ABSTRACT

DWEKAT, ZYAD A. Practical Fair Queuing Schedulers: Simplification through Quantization. (Under the direction of Dr. George Rouskas and Dr. Mladen Vouk).

Many packet scheduling schemes have been proposed, but most of them suffer from one of two extremes. A scheduling scheme may have simple implementation with low fairness qualities, or it may have good fairness qualities but high complexity. Our goal in this research is to design a frame work of schedulers that has both the desired qualities of fairness and simple implementation. We began our research by conducting a survey of the scheduling techniques and associated analysis models proposed during the last decade. Then, in the first part of this thesis we present a suite of packet fair queuing schedulers with low complexity and good fairness and delay properties. Our designs employ the concept of quantization by exploiting two widely-observed characteristics of the Internet, namely that service providers offer some type of tiered service with a small number of service levels, and that a small number of packet sizes dominate. Taken together, these two observations permit us to design a good fair queuing algorithm in a manner that packet sorting operations only need to consider a small, fixed number of packets, independent of the number of flows, and hence can be performed in constant time. Specifically, the scheduler we present is equivalent to WF^{2}Q, with the additional advantage that the virtual time function can be computed in $O(1)$ time. Our tiered-service schedulers operate under assumptions that are valid under a wide range of practical scenarios, and combine provable good performance with amenability to hardware implementation in high-speed routers. In the second part of this thesis, we use quantization of virtual time to design a novel packet scheduler called Worst-Case Bin Sort Queuing (WBSQ). WBSQ has constant complexity, and can be utilized in a simple hardware implementation. The WBSQ scheduler uses two methods of quantization. First, WBSQ exploits quantization of virtual time in a manner similar to the bin sorting idea in BSFQ [1] scheduler. In addition, WBSQ simplifies the system virtual time implementation of WF^{2}Q+ [2]. Worst-Case Bin Sort Queuing has good worst-case fairness and delay properties that are demonstrated through both analytical results and simulations.

Practical Fair Queuing Schedulers: Simplification through Quantization

by
Zyad A. Dwekat

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2009

APPROVED BY:

DEDICATION

To my father and to the memory of my mother:

Without their sacrifice, and guidance, I would not have accomplish any of my goals.

To my wife Leila and my kids Tariq, Hamza, Noor and Sarah:

You have been my strength through all these years.

To my brother Eyad and my brother Fuad:

Thank you for your continuous support.

BIOGRAPHY

Zyad Dwekat was born in Dareen, Saudi Arabia. He received an undergraduate degree in Electrical Engineering from Mutah University. He later received a MS degree in Computer Networking from the department of Electrical Engineering in North Carolina State University. Where he is also pursuing his PhD. degree in Electrical Engineering.

He has worked as a Research Assistant at North Carolina State university and has interned at Alcatel Research Center. He then joined Sprint/Nextel as network engineer since 2002. After graduation he will continue working with Sprint/Nextel.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Dr. George N. Rouskas, for his guidance, inspiration and support through-out my graduate study. Without him, this dissertation would not have happened. I would like also to thank Dr. Mladen Vouk, for believing in me since the beginning. Without his support, I would not be able to continue my graduate study.

I would like to extend my appreciation to Dr. Michael Devetsikiotis and Dr. Mihail Sichitiu for their valuable discussions and comments regarding my research and dissertation. Finally, I want to express my utmost appreciation to my wife for her love, support and encouragement throughout my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Quantization and Packet Scheduling Implementation

During the past few years, the popularity of multimedia applications that use computer networks instead of circuit-switched networks has increased. Applications like VOIP teleconferencing, streaming video and audio, email, file transfer etc., all share the same network. Each of these applications has its own requirements of loss rate, delay bound and delay jitter. Sharing introduces the problem of contention for shared resources. The exponential growth of the users of these applications has placed high demand on the network links. As a consequence, control of congestion and maintaining the quality of service of such networks is a challenging task. Given a set of resource requests in a service queue, a server uses scheduling discipline to decide which request to serve next. Packet scheduling is an important means of controlling or avoiding congestion and providing specific quality of service in packet-switched networks.

