag`: this is the start tag of the logical packet.

  The priority queue is maintained as a binary heap using start tag of the packets as the key. We assume the number of backlogged flows at a server is bounded and implement the binary heap using an array [7].

The fields of a class that are maintained for its role as a flow are `weight` and `finish_tag`. These fields have the same semantics as in the case of a flow. Additionally, a class has a list of its children nodes and a reference to its parent node.

A HSFQ module has to handle two main events: (1) enqueuing of a packet by the IPv6 module; and (2) selection of a packet to be enqueued in the device driver when a packet has been transmitted on the network and packets are backlogged in the HSFQ module. These two events are handled by the $enqueue()$ and $dequeue()$ functions presented in Figures 7.4 and 7.5, respectively. These two functions use the following primitives:

- $get\_flow(p)$: This function determines the flow to which packet p belongs.
- $BACKLOGGED(n)$: This function determines if node n in the tree is backlogged.
- Physical queue manipulation functions: $add\_physical(f, p)$ adds packet p to the queue of physical packets at flow f; $delete\_physical(f)$ deletes the packet at the head of the queue of flow f, and $head(f)$ returns a reference to the packet at the head of the queue of flow f.
- Logical queue manipulation functions: $add\_logical(s, f, l, S)$ inserts a logical packet with length l and start tag S for flow f in the priority queue of logical packets at server s (which is a class node); $delete\_min\_logical(s)$ deletes the logical packet with the minimum priority from the priority queue at server s; and $min\_priority\_logical(s)$ returns a reference to the logical packet with the minimum priority.

The algorithms for the $enqueue()$ and $dequeue()$ functions are as follows:

- $enqueue(p)$: We first insert packet p into its flow queue. If the flow was not backlogged prior to the arrival of the packet, then some of the ancestors of the flow may become backlogged. In such a case, we traverse up the path from the flow to the root node and make all the ancestors that were not backlogged previously, backlogged. If a node makes transition to backlogged state, a logical packet corresponding to the packet in service at that node is inserted in the priority queue of the parent node.
- $dequeue()$: We traverse the tree starting from the root node. At each step in traversal, we: (1) delete the logical packet in service at the current node, (2) update the finish tag of the node to which the logical packet belongs, and (3) set the current node to the child node to which the logical packet belongs. On reaching a leaf node, we delete the packet from the head of the physical queue at the leaf node and enqueue it into the device driver. We then determine the next packet to be scheduled at each of the servers on the path from the

```
enqueue(packet)
{
    flow = get_flow(flowid);
    backlogged= BACKLOGGED(flow);
    add_physical(flow,packet);
    server= flow.parent;
    while (not backlogged and server ≠ NULL)
    {
        start_tag= max{server.virtual_time, flow.finish_tag};
        backlogged= BACKLOGGED(server.parent);
        add_logical(server,flow,packet.length, start_tag);
        flow=server;
        server= server.parent;
    }
}
```

Figure 7.4: The algorithm for enqueuing a packet

leaf node to the root node using the $schedule()$ function presented in Figure 7.6. In the $schedule()$ function, we traverse the path from a leaf node to the root node. If a node on the path is not backlogged, then we set its virtual time to the maximum finish tag assigned to any of its children. Otherwise, we determine the logical packet in service at the node, set the virtual time of the node to be the start tag of the packet in service, and insert a logical packet in the parent node.

This completes the description of the main data structures and algorithms of our implementation. In the next section, we present experimental evaluation of the implementation.

7.5.2 Experimental Evaluation

To evaluate our implementation of Hierarchical SFQ, we have set up a network of 4 Pentium 150 MHz computers connected by 100 Mb/s point-to-point ethernet links (see Figure 7.7). These computers are running Solaris 2.5 operating system and IPv6 protocol. We have configured Bharani as the router and pushed HSFQ module on the network interface connecting Bharani and Shravan.

For most of our experiments, we used the hierarchical structure shown in Figure 7.8.

```
dequeue()
{
    server= ROOT;
    while (server.type ≠ FLOW)
    {
        logical_packet= delete_min_logical(server);
        flow= logical_packet.flow;
        length= logical_packet.length;
        start_tag= logical_packet.start_tag;
        flow.finish_tag= start_tag + length/flow.weight;
        server.max_finish_tag = max{server.max_finish_tag, flow.finish_tag};
        server=flow;
    }
    scheduled_packet= delete_physical(flow);
    enqueue_device(scheduled_packet);
    schedule(flow);
}
```

Figure 7.5: The algorithm for dequeuing a packet

Flows 1 and 2 originated from Revati and terminated at Shravan. Flows 3 and 4 originated from Hasta and terminated at Shravan. All the flows were TCP flows and used a sender and receiver window size of 20KB. All the flows were assigned the same weights. In most of our experiments, flows 1,2,3, and 4 terminated after sending 50,000, 150,000, 100,000, and 200,000 4KB packets, respectively. All the flows were started almost simultaneously.

We conducted experiments to determine the overhead of employing the HSFQ module and demonstrate that Hierarchical SFQ achieves hierarchical allocation of bandwidth as well as enables coexistence of different congestion control techniques. Specifically, we conducted the following experiments:

- *Overhead evaluation*: To determine the overhead of employing HSFQ, we first determined the throughput achievable by the flows when the router was not modified. We then pushed a module that implements a FCFS queuing algorithm (using the streams queuing functions) on the network interface connecting Bharani and Hasta and then determined the throughput of all the flows. Finally, we determined the throughputs with the HSFQ module. In case of HSFQ, we assigned equal weights to Revati and Hasta classes.

```
schedule(flow)
{
    packet= head(flow);
    server= flow.parent;
    if (packet ≠ NULL)
    {
        start_tag= flow.finish_tag;
        add_logical(server,flow,packet.length,start_tag);
    }
    while( server ≠ NULL)
    {
        if (not BACKLOGGED(server))
        {
            server.virtual_time= server.max_finish_tag;
            server=server.parent;
        }
        else
        {
            logical_packet= min_priority_logical(server);
            length= logical_packet.length;
            start_tag= logical_packet.start_tag;
            server.virtual_time = start_tag;

            server=server.parent;
            flow=server;
            start_tag=flow.finish_tag;
            add_logical(server,flow,length,start_tag);
        }
    }
}
```

Figure 7.6: The algorithm for scheduling a packet

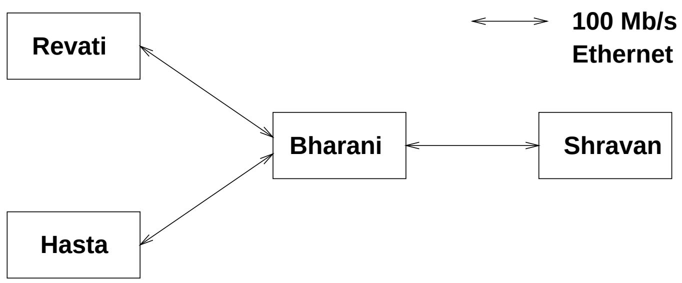


Figure 7.7: The network configuration

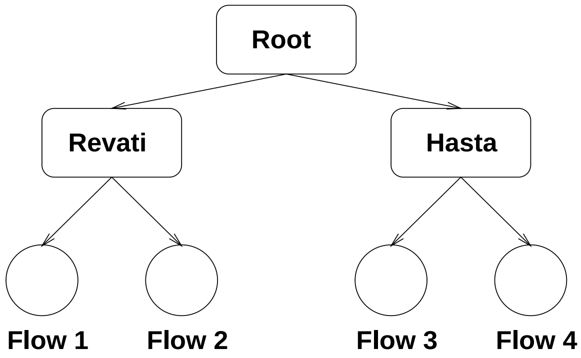


Figure 7.8: The hierarchical structure employed for experimental evaluation

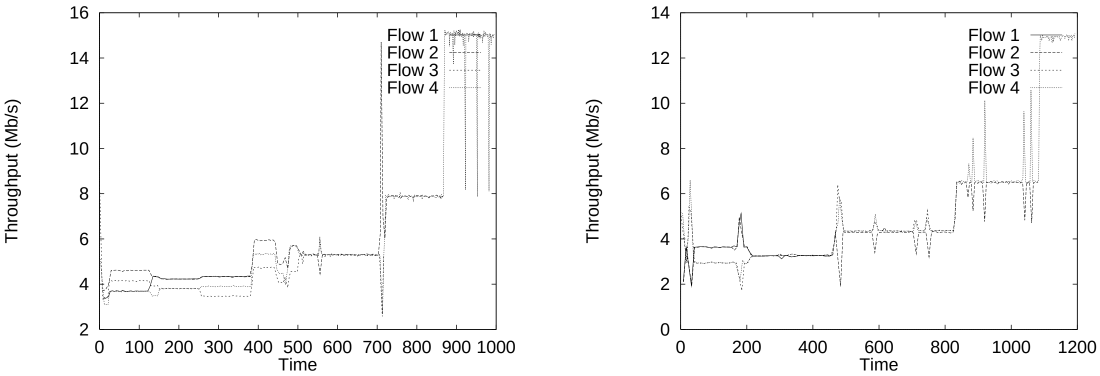


Figure 7.9: Throughput of the flows: (a) without any module between IPv6 and device driver and (b) a FCFS module between them

Figures 7.9(a), 7.9(b), and 7.10 plot the variation in the throughput of the flows with time for the unmodified router, the router with FCFS module, and the router with HSFQ module, respectively. As the figures illustrate, the aggregate throughputs of all the flows was 16.12 Mb/s, 13.16 Mb/s, and 12.85 Mb/s in case of the unmodified router, FCFS module, and HSFQ module, respectively. This demonstrates that a FCFS module introduces a 18.3% overhead and the Hierarchical SFQ scheduling algorithm introduces a further 2.35% overhead.

To evaluate the effect of increase in the number of flows on the overhead of Hierarchical SFQ algorithm, we initiated five TCP flows in both Revati and Hasta classes. These flows terminated after transmitting 5,000 4KB packets. Figures 7.11(a) and 7.11(b) plot

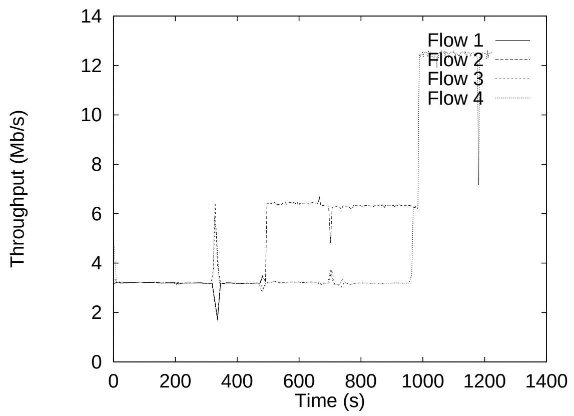


Figure 7.10: Throughput of the flows with HSFQ module

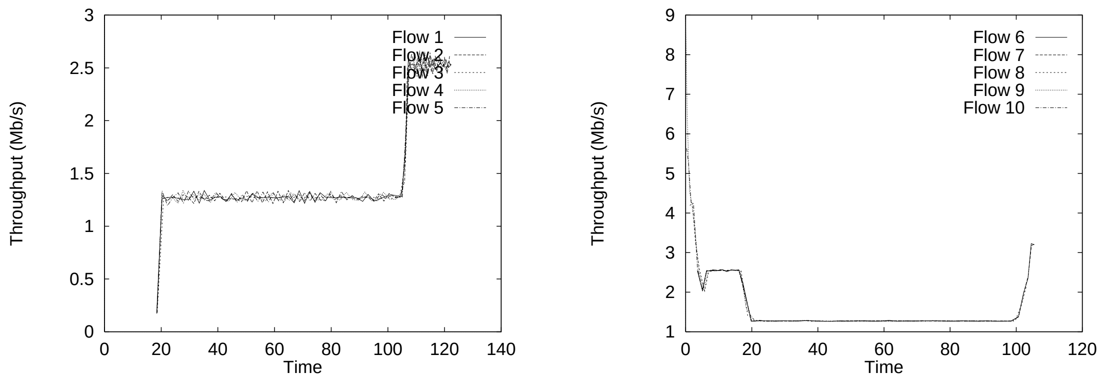


Figure 7.11: Throughput of flows in: (a) Revati class and (b) Hasta class

the throughputs of the flows in Revati and Hasta classes, respectively. The aggregate throughput of the flows in this case is 12.74 Mb/s. Hence, when the number of flows increased from 4 to 10, the overhead increased from 2.35% to 3.19%.

An important point to observe from Figures 7.9, 7.10, and 7.11 is that whereas in case of the unmodified and FCFS router the flows received different throughputs, in case of HSFQ all the flows always received equal throughput. This demonstrates that a fair algorithm such as SFQ is desirable for achieving fair allocation of bandwidth even when all flows employ the same congestion control algorithm.

- *Hierarchical link bandwidth allocation*: Observe from Figure 7.10 that when flow 1 terminated, the throughput of flow 2 increased by a factor of 2. The throughputs of flows 3

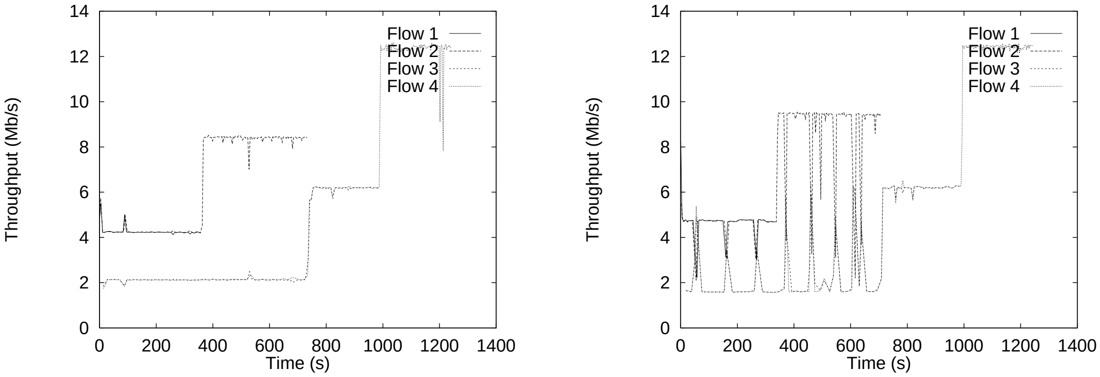


Figure 7.12: Throughput of flows when weights of Revati and Hasta are in the ratio: (a) 2:1 and (b) 3:1

and 4 that belong to different class, on the other hand, remained uneffected. This demonstrates that HSFQ achieves hierarchical allocation of link bandwidth.

To further demonstrate hierarchical allocation of link bandwidth, we conducted three additional experiments:

- We assigned weights to Revati and Hasta classes in the ratio 2:1. Figure 7.12(a) plots the throughput of the flows. As it illustrates, the aggregate throughput of flows 1 and 2 in Revati class was twice the throughput of flows 3 and 4 in Hasta class. Also, when flow 1 terminated, only the throughput of flow 2 increased.
- We assigned weights to Revati and Hasta classes in the ratio 3:1. Figure 7.12(b) plots the throughputs of the flows. It demonstrates that flows in Revati class received 3 times the bandwidth of the flows in Hasta class.
- We assigned weights to Revati and Hasta classes in the ratio 3:1 and changed the behaviour of flow 1. Flow 1 sent 50,000 4KB packets and became inactive for 60 seconds. On becoming active again, it transmitted 50,000 4KB packets. Figure 7.13 plots the throughputs of flows 1 and 2. It demonstrates that when flow 1 became inactive, the throughput of flow 2 increased by a factor of 2. On the other hand, when it became active again, the throughput of flow 2 reduced by a factor of 2.