Many scheduling schemes have been proposed in recent years. For example, weighted fair queuing schemes which emulate the general processor sharing (GPS). These schemes have the best performance in terms of fairness and isolation properties but at the expense of high complexity. This makes these schemes unsuitable for high speed transmission. Many lighter versions of weighted fair queuing have been proposed but still have high complexity. In contrast, weighted round robin (WRR) and its derivative deficit round robin (DRR) have low complexity but have problems with short term fairness. The goal of this research is to design an efficient scheduling schemes that have low complexity and high

Table 1.1: Notation used in this dissertation

N	the number of flows in the system
p	the number of tiered weight classes in the system
r	total link bandwidth
r_i	guranteed bandwidth for flow f_i
$\phi_i = \frac{r_i}{r}$	the weight associated with flow f_i
l_M	maximum packet size
$S_i(t_1, t_2)$	the amount of work received by session i during $[t_1, t_2)$
F_i^k	the finishing time of packet k of flow f_i
Q_i	the queue size of flow f_i at time t
p_i^k	the kth packet on flow f_i

performance regarding fairness and isolation. We had exploited quantization to achieve our goals. In the first scheduler discipline, we used service quantization as our way to build low complexity scheduler. In the second scheduler, we used virtual time quantization. Both approaches resulted in worst-case fair and low complexity implementations.

1.2 Thesis Notations

Major notations used in this dissertation are summarized in Table 1.1. There are N flows $f_1, f_2, \cdots, f_N$ sharing a link of bandwidth r as shown in Figure 2.1. Each flow f_i has a minimum guaranteed rate of r_i. We will assume that $\sum_{i=1}^{N} r_i \leq r$. The weight ϕ_i of flow f_i is defined as its guaranteed rate normalized with respect to the total rate of the link, i.e.,

$$\phi_i = \frac{r_i}{r}. \tag{1.1}$$

The other quantities in Table 1.1 will be discussed in more detail at the time they are introduced in the text.

1.3 Thesis Organization

This thesis is organized as follows. In Chapter 2 we discuss packet scheduling requirements and review a number of scheduling disciplines. Then, we present a new scheduling algorithm called Tiered-Service Fair Queuing (TSFQ). TSFQ represents our first quantization approach which is quantizing the service offered by schedulers. In Chapter 3, we discuss the fixed-size packet case of TSFQ while we discuss the general variable-size packet case in Chapter 4. In Chapter 5 we present another new packet scheduler called the Worst-Case Bin Sort Queuing (WBSQ). WBSQ represents our second approach of quantization which is quantizing the virtual time of the scheduler. Finally, we conclude the thesis and discuss future work in Chapter 6.

Chapter 2

Packet Scheduling

In packet-switched networks, packets from various users (flows) have to share the network resources, including buffer space at the routers and link bandwidth. Whenever resources are shared, contention arises among users seeking service. Consequently, shared resources employ a *scheduling discipline* to resolve contention by determining the order in which users receive service. In particular, the scheduling algorithm is a central component of the quality of service (QoS) architecture of packet switched networks. In this chapter, we discuss the requirements for link scheduling disciplines, we review and classify a number of packet schedulers, and we describe their properties and the trade-offs involved. This discussion sets the stage for the introduction in the few chapter of two new, scalable packet schedulers that is based on the concept of quantization.

2.1 Scheduling Objectives and Requirements

As the Internet has developed into a ubiquitous global communication medium, it is used to carry a constantly evolving mix of applications that is becoming richer as innovation and improvements in technology spawn new services and uses of the network. Nevertheless, network applications can be broadly classified into two fundamental classes: *best-effort* and *guaranteed-service*. The two types of applications differ in terms of their sensitivity to delay and availability of bandwidth, as well as in terms of the level of service quality they expect from the network, as we discuss next.