• *Coexistence of heterogeneous congestion control techniques*: To demonstrate that Hier-

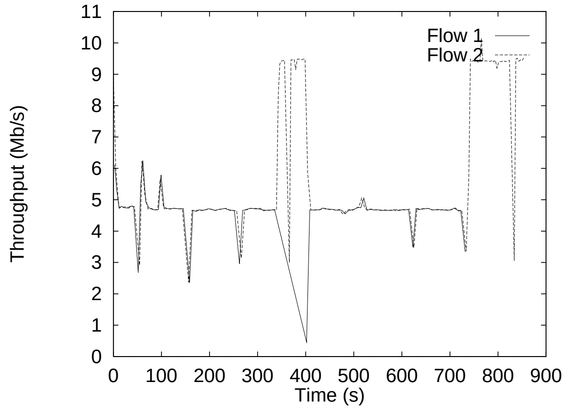
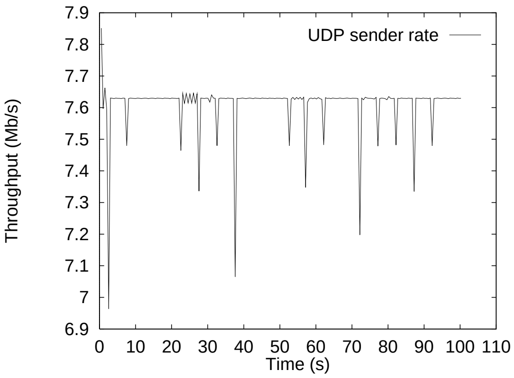


Figure 7.13: Throughput of flows 1 and 2 when flow 1 becomes active after being inactive for60 seconds

archical SFQ enables coexistence of different congestion control techniques, we initiated a TCP flow and a UDP flow. TCP and UDP flows belonged to Revati and Hasta classes, respectively. UDP flow source sent UDP packets at 7.6 Mb/s (see Figure 7.14(a)). Since both classes were assigned equal weights and the aggregate throughput of the router with HSFQ module was 12.8Mb/s, UDP flow was sending at rate higher than its fair share. However, as Figure 7.14(b) demonstrates, even though UDP flow sent at a rate higher than its fair share, TCP flow remained uneffected and received its fair share. This demonstrates that Hierarchical SFQ enables coexistence of heterogeneous congestion control techniques.

We repeated this experiment with a FCFS module. Figure 7.15 plots the throughputs of TCP and UDP flows as measured at Shravan. It demonstrates that when FCFS scheduling algorithm is employed, UDP flow received more than its fair share of the link bandwidth. This experiment demonstrates the need for a scheduling algorithm such as Hierarchical SFQ that provides protection between classes of flows employing different congestion control techniques.

To summarize, our experiments demonstrate that Hierarchical SFQ achieves hierarchical allocation of link bandwidth and enables coexistence of heterogeneous control techniques. Furthermore, its untuned implementation introduces 2-3% overhead.



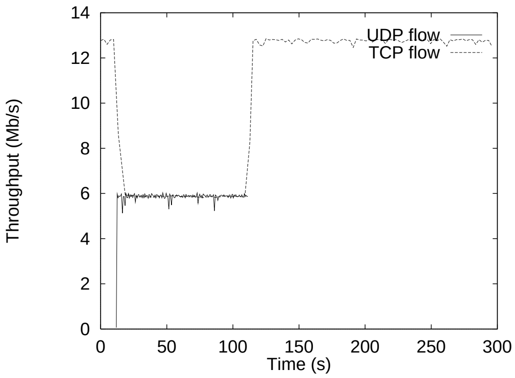


Figure 7.14: Throughput of: (a) UDP source and (b) TCP and UDP flows

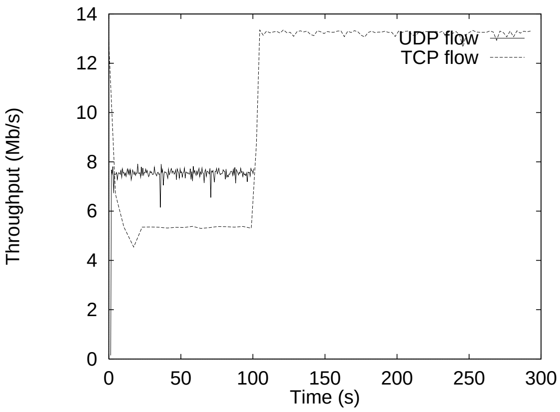


Figure 7.15: Throughput of TCP and UDP flows with FCFS module

7.6 Conclusions

In this chapter, we presented hierarchical fair schedulers for achieving hierarchical link bandwidth allocation. We demonstrated that the fair scheduling algorithm that is employed by a hierarchical scheduler must achieve fairness over variable rate servers. We demonstrated that SFQ and Fair Airport algorithms meet this fundamental requirement. Thus, Hierarchical SFQ and FA achieve the objectives of hierarchical link bandwidth allocation. We derived the performance bounds for each class in the tree for Hierarchical SFQ and FA schedulers. We demonstrated that if the root class can be modeled as a Fluctuation Constrained (FC) or Exponentially Bounded Fluctuation (EBF) server, then each of the subclasses can be modeled as a FC or EBF server, respectively. Using this analysis method, if the performance guarantees of the scheduler employed by a leaf class are known for FC/EBF servers, then the performance bounds for a flow can be determined even when it is hierarchically scheduled. We derived the performance guarantees of a flow when the leaf scheduler is either SFQ, FA, Virtual Clock, or Delay EDD. Our analysis method is general and leads to tighter results. Finally, we presented an implementation of Hierarchical SFQ. Our experiments demonstrated that Hierarchical SFQ achieves hierarchical allocation of link bandwidth and enables coexistence of heterogeneous congestion control techniques.

Chapter 8

Evaluation of SFQ and FA

In this chapter, we evaluate Start-time Fair Queuing (SFQ) and the class of Fair Airport (FA) algorithms that we have presented in the previous chapters. We first determine the tradeoffs between SFQ and FA. Our analysis demonstrates that SFQ has lower implementation complexity and better fairness properties. FA, on the other hand, has better isolation of delay guarantee. Since SFQ has better fairness properties, we then evaluate its properties from a theoretical perspective. Our analysis demonstrates that fairness guarantee of SFQ is close to the best achievable guarantee. Furthermore, the delay properties of SFQ are similar to an online scheduler that minimizes unfairness. Finally, we compare the various fairness definitions presented in the literature and demonstrate that the fairness definition chosen by us to evaluate SFQ and FA is appropriate.

The rest of the chapter is structured as follows. The tradeoffs between SFQ and FA are presented in Section 8.1. We evaluate the fairness and deadline guarantees of SFQ in Section 8.2. We justify the appropriateness of our fairness definition in Section 8.3. Finally, Section 8.4 presents the conclusions of this chapter.

8.1 SFQ and FA Tradeoffs

To evaluate the tradeoffs between SFQ and FA, consider their implementation complexity and fairness and deadline guaranties:

- *Implementation complexity*: The per packet complexity of both SFQ and FA is $O(logQ)$. However, since FA employs SFQ it performs more computations than SFQ. Hence, the implementation complexity of FA is higher than that of SFQ.

- *Fairness guarantee*: The differences in the fairness guarantee of SFQ and FA are:

  - The unfairness measure of FA is higher than that of SFQ. Specifically, Theorem 5.1 demonstrates that in a SFQ server for all intervals $[t_1, t_2]$ in which flows f and m are continuously backlogged:

    $$\left| \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \right| \leq \frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}$$

    Thus, unfairness measure of SFQ, i.e., $H(f, m)$ value, is

    $$\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m} \tag{8.1}$$

    On the other hand, from Theorem 6.1 we know that a FA server guarantees that

    $$\left| \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \right| \leq 2\left(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}\right) + (\beta_f^{max} + \beta_n^{max}) + 2\epsilon$$

    Thus, unfairness measure of FA is

    $$2\left(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}\right) + (\beta_f^{max} + \beta_m^{max}) + 2\epsilon \tag{8.2}$$

    Clearly, (8.2) is at least factor of 2 more than (8.1). If Virtual Clock is employed as the GSQ scheduling algorithm, then $\beta_f^{max} \geq \frac{l_f^{max}}{r_f}$ and $\beta_m^{max} \geq \frac{l_m^{max}}{r_m}$. In such a case, (8.2) is 3 times (8.1).

  - Whereas SFQ is fair regardless of variation in server capacity, FA is fair only if a server is a Fluctuation Constrained variable rate server.

  - Whereas FA requires admission control to ensure fairness, SFQ does not require it.

Table 8.1 summarizes these tradeoffs. The requirements of admission control as well as the server being fluctuation constrained in case of FA, may restrict its applicability in some scenarios. To observe this, consider a server which gives priority to real-time

Algorithm	Unfairness measure	Server requirements	Admission control
SFQ	$\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}$	None	Not required
FA	$2(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}) + (\beta_f^{max} + \beta_m^{max}) + 2\epsilon$	Fluctuation Constrained	Required

Table 8.1: Fairness property of SFQ and FA

flows and allocates the residual bandwidth fairly among best-effort flow controlled flows or hierarchically allocates it among best-effort flows belonging to various administrative domains (this is the scheduling architecture articulated in [49]). It may not be possible for such a server to employ FA for achieving these objectives since: (1) it may not be possible to model the residual bandwidth as being fluctuation constrained, and (2) it may not be possible to perform admission control for best effort flow-controlled flows.

- *Deadline guarantees*: The deadline guarantee of FA depends on the GSQ scheduling algorithm. Hence, we compare the deadline guarantees of SFQ with FA assuming either Virtual Clock or Delay EDD is employed as the GSQ scheduler. The tradeoffs in the deadline properties are:
  - SFQ and FA with Delay EDD provide low delay to low throughput flows without increasing rate reservation. However, since FA with Virtual Clock allocates only rate, it can provide low delay to low throughput flows only by increasing their rate reservation.
  - The admission control complexity of SFQ and FA with Virtual Clock is $O(1)$. The complexity of FA with Delay EDD, in contrast, is $O(Q)$.
  - The deadline guarantees in SFQ depend on the packet lengths of the other flows at a server. In contrast, the deadline guarantees of both FA with Virtual Clock or FA

Algorithm	Low delay to low throughput flows	Admission control complexity	Delay independent of other flows
SFQ	Yes	$O(1)$	No
FA with Virtual Clock	No	$O(1)$	Yes
FA with Delay EDD	Yes	$O(Q)$	Yes

Table 8.2: Summary of delay properties of SFQ and FA

with Delay EDD are independent of the other flows at a server.

Table 8.2 summarizes these tradeoffs.

From the above discussion, we conclude that SFQ has lower implementation complexity and better fairness properties. On the other hand, FA has better isolation of delay guarantee. However, this isolation is at the cost of either increased admission control complexity or high delay to low throughput flows.

Since SFQ has better fairness properties but not isolation of deadline guarantee, there are two questions that arise: (1) how good is the fairness guarantee of SFQ? and (2) is there an inherent tradeoff between fairness and isolation of deadline guarantee? To answer these questions, we further evaluate the fairness and deadline guarantees of SFQ in the next section.

8.2 Fairness and Deadline Guarantees of SFQ

In this section, we first evaluate the fairness guarantee of SFQ and then consider its deadline guarantee.

8.2.1 Fairness Guarantee

The unfairness measure $H(f, m)$ of SFQ was demonstrated in Theorem 5.1 to be $\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}$. Since the lower bound on unfairness measure achievable by any algorithm was shown to be $\frac{1}{2}(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m})$ in [22], it seems that the fairness guarantee of SFQ is factor of two higher than the best possible guarantee. We derive a tighter lower bound on unfairness measure and demonstrate that fairness guarantee of SFQ is better than what the current lower bound indicates.

To derive a tighter lower bound on $H(f, m)$, consider two flows f and m. Let $\alpha = \frac{l_f^{max}}{r_f}$, $\beta = \frac{l_m^{max}}{r_m}$ and $\alpha \leq \beta$. Let c be a positive integer such that $c\alpha < \beta \leq (c+1)\alpha$. Theorem 8.1 derives a tighter lower bound on $H(f, m)$.

Theorem 8.1 *If an algorithm guarantees that* $\mid \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \mid \leq H(f, m)$ *for all intervals* $[t_1, t_2]$ *in which flows* f *and* m *are continuously backlogged, then* $H(f, m) \geq L(f, m)$ *where* $L(f, m) = (c+1)\alpha$

Proof: Let unfairness in interval $[t_1, t_2]$ refer to $\mid \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \mid$. There are two cases to consider:

- $(c+1)\alpha = \beta$: Unfairness in the interval in which a packet of length l_m^{max} of flow m is served is $\frac{l_m^{max}}{r_m} = \beta$. Hence, the theorem holds for this case.

- $c\alpha < \beta < (c+1)\alpha$: Consider the schedule generated when flow f and m are continuously backlogged and send all packets of length l_f^{max} and l_m^{max}, respectively. Then, there are two cases to consider:

  - More than one flow m packets are scheduled back to back: Since no flow f packets are served in the interval in which these packets are served, the unfairness is at least $2\frac{l_m^{max}}{r_m} = 2\beta \geq (c+1)\alpha$.
  - No more than one flow m packet is scheduled back to back: There are three further cases to consider:
    - ∗ More than c, i.e., at least $c+1$, flow f packets are scheduled back to back: Since no flow m packets are served in the interval in which these $c+1$ packets are served, the unfairness is at least $(c+1)\frac{l_f^{max}}{r_f} = (c+1)\alpha$.

* Less than c, i.e., at most $c-1$, flow f packets are scheduled back to back: Then a part of the schedule is:

$$..., p_m^k, p_f^n, ..., p_f^{n+o}, p_m^{k+1}$$

where $o \leq c-1$. Then the unfairness in the interval in which packets p_m^k and p_m^{k+1} are served, denoted by U, is given as:

$$\begin{aligned} U &= 2\frac{l_m^{max}}{r_m} - o\frac{l_f^{max}}{r_f} \\ &\geq 2\frac{l_m^{max}}{r_m} - (c-1)\frac{l_f^{max}}{r_f} \\ &\geq 2c\alpha - (c-1)\alpha \\ &\geq (c+1)\alpha \end{aligned}$$

* Always one flow m and c flow f packets are scheduled back to back: Consider the following schedule:

$$..., p_m^1, p_f^n, ..., p_f^{n+c}, p_m^2, ..., p_m^N$$

The unfairness in the interval in which packets p_m^1 and p_m^N are served, denoted by U, is given as:

$$\begin{aligned} U &= N\frac{l_m^{max}}{r_m} - (N-1)c\frac{l_f^{max}}{r_f} \\ &= N\beta - (N-1)c\alpha \\ &= N(\beta - c\alpha) + c\alpha \end{aligned}$$

Hence, $U \geq (c+1)\alpha$ if:

$$\begin{aligned} N(\beta - c\alpha) + c\alpha &\geq (c+1)\alpha \\ N &\geq \frac{\alpha}{\beta - c\alpha} \end{aligned}$$

Since flow f and m send infinite number of packets, packet p_m^N exists. Hence, the unfairness in this case also is at least $(c+1)\alpha$.