Most of the applications that were originally developed for the Internet (including email, file transfer, and web browsing) continue to be quite common today, and have *elastic* requirements from the network. In other words, such applications are able to adapt to the bandwidth, delay, or loss performance they experience. Elastic applications do not require any explicit guarantees and work correctly with a best-effort service under which the network only makes a promise to attempt to deliver their packets.

On the other hand, real-time and interactive applications (including streaming audio and video, multimedia conferencing, etc) do require performance bounds from the network in terms of bandwidth, delay, or delay jitter. For instance a voice-over-IP (VoIP) application requires both a minimum bandwidth (generally, between 20-80 Kbps, depending on the voice codec used) and a round-trip delay of about 150 ms (dictated by human ergonomics) to ensure a “good” user experience. These applications require a guarantee of service quality from the network, and the latter must reserve resources on their behalf. Furthermore, the performance that guaranteed-service applications receive is directly affected by the scheduling discipline employed by the nodes along their path, as these disciplines are responsible for scheduling packets on the outgoing links.

Based on this discussion, it is desirable that a packet scheduler possesses three important properties [3]:

- *Isolation and fairness.* When serving best-effort flows, it is important that the scheduler provide isolation among the competing flows and ensure that each flow receives its fair share of the link bandwidth. Isolation prevents misbehaving flows (e.g., flows transmitting too fast) from affecting other flows sharing the same link. In this context, fairness typically refers to max-min fair allocation [4] of link bandwidth among the flows, whereby flows with “small” bandwidth demands receive what they want while flows with “large” demands receive an equal share of the remaining link bandwidth.

- *Performance bounds.* Typically, guaranteed-service applications require a bandwidth bound, i.e., they must receive a minimum amount of bandwidth (measured over an appropriate interval of time). In addition, certain real-time and/or interactive applications may require bounds on packet delay. Such bounds may be expressed *deterministically* (e.g., in the form of a worst-case delay that no packet must exceed) or *statistically* (i.e., in the form of a delay threshold and a probability that any packet's delay will not exceed the threshold). Other performance bounds that have been considered include bounds on the delay jitter (defined as the difference between the largest and smallest delays experience by any packet of a flow) and on packet loss.

- *Low algorithmic complexity.* A link scheduler may need to select the next packet to serve every time a packet departs. As optical link speeds increase from a few Gbps currently to tens of Gbps and beyond, a scheduler may have only a few microseconds or less to make a decision. Hence, in order to operate at wire speeds, the scheduling discipline must be amenable to hardware implementation and require few, preferably simple, operations. In particular, since the links of backbone networks may serve hundreds of thousands of simultaneous flows, the number of operations involved in making a scheduling decision should be independent of the number of flows sharing the link.

These requirements are often contradictory. For instance, the first-come, first-served (FCFS) discipline maintains a single queue of packets and transmits them in the order of their arrival to the queue. This discipline is easy to implement in hardware and is widely deployed in routers. However, since an FCFS scheduler cannot distinguish between different flows, it cannot provide isolation among flows or guarantee per-flow performance bounds. To guarantee such bounds, a scheduler must maintain additional *scheduling state* in the form of separate queues and information regarding the requirements of each flow, which increases the complexity of its implementation.

In the following section, we review the most common packet scheduling disciplines and we discuss the tradeoffs involved with respect to the three requirements above.

2.2 Packet Scheduling Disciplines

In general, packet schedulers can be classified according to their internal structure as follows:

- *Timestamp-based schedulers* maintain a global variable, usually referred to as virtual time, to sort arriving packets and serve them in that order.

- *Frame-based schedulers* divide time into slots of fixed or variable length, and assign slots to flows in some sort of round-robin fashion.