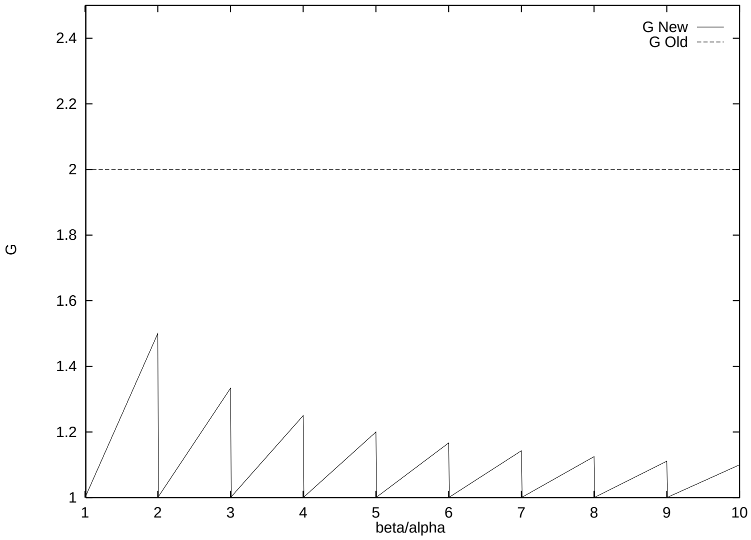


Figure 8.1: Ratio of fairness guarantee of SFQ and lower bound.

■

Define G as the ratio of fairness guarantee of SFQ and the lower bound on fairness guarantee. Then using old lower bound, we get $G = 2$. On the other hand, using the lower bound derived in Theorem 8.1, we get:

$$G = \frac{1 + \frac{\beta}{\alpha}}{1 + c}$$

Figure 8.1 plots the metric G for different values of $\frac{\beta}{\alpha}$. As it illustrates, G computed using the new lower bound is smaller. Furthermore, on an average, fairness guarantee of SFQ is within 11% of the lower bound.

8.2.2 Deadline Guarantee

Recall that whereas delay guarantee of WFQ for a flow depends only on the flow's properties, delay guarantee of SFQ depends on the maximum packet length of all the flows at the server. Hence, WFQ has better isolation of delay guarantee of a flow than SFQ. On the other hand, from (4.2), we know that fairness guarantee of WFQ is worse than that of SFQ. Similarly, as we demonstrated in Section 8.1, FA algorithms that have the delay guarantee of WFQ have worse fairness guarantee than SFQ. From this it appears that there is a tradeoff between fairness guarantee and isolation of delay guarantee of a flow. Hence, a natural question that arises is: is it possible to design an algorithm that achieves the best possible fairness and has the same deadline guarantee as WFQ? The following example illustrates that such an *online* scheduler does not exist. An online scheduler is one that does not use the length of packet p_f^{k+1} in making a scheduling decision for packet p_f^k.

Example 8.1 *Let unfairness in interval $[t_1, t_2]$ in which flows f and m are continuously backlogged, refer to $\left| \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \right|$. An algorithm is said to minimize unfairness if it minimizes the maximum value of unfairness for all the combinations of flows over all intervals. Let an online scheduler that minimizes unfairness serve N flows. Let each flow have weight r such that $Nr = C$ where C is the capacity of the server. Let all the flows become backlogged simultaneously and send infinite number of packets. Furthermore, for flows 1 through $N-1$ let the length of the second packet be $(2-\epsilon)l$, $0 < \epsilon < 1$, and the length of all the other packets be l. For flow N, let the length of all packets be l. Then the schedule that has the lowest unfairness is:*

$$p_1^1, p_2^1, ..., p_N^1, p_1^2, p_2^2, ..., p_{N-1}^2, p_N^2, p_1^3, ..., p_N^3, ... \tag{8.3}$$

To observe this, note that this schedule for any two flows has a maximum unfairness of $\frac{(2-\epsilon)l}{r}$. Since the length of packets $p_1^2, p_2^2, ..., p_{N-1}^2$ is $(2-\epsilon)l$, this is the minimum achievable unfairness. Furthermore, if packet p_N^2 were to be served before packet p_k^2 ($k < N$), then the unfairness between flows k and N in the interval in which packets p_N^1 and p_N^2 are served is $\frac{2l}{r}$.

In this schedule, the departure time of packet p_N^2 is greater than that in WFQ by Δ which is given as:

$$\Delta = \frac{(N-1)(2-\epsilon)l}{C} + \frac{l}{C} + \frac{Nl}{C} - \frac{2Nl}{C}$$

Clearly, Δ is greater than 0 and depends on the maximum packet length of all the other flows.

The schedule obtained by interchanging all flow 1 packets with flow N packets in (8.3), would have the same unfairness as the schedule in (8.3) but would have the same delay guarantee as WFQ. However, to generate such a schedule, the length of second packet of flow N packet would have to be used to make a scheduling decision for flow N's first packet. Consequently, such a schedule can not be generated by an online scheduler.

Hence, we conclude that an online scheduler that minimizes unfairness does not have the deadline guarantee of WFQ. Furthermore, its deadline guarantee for a flow depends on the maximum packet length of all the other flows.

From the above example, we conclude that isolation of delay guarantee of a flow in SFQ is similar to that in an online algorithm that minimizes unfairness. It is important to note that the above example does not demonstrate the impossibility of designing a scheduling algorithm that has the fairness guarantee of SFQ but the deadline guarantee of WFQ; it only demonstrates a tradeoff between fairness and isolation.

8.3 Fairness Definitions

We have chosen the definition of fairness, termed *relative fairness*, that was introduced in [22] and consider an algorithm fair if the difference in service normalized by weight received by two flows when they are backlogged is bounded, i.e., $\mid \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \mid$ is bounded.

Another definition of fairness termed Generalized Processor Sharing (GPS) Fairness has been introduced in the literature. A packet scheduling algorithm is considered fair if it "emulates" a hypothetical bit-by-bit round robbin server in which the number of bits of a backlogged flow served in a round is proportional to the weight of a flow. The hypothetical server is referred to as a GPS server. In this definition, the meaning of the term "emulates" needs to be defined. A scheduling algorithm can be said to an emulation of the GPS server if:

1. The departure time of a packet in the scheduling algorithm is close to its departure time in the GPS server, i.e.,

$$L_{GPS}(p_f^j) - \delta_1 \leq L(p_f^j) \leq L_{GPS}(p_f^j) + \delta_2$$

   where $L_{GPS}(p_f^j)$ and $L(p_f^j)$ are the departure time of a packet in the GPS server and the scheduling algorithm, respectively, and δ_1 and δ_2 are some constants. A scheduling algorithm that has smaller δ_1 and δ_2 is considered to be a better emulation.

2. The aggregate service received by a flow in any interval in the scheduling algorithm is close to that received by it in the GPS server, i.e., for all intervals $[t_1, t_2]$,

$$| W_f(t_1, t_2) - W_f^{GPS}(t_1, t_2) | \leq \delta$$

where $W_f(t_1, t_2)$ and $W_f^{GPS}(t_1, t_2)$ are the service received by a flow in the interval $[t_1, t_2]$ in the scheduling algorithm and the GPS server, respectively, and δ is a constant.A scheduling algorithm that has smaller δ is considered to be a better emulation.

3. It ensures relative fairness. Since the GPS server could also have been defined as one in which $| \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} | = 0$, this is an appropriate definition of emulation.