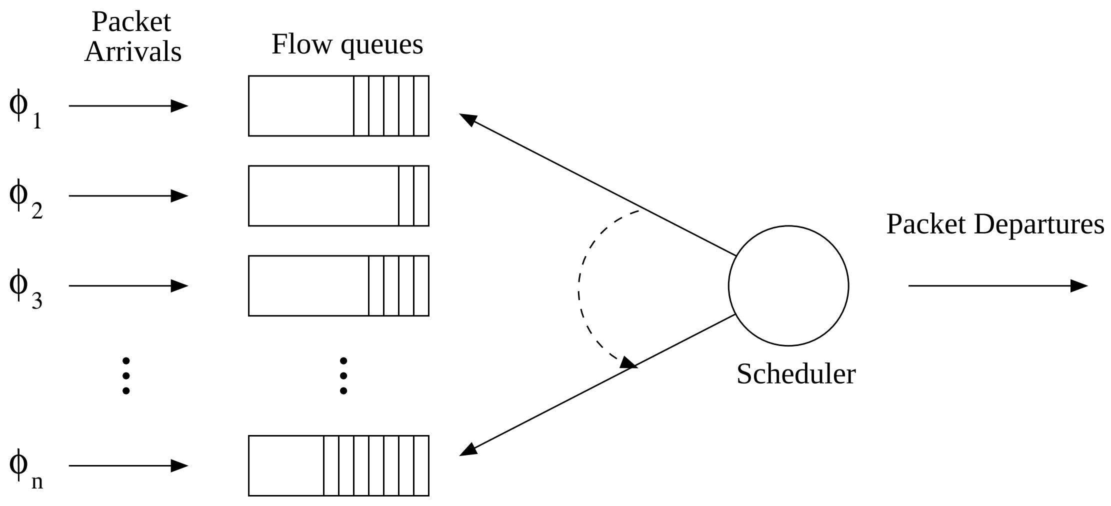


Figure 2.1: Model of link scheduler serving n flows; ϕ_i is the weight assigned to flow i

- *Hybrid schedulers* combine features from both timestamp-based and frame-based schedulers.

Figure 2.1 illustrates the general scheduler model that we will use in our discussion. Specifically, we assume that the scheduler serves n flows and employs per-flow queuing such that an arriving packet belonging to flow $i, i = 1, \ldots, n$, is inserted at the tail of the queue dedicated to this flow. As a result, each flow queue is sorted in increasing order of packet arrival times and its packets are served in a FCFS order. As shown in the figure, each flow i is associated with a positive real weight ϕ_i that is determined in advance (e.g., based on the application's bandwidth or delay requirements). The scheduler uses the weights in some discipline-specific manner to determine which of the head-of-line packets in the flow queues to serve next.

2.2.1 Timestamp-Based Schedulers

Timestamp schedulers emulate the ideal but unimplementable generalized processor sharing (GPS) algorithm by maintaining a virtual time function. Packets are assigned a timestamp based partly on the virtual time value at the time of their arrival, and are transmitted in increasing order of timestamp. In general, timestamp schedulers have good delay and fairness properties, but high implementation complexity, hence there has been limited deployment of such schedulers in high-speed routers.

The complexity of timestamp schedulers arises from two factors.

1. *Packet sorting.* The scheduler selects among the head-of-line packets of the backlogged flows the one with the smallest timestamp to serve next; if there are n backlogged flows, this operation takes time $O(\log n)$ using a priority queue. Whereas current router technology makes it possible to support millions of flows, each with its own queue, the logarithmic complexity and the fact that the priority queue structure is not suited to hardware implementation pose significant challenges.

2. *Virtual time computation.* In order to assign a timestamp to an arriving packet, the scheduler must compute the virtual time function at the time of arrival; as we explain shortly, this computation can be expensive. One of the differentiating characteristics of timestamp scheduler variants is their use of simplified virtual time functions that are more efficient to compute.

It has also been shown that achieving a delay bound relative to GPS that is independent of the number of flows is impossible if the scheduler has a complexity below $O(\log n)$ [5].

Generalized Processor Sharing (GPS)

Generalized processor scheduling (GPS) [6] is an ideal scheduler, a theoretical construct that serves both as a starting point for designing practical scheduling disciplines and as a reference point for evaluating the fairness and delay properties of these disciplines. GPS visits each backlogged flow queue in turn and serves an infinitesimal fraction of the head-of-line packet at each queue. If flows are assigned different weights ϕ_i (refer to Figure 2.1), then the service they receive from GPS is proportional to their weight.