4. It is worst-case fair. A scheduling algorithm is called worst-case fair for flow f, if for all time t, the delay of a packet arriving at time t is at most $\frac{Q_f(t)}{r_f} + \frac{c_f}{r_f}C$ where r_f is the throughput guaranteed to flow f, $Q_f(t)$ is the queue size of flow f at time t (including the length of the packet arriving at time t), C is the capacity of the server, and c_f is a constant, referred to as the *Normalized worst-case fairness index for flow* f, independent of the queues of the other flows at the server [4]. Define *Normalized worst-case fairness index for the algorithm*, denoted by c, as $c = \max_{n \in Q}\{c_n\}$ where Q is the set of flows served by a server. Then, an algorithm that has smaller c value is considered to be a better emulation of the GPS server.

   Though worst-case fairness captures an important property of the scheduling algorithms, it is an inappropriate definition for emulation of a GPS server. This is because an algorithm that does not emulate GPS server may have the smallest value of c. For example, consider a work conserving Rate Controlled Service Discipline that we had discussed in Chapter 6. Let the scheduling algorithm for GSQ and ASQ be Virtual Clock and FCFS, respectively. Then, it can be shown that if the rate controller is the same as one used in Fair Airport algorithm presented in Chapter 6, then this algorithm has the smallest value of c. However, since flows can receive arbitrary service from ASQ, it will not emulate a GPS server.

Since worst-case fairness is an inappropriate definition of emulation, we will not consider it further and focus only on the first three definitions.

Example 8.1 demonstrates that if an algorithm is a good emulation as per relative fairness, it may not be a good emulation as per the first definition (WFQ has the smallest possible value of δ_2 in the first definition). Since there are multiple definitions of emulation, at least two of which are incompatible, let us evaluate which of the definitions is more appropriate. There are two criteria that we may use for evaluation:

- Ability to achieve the objectives: There are two objectives of a fair scheduling algorithm: (1) it should allocate a server resources for flow-controlled data sources such that overall network resource allocation is max-min fair [32]; and (2) it should provide QoS guarantees to video sources even when persistent congestion occurs. There is no theoretical/experimental evidence that demonstrates the superiority of one emulation over the other in achieving these objectives.

- Efficiency of emulation: Observe that the first two definitions of emulation are defined with reference to the GPS server. Hence, it appears that to emulate the GPS server as per those definitions, a scheduling algorithm would have to simulate it. In fact, no scheduling algorithm that emulates as per those definitions but does not simulate a GPS server is known. As we had observed in Section 4.4, the simulation of a GPS server is computationally expensive for constant rate servers and becomes even more expensive/infeasible for variable rate servers. The relative fairness definition, in contrast, does not refer to a GPS server and hence enables efficient emulations that do not require a simulation. Thus, from the perspective of efficient emulation, the relative fairness definition is more desirable than the first two.

Since the first two definitions lead to inefficient algorithms and there is no theoretical/experimental evidence that indicates that they are more suitable for achieving the end objective, we conclude that relative fairness is an appropriate interpretation of GPS Fairness. Thus, the fairness definition chosen by us to evaluate SFQ and FA is appropriate.

8.4 Conclusions

In this chapter, we evaluated Start-time Fair Queuing and the class of Fair Airport (FA) algorithms. We first determined the tradeoffs between SFQ and FA. Our analysis demonstrated that SFQ has lower implementation complexity and better fairness properties. FA, on the other hand, has better isolation of delay guarantee. Since SFQ has better fairness properties, we then evaluated its properties from a theoretical perspective. Our analysis demonstrated that fairness guarantee of SFQ is close to the best achievable guarantee. In fact, on an average, it is within 11% of the lower bound. Furthermore, the delay properties of SFQ are similar to an online scheduler that minimizes unfairness. Finally, we compared the various fairness definitions presented in the literature and demonstrated that the fairness definition chosen by us to evaluate SFQ and FA is appropriate.

Chapter 9

Conclusions

The emergence of multimedia and real-time applications has led to the design of integrated services networks. An integrated services network supports applications with diverse QoS requirements. Packet scheduling is a fundamental mechanism employed by such a network to meet the QoS requirements of the applications. In this dissertation, we focused on the design, analysis, and implementation of packet scheduling algorithms for integrated services networks.

In this chapter we summarize the contributions of this dissertation and then present some avenues for future research.

9.1 Summary of Contributions

In this dissertation, we first developed a framework for the analysis of packet scheduling algorithms in heterogeneous networks and employed it to determine the desirable characteristics of packet scheduling algorithms. We then determined the requirements for suitable packet scheduling algorithms for integrated services networks and developed new algorithms to meet those requirements. Through analysis and experiments we demonstrated that these new scheduling algorithms are suitable for integrated services networks. In what follows, we present our contributions in more detail.

We first defined and analyzed a class of Guaranteed Rate (GR) scheduling algorithms. The scheduling algorithms in GR provide a service guarantee to a flow that is independent of a traffic specification and the behaviour of the other flows in a network. This enables a GR server to provide QoS guarantees to flows conforming to any specification. We demonstrated that the class of GR scheduling algorithms is broad and includes most of the algorithms proposed in the literature. To efficiently support flows with variable bit rate requirements, we defined several algorithms that allocate variable rates to the packets of a flow and demonstrated that they belong to the GR class. We also defined scheduling algorithms in GR that are suitable for servers where packet fragmentation occurs.

To enable a network to provide QoS guarantees similar to those provided by a single server, we developed a method for deriving the service guarantee of a network of GR servers. This method enables the derivation of service guarantee of a heterogeneous network in which packet fragmentation and/or reassembly may occur. We demonstrated that the end-to-end service guarantee is similar to a single server guarantee. This reduces the end-to-end QoS determination problem to a single server problem and enables a network to provide guarantees to flows with any traffic specification. We illustrated the end-to-end delay bound computation for flows conforming to leaky bucket and demonstrated that our method leads to tighter results. The key contributions of the GR framework are: (1) it decouples service guarantee provided by a network from source traffic characterization, and (2) it is the first framework that tightly analyzes heterogeneous networks. This framework is now being employed by other researchers to analyze new scheduling algorithms [53].

To gain a further understanding of the GR class, we evaluated the completeness and relative merits of algorithms in GR.

- *Completeness of GR*: Though GR does not contain all the algorithms, we experimentally demonstrated that it does not lead to loss in achievable utilization of a network. In fact, contrary to intuition, we demonstrated that algorithms that are not in GR lead to lower achievable utilization.

- *Relative merits of the algorithms in GR*: We evaluated the merits of algorithms classified along two dimensions:

  - *Fair vs. unfair*: To evaluate fair and unfair algorithms, we introduced the concept of delay-cum-throughput guarantee which refers to a deadline guarantee for a packet based on the departure time of the previous packet of the flow. We derived end-to-end delay-cum-throughput guarantee of an internetwork of servers that provide such a guarantee. We demonstrated that fair algorithms, in addition to the guarantees provided by unfair algorithms, provide this guarantee. This enables fair algorithms to provide better bounds on QoS to a flow when the network is uncongested. Hence, fair algorithms are desirable.

  – *Rate-only vs. rate-and-delay allocation algorithms*: We experimentally demonstrated that algorithms that separate rate and delay allocation achieve higher utilization and, hence, are desirable.

Thus, we conclude that GR is a useful class of algorithms. Furthermore, algorithms that are fair and/or achieve separation of rate and delay allocation are desirable.