If a queue is empty, GPS skips it to serve the next non-empty queue. Therefore, whenever some queues are empty, backlogged flows will receive additional service in proportion to their weights. Consequently, GPS achieves an *exact* max-min weighted fair bandwidth allocation [3]. It also provides isolation (protection) among flows, since a misbehaving flow is restricted to its fair share and does not affect other flows.

GPS is defined in a theoretical fluid flow model in which multiple queues may be served simultaneously. In a practical packet system, on the other hand, packet transmissions may not be preempted and only one queue may be served at any given time. The schedulers we describe in the following subsections attempt to emulate GPS but are designed for packetized systems.

Weighted Fair Queuing (WFQ)

Weighted fair queuing (WFQ) [6, 7] is an approximation of GPS that serves packets in the order they would complete service had they been served by GPS. Therefore, the WFQ scheduler needs to emulate the operation of the GPS server. To this end, a virtual time function $V(t)$ was proposed in [6] to track the progress of GPS. The rate of change of $V(t)$ is:

$$\frac{\vartheta V(t+\tau)}{\vartheta \tau} = \frac{1}{\sum_{i \in B(t)} \phi_i} \tag{2.1}$$

where $B(t)$ denotes the set of backlogged flows at time t and ϕ_i is the weight assigned to flow i. Let r be the rate of the link (server). In GPS, if flow i is backlogged at time t, it receives a rate of

$$\frac{\vartheta V(t+\tau)}{\vartheta \tau} \phi_i \, r = \frac{\phi_i}{\sum_{i \in B(t)} \phi_i} r. \tag{2.2}$$

In other words, $V(t)$ is the marginal rate at which backlogged flows receive service in GPS.

Suppose that the k-th packet of flow i arrives at time a_i^k, and has length L_i^k. Let S_i^k and F_i^k denote the virtual times at which this packet begins and completes service, respectively, under GPS. Letting $F_i^0 = 0$ for all flows i, we have [6]:

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\} \tag{2.3}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}. \tag{2.4}$$

The WFQ scheduler serves packets in increasing order of their virtual finish times F_i^k, a policy referred to as "smallest virtual finish time first (SFF)" [2].

Let us consider the complexity of WFQ. At packet departure instants, the SFF policy is used to select the next packet to transmit. This selection can take $O(\log n)$ time, where n is the number of (backlogged) flows, if packet virtual finish times are organized in a heap-based priority queue data structure. In addition, there is the cost of maintaining the virtual time function $V(t)$ at packet arrival and departure instants. The worst-case complexity of computing $V(t)$ can be $O(n)$, although the average-case complexity is $O(1)$ [8].

Therefore, WFQ is expensive to implement within core routers that may handle hundreds-of-thousands to millions of flows at any given time.

The degree to which WFQ approximates GPS is determined by two properties that were established in [6]:

- *Bounded delay property.* A packet will finish service in a WFQ system no later than the time it would finish in the corresponding GPS system plus the transmission time of a maximum size packet.
- *Weak service property.* The service (in terms of total number of bits) that a flow receives in a WFQ system does not fall behind the service it would receive in the fluid GPS system by more than one maximum packet size.

While due to the second property above a WFQ system may not fall behind GPS by more than one maximum packet size, it may in fact be ahead of GPS in terms of the service provided to some flows. In particular, it was shown in [9] that WFQ may introduce substantial unfairness relative to GPS in terms of the worst-case fairness index (WFI). WFI is a metric introduced in [9] to represent the maximum time a packet arriving to an empty queue will have to wait before receiving its guaranteed service rate. Specifically, GPS has a WFI of zero, but the WFI of WFQ increases linearly with the number of flows n. Consequently, there may be substantial discrepancies in the service experienced by individual flows under the WFQ and GPS schedulers.

Worst-Case Fair Weighted Fair Queuing (WF^{2}Q)

The WF^{2}Q algorithm was introduced in [9] as a better packet approximation of GPS than WFQ. Specifically, WF^{2}Q employs a “smallest eligible virtual finish time first (SEFF)” policy for scheduling packets. A packet is eligible if its virtual start time is no greater than the current virtual time; hence, the WF^{2}Q scheduler only considers the packets that have started service in GPS when selecting the packet to be transmitted next. It has been shown [9] that WF^{2}Q is work-conserving, maintains the bounded delay property of WFQ, and has these two additional properties:

- *Strong service property.* The service (in terms of total number of bits) that a flow receives from a WF^{2}Q system cannot fall behind (respectively, be ahead of) the service it would receive in the fluid GPS system by more than one maximum packet size (respectively, a fraction of the maximum packet size).

- *Worst-case fairness property.* The worst-case fairness index of WF^{2}Q is a constant independent of the number n of flows served by the scheduler.

The first property implies that the WF^{2}Q scheduler closely tracks the GPS system in terms of the service received by each flow, and due to the second property, WF^{2}Q is an optimal packet scheduler in terms of worst-case fairness [9].

However, the worst-case complexity of WF^{2}Q is $O(n)$, identical to that of WFQ, as both schedulers need to compute the virtual time function $V(t)$.

WF^{2}Q+

A lower-complexity scheduler, WF^{2}Q+ was introduced in [2]. The WF^{2}Q+ scheduler is work-conserving, has the same bounded delay, strong service, and worst-case fairness properties of WF^{2}Q, but uses a different virtual time function that can be computed more efficiently than the function $V(t)$ in (2.1) used by WFQ and WF^{2}Q. The new function is [2]:

$$V_{WF^2Q+}(t+\tau) = \max\left\{V_{WF^2Q+}(t) + \tau, \min_{i\in B(t)}\left\{S_i^{h_i(t)}\right\}\right\}. \tag{2.5}$$

In the above expression, $B(t)$ is the set of backlogged flows at time t, $h_i(t)$ is the sequence number of the packet at the head of flow i's queue at time t, and $S_i^{h_i(t)}$ is the virtual start time of that packet. The minimum operation in the right-hand side of (3.6) can be performed in time $O(\log n)$ in the worst-case using a priority queue structure, hence the overall complexity of WF^{2}Q+ is $O(\log n)$, significantly lower than the $O(n)$ complexity of WFQ and WF^{2}Q.

As pointed out in [2], the WF^{2}Q+ scheduler implementation can be further simplified by maintaining a single pair of start and finish virtual time values per flow, rather than on a per-packet basis. Specifically, only a single pair of values, S_i and F_i, needs to be maintained for each flow i, corresponding to the virtual start and finish times, respectively, of the packet at the head of the queue of flow i. Let $Q_i(t-)$ denote the queue size of flow i just before time t. When a new packet reaches the head of the queue at time t, the values of S_i and F_i are updated according to the following expressions [2]:

$$S_i = \begin{cases} F_i, & Q_i(t-) \neq 0 \\ \max\{F_i, V_{WF^2Q+}(t)\}, & Q_i(t-) = 0 \end{cases} \tag{2.6}$$

$$F_i = S_i + \frac{L_i^k}{\phi_i} \tag{2.7}$$

where L_i^k is the length of this packet and ϕ_i is the weight assigned to the flow.

Overall, the WF^{2}Q+ scheduler achieves tight delay bounds and good worst-case fairness with a relatively low $O(\log n)$ algorithmic complexity.