We then examined the requirements for a suitable packet scheduling algorithm for integrated services network. Our analysis demonstrated that a suitable packet scheduling algorithm should: (1) achieve low average and maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) provide fairness for variable bit rate video; (3) provide fairness over variable rate servers for throughput-intensive, flow-controlled data applications; (4) provide QoS guarantees over variable rate servers; (5) support hierarchical link bandwidth allocation to enable a network to provide link-sharing service and facilitate coexistence of multiple services and protocols; and (6) be computationally efficient. To meet these requirements, we designed Start-time Fair Queuing (SFQ) and a class of Fair Airport (FA) algorithms, and employed them to develop Hierarchical SFQ and FA algorithms for hierarchically allocating link bandwidth. Specifically:

- *SFQ*: We demonstrated that SFQ is computationally efficient and allocates bandwidth fairly regardless of variation in a server rate. We demonstrated that its fairness guarantee is close to the optimal and is at least as good as that of all the known algorithms. We analyzed the throughput, single server deadline, and end-to-end deadline guarantee of SFQ. This analysis was carried out for variable rate servers that can be modeled as either

Fluctuation Constrained (FC) or Exponentially Bounded Fluctuation (EBF) servers.

- *FA algorithms*: We designed FA algorithms by combining SFQ with a class of non-work conserving algorithms. Though FA algorithms have higher implementation complexity than SFQ, we demonstrated that they are computationally efficient. We derived fairness and deadline guarantees for FC FA servers. We demonstrated that appropriate choice of a non-work conserving algorithm leads to the design of first fair algorithm that achieves separation of rate and delay allocation. We also illustrated that FA class contains algorithms with different implementation complexities and performance guarantees and, thus, enables tradeoffs between the two. Finally, we demonstrated that it can be generalized to achieve several other objectives.

SFQ and FA belong to the GR class and hence have all the desirable properties of the GR algorithms. Our analysis and experiments demonstrated that SFQ and FA algorithms meet all the requirements for a suitable packet scheduling algorithm.

To allocate link bandwidth hierarchically, we employed SFQ and FA and developed *Hierarchical SFQ* and *FA* scheduling algorithms. Hierarchical SFQ and FA enable a network to provide a link-sharing service and manage heterogeneity in services and protocols. These algorithms allow different scheduling algorithms to be employed for the leaf classes of a hierarchy. We developed a technique for analyzing the performance guarantees of Hierarchical SFQ and FA. Using this technique the QoS guarantees for a flow can be determined if the performance guarantees of a leaf class scheduling algorithm are known for FC/EBF servers. We derived the performance guarantees for a flow when the leaf class scheduling algorithm is either SFQ, FA, Virtual Clock, or Delay EDD. Our analysis technique is general and applicable for other hierarchical schedulers constructed using algorithms that provide guarantees similar to SFQ and FA. Furthermore, this analysis technique is the first in the literature that yields tight performance bounds for hierarchical scheduling algorithms. Finally, we presented an implementation of Hierarchical SFQ in an IPv6 router.

The key contributions of our work on SFQ and FA include: (1) illustration of the need for fairness over variable rate servers and design of algorithms that meet this requirement; (2)

design of hierarchical scheduling algorithms that provide quantitative performance guarantees; (3) the single server and end-to-end analysis of SFQ and FA for variable rate FC and EBF servers; and (4) analysis technique for hierarchical scheduling algorithms that yields tight performance guarantees.

9.2 Directions for Future Research

There are several directions in which further research on packet scheduling algorithms can be pursued. Some of the directions that we believe are important are:

- *Statistical analysis techniques*: The analysis techniques that we and other researchers have developed analyze the guarantees of a packet scheduling algorithm when the aggregate rate reserved by the flows is always less than the capacity of the server. However, to achieve efficient utilization of resources when the flows have variable bit rate requirement, a network may have to overbook the capacity of the server, i.e., the aggregate reserved rate may exceed the capacity of the server for transient durations. Hence, to enable a network to achieve efficient utilization of resources, it is desirable to analyze the guarantees of packet scheduling algorithms when a server capacity may be overbooked.

- *Complexity analysis of fair algorithms*: The computational complexity of fair algorithms that we have designed is $O(logQ)$ per packet where Q is the number of flows at a server. It seems that this complexity can be reduced only by worsening the fairness guarantee of an algorithm. For example, Deficit Round Robin has $O(1)$ complexity per packet but significantly worse fairness guarantee. A proof that there is an inherent tradeoff between fairness guarantee and the computational complexity of an algorithm is desirable. It is our conjecture that the problem of achieving fairness is closely related to that of sorting and, hence, has a tight lower bound of $O(logQ)$ [20].

- *Optimal scheduling algorithm for providing heterogeneous services*: The hierarchical scheduling algorithms that we have developed for providing heterogeneous services do not exploit the semantics of the services. Though this approach has the advantage that different services can be designed independently, it may lead to sub-optimal performance. To illustrate, consider a server that supports best-effort and real-time services. The objectives of the best-effort and real-time services are to minimize the average delay of the packets and meet the deadlines of the packets, respectively. Now, consider a scenario in which 50% of the bandwidth is allocated to each of the service classes and the server capacity is $4pkt/s$. Let 1 and 3 packets arrive in the real-time and best-effort service classes at time 0, respectively. Let the deadline of the packet in the real-time class be $4s$. In such a case, the hierarchical schedulers that we have developed may first schedule the real-time packet and then the 3 best-effort packets. However, an optimal scheduler that exploits the semantics of the services will schedule the 3 best-effort packets before the real-time packet thereby minimizing the average response time of best-effort packets while meeting the deadline of the real-time packet. Thus, scheduling algorithms that exploit the semantics of services can achieve better performance. To the best of our knowledge, the design of scheduling algorithms that exploit semantics of services to achieve optimal performance is an open problem. It would also be fruitful to evaluate the tradeoffs between the complexity and performance of hierarchical scheduling algorithms and algorithms that exploit semantics.

Bibliography

[1] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line Scheduling in the Presence of Overload. In *Proceeding of Foundations of Computer Science*, pages 100–110, 1991.

[2] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. Fair On-Line Scheduling of a Dynamic Set of Tasks on a SingleResource. Technical Report TR-96-03, Department of Computer Sciences, The University of Texas at Austin, February 1996.

[3] J.C.R. Bennett and H. Zhang. Hierarchical Packet Fair Queuing Algorithms. In *Proceedings of SIGCOMM'96*, pages 143–156, August 1996.

[4] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case Fair Weighted Fair Queuing. In *Proceedings of INFOCOM'96*, pages 120–127, March 1996.

[5] R. Brown. Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem. *Communications of the ACM*, 31(10):1220–1227, October 1988.

[6] D.D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network. In *Proceedings of ACM SIGCOMM*, pages 14–26, August 1992.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge MA, 1990.

[8] R.L. Cruz. Service Burstiness and Dynamic Burstiness Measures: A Framework. *Journal of High Speed Networks*, 2:105–127, 1992.

[9] R.L. Cruz. A Calculus for Network Delay, Part I : Network Elements in Isolation. *IEEE Transactions on Information Theory*, 37:114–131, Jan 1991.

[10] R.L. Cruz. A Calculus for Network Delay, Part II : Network Analysis. *IEEE Transactions on Information Theory*, 37:132–141, Jan 1991.

[11] J. Davin and A. Heybey. A Simulation Study of Fair Queueing and Policy Enforcement. *Computer Communication Review*, 20(5):23–29, October 1990.

[12] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, September 1989.

[13] D. Ferrari. Client Requirements for Real-Time Communication Services. *IEEE Communications Magazine*, pages 65–72, November 1990.

[14] D. Ferrari and D. C. Verma. A Scheme for Real-Time Channel Establishment in Wide-Area Networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[15] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[16] D. Le Gall. MPEG: A Video Compression Standard for Multimedia Apllications. *Communications of the ACM*, 34(4):46–58, April 1991.

[17] L. Georgadis, R. Guerin, and A. Parekh. Optimal Multiplexing on a Single Link: Delay and Buffer Requirements. In *Proceeding of INFOCOM'94*, 1994.