Self-Clocked Fair Queuing (SCFQ)

The $O(n)$ worst-case algorithmic complexity of the WFQ and WF^{2}Q schedulers is due to the fact that the order of packet transmissions in these queuing schemes is determined by tracking the progress of the fluid-flow GPS reference system, which, in turn, requires the computation of the virtual time function $V(t)$ whose rate of change is defined in (2.1). Self-clocked fair queuing (SCFQ) [8] avoids the computationally expensive emulation of a hypothetical reference system by adopting a self-contained approach to fair queuing. Specifically, instead of using a virtual time to compute the finish times of packets as in expressions (5.1) and (5.2), SCFQ computes the finish time F_i^k of the k-th packet of flow i as:

$$F_i^k = \max\{F_i^{k-1}, F_{cur}\} + \frac{L_i^k}{\phi_i} \tag{2.8}$$

where F_{cur} is the finish time of the packet currently in service, and finish times are initialized to $F_i^0 = 0$ for all flows i. Since the finish times can be computed in $O(1)$ time using expression (2.8), the algorithmic complexity of SCFQ is $O(\log n)$ because of the requirement to select the packet with the smallest finish time for transmission.

Although the rule (2.8) that SCFQ uses to compute packet finish times is easy to implement, the tradeoff is a much larger delay bound than WFQ. In particular, the delay bound provided by SCFQ increases linearly with the number n of flows served by the scheduler, in the worst case [10]. The worst-case fair index (WFI) of SCFQ is the same as that of WFQ, i.e, proportional to the number n of flows.

Start-Time Fair Queuing (SFQ)

Start-time fair queuing (SFQ) [11] is a variant of SCFQ that maintains both a start time and a finish time for each packet. Upon arrival, the k-th packet of flow i is assigned the start time:

$$S_i^k = \max\{F_i^{k-1}, S_{cur}\} \tag{2.9}$$

where S_{cur} is the start time of the packet in service at the time of arrival. The finish time F_i^k of the k-th packet is computed as:

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}. \tag{2.10}$$

Unlike the other packet fair schedulers we have considered so far, SFQ serves packets in increasing order of their *start times*, not their finish times.

It can be seen that expressions (2.9) and (2.10) may be computed in constant time, hence SFQ has the same low algorithmic complexity $O(\log n)$ as SCFQ. However, it has been shown [11] that the worst-case delay of SFQ is significantly lower than with SCFQ. The worst-case fairness properties of SFQ are similar to those of WFQ and SCFQ.

Virtual Clock (VC)

The scheduler introduced in [12] was the first to adopt the notion of a virtual clock to represent the progress of a queuing system in terms of work (service) performed. Whereas the similar notion of virtual time has been used by fair packet queuing schedulers such as WFQ to emulate GPS, the virtual clock scheduler instead emulates time division multiplexing (TDM). Specifically, the k-th packet of flow i arriving at time t is assigned the finish time:

$$F_i^k = \max\{F_i^{k-1}, t\} + \frac{L_i^k}{\phi_i} \tag{2.11}$$

Note that the above expression uses real time t instead of virtual time, greatly simplifying the computation of finish times. The scheduler serves packets in increasing order of their finish times, hence the complexity of virtual clock is $O(\log n)$.

Despite its simplicity, the virtual clock scheduler is able to provide delay bounds to flows. However, the use of real time t in (2.11) does not accurately represent the progress of work in the system upon the arrival of the packet. As a result, the worst-case fairness index of virtual clock can be arbitrarily large [13], even in the case of only two sessions.

2.2.2 Frame-Based Schedulers

Even with simplified virtual time computations, timestamp-based schedulers incur a substantial per-packet overhead that is related to selecting the packet with the smallest finish time to be transmitted next. Frame-based schedulers eliminate the need for packet sorting and hence achieve an $O(1)$ packet processing operation. Such schedulers typically operate by dividing time into frames. Within each frame (also referred to as round), flows are mapped to time slots of fixed or variable length and are served in a round-robin manner. Because of their low implementation complexity, frame-based schedulers have been widely deployed in high-speed routers. However, these schedulers have poor delay bound and fairness properties under most realistic traffic conditions.

Weighted Round-Robin (WRR)

The round-robin scheduler, which serves a single packet from each flow with backlogged traffic, is the simplest emulation of GPS and an early form of fair queuing. If all packets have the same size (e.g., as in ATM networks