[18] L. Georgiadis, R. Guerin, V. Peris, and R. Rajan. Efficient Support of Delay and Rate Guarantees in an Internet. In *Proceedings of ACM SIGCOMM'96*, pages 106–116, August 1996.

[19] L. Georgiadis, R. Guerin, V. Peris, and K.N. Sivarajan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. In *Proceedings of INFOCOM'96*, pages 102–110, March 1996.

[20] J. Golestani. Personal Communication. March 1997.

[21] S.J. Golestani. A Framing Strategy for Congestion Management. *IEEE Journal on Selected Areas in Communications*, pages 1064–1077, September 1991.

[22] S.J. Golestani. A Self-Clocked Fair Queueing Scheme for High Speed Applications. In *Proceedings of INFOCOM'94*, pages 636–646, April 1994.

[23] P. Goyal, S. S. Lam, and H. M. Vin. Determining End-to-End Delay Bounds In Heterogeneous Networks. *ACM/Springer-Verlag Multimedia Systems Journal*, 5(3):157–163, May 1997. Also appeared in the Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video, Pages 287-298, April 1995.

[24] P. Goyal and H. M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. In *IEEE/ACM Transactions on Networking (to appear)*. Also available as technical report TR95-30, Department of Computer Sciences, The University of Texas at Austin.

[25] P. Goyal and H. M. Vin. Network Algorithms and Protocol for Multimedia Servers. In *Proceedings of INFOCOM'96*, pages 1371–1379, March 1996.

[26] P. Goyal, H. M. Vin, C. Shen, and P.J. Shenoy. A Reliable, Adaptive Protocol for Video Transport. In *Proceedings of INFOCOM'96*, pages 1080–1090, March 1996.

[27] A. Greenberg and N. Madras. How Fair is Fair Queuing. *The Journal of ACM*, 39(3):568–598, July 1992.

[28] M. Grossglauser, S. Keshav, and D. Tse. RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic. In *Proceedings of ACM SIGCOMM'95*, pages 219–230, August 1995.

[29] J. D. Guyton and M. F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *Proceedings of ACM SIGCOMM'95*, pages 288–298, August 1995.

[30] C.R. Kalmanek, H. Kanakia, and S. Keshav. Rate Controlled Servers for Very High-Speed Networks. In *Proceedings of IEEE GLOBECOM'90, San Diego, CA*, pages 300.3.1–300.3.9, December 1990.

[31] D. D. Kandlur, K. G. Shin, and D. Ferrari. Real-Time Communication in Multihop Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1044–1056, October 1994.

[32] S. Keshav. *Congestion Control for Computer Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1991.

[33] S. Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of ACM SIGCOMM'91*, pages 3–15, 1991.

[34] S. Keshav. *An Engineering Approach to Computer Networking*. Addison Wesley, 1997.

[35] E. Knightly and H. Zhang. Traffic Characterization and Switch Utilization Using a Deterministic Bounding Interval Dependent Traffic Model. In *Proceedings of INFOCOM'95, Boston, MA*, April 1995.

[36] S.S. Lam and G.G. Xie. Burst Scheduling: Architecture and Algorithm for Switching Packet Video. In *Proceedings of INFOCOM'95*, April 1995.

[37] K. Lee. Performance Bounds in Communication Networks With Variable-Rate Links. In *Proceedings of ACM SIGCOMM'95*, pages 126–136, 1995.

[38] Q. Ma, P. Steenkiste, and H. Zhang. Routing High-bandwidth Traffic in Max-min Fair Networks. In *Proceedings of SIGCOMM'96*, pages 206–217, August 1996.

[39] E. M. McCreight. Priority Search Trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[40] A.K. Mok, P. Amarsinghe, M. Chen, S. Sutanthavibul, and K. Tanstisirivat. Synthesis of a Real-Time Message Processing System with Data-driven Timing Constraints. In *IEEE 7th Real-Time Systems Symposium*, pages 133–143, 1987.

[41] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. *IEEE/ACM Transactions On Networking*, 2(2):137–150, April 1994.

[42] A.K. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[43] S. A. Rago. *Unix System V Network Programming*. Addison-Wesley, 1993.

[44] S. Shenker. A Theoretical Analysis of Feedback Flow Control. In *Proceedings of ACM SIGCOMM'90*, pages 156–165, 1990.

[45] S. Shenker. Making Greed Work in Networks: A Game-Theoretic Analysis of Switch Service Disciplines. In *Proceedings of ACM SIGCOMM'94*, pages 47–57, 1994.

[46] S. Shenker. Contribution to the Int-serv mailing list. July 1995.

[47] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal of Selected Areas in Communications*, 13:1176–1188, September 1995.

[48] S. Shenker and C. Partridge. Specification of Guaranteed Quality of Service. Available via anonymous ftp from ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-intserv-guaranteed-svc-03.txt, November 1995.

[49] S. Shenker, L. Zhang, and D. Clark. A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Networks. Available via anonymous ftp from ftp://ftp.parc.xerox.com/pub/archfin.ps, 1995.

[50] M. Shreedhar and G. Varghese. Efficient Fair Queuing Using Deficit Round Robin. In *Proceedings of ACM SIGCOMM'95*, pages 231–242, 1995.

[51] D. Stiliadis. *Traffic Scheduling in Packet-Switched Networks: Analysis, Design and Implementation*. PhD thesis, Department of Computer Science and Engineering, University of California at Santa Cruz, 1996.

[52] D. Stiliadis and A. Varma. Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet Switched Networks. In *Proceedings of SIGMETRICS'96*, May 1996.

[53] S. Suri, G. Varghese, and G. Chandramenon. Leap Forward Virtual Clock: A New Fair Queuing Scheme with Guaranteed Delays and Throughput fairness. In *Proceedings of INFOCOM'97*, April 1997.

[54] J. Wroclawski. Specification of Controlled-load Network Element Service. Available via anonymous ftp from ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-intserv-ctrl-load-svc-09.txt,, June 1996.

[55] G.G. Xie and S.S. Lam. Delay Guarantee of Virtual Clock Server. *IEEE/ACM Transactions on Networking*, 3(6):683–689, December 1995.

[56] O. Yaron and M. Sidi. Performance and Stability of Communication Networks via Robust Exponential Bounds. In *IEEE/ACM Transactions on Networking*, volume 1, pages 372–385, 1993.

[57] O. Yaron and M. Sidi. Generalized Processor Sharing Networks with Exponentially Bounded Burstiness Arrivals. In *Proceedings of INFOCOM'94*, 1994.

[58] H. Zhang. Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10), October 1995.

[59] H. Zhang and D. Ferrari. Rate Controlled Static Priority Queueing. In *Proceedings of INFOCOM'93*, volume 2, pages 227–236, 1993.

[60] H. Zhang and D. Ferrari. Rate-controlled Service Disciplines. *Journal of High Speed Networks*, 3(4):389–412, 1994.

[61] H. Zhang and S. Keshav. Comparison of Rate-Based Service Disciplines. In *Proceedings of ACM SIGCOMM*, pages 113–121, August 1991.

[62] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proceedings of ACM SIGCOMM'90*, pages 19–29, August 1990.

[63] L. Zhang, S.E. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 7(5):8–18, September 1993.

[64] Q. Zheng and K. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-switching Networks. *IEEE Transactions on Communications*, 42(3):1096–1105, March 1994.

Vita

Pawan Goyal was born on December 22 1970. He is the son of Kamla Goyal and Om Prakash Goyal. He joined the Indian Institute of Technology, Kanpur, India in 1988 and received the degree of Bachelor of Technology in Computer Science and Engineering from there in May 1992. He joined the University of Texas at Austin for graduate studies in August 1992. He received the degree of Master of Science in Computer Sciences from there in May 1994.

Permanent Address: B 50, Sector 14
NOIDA, UP
India

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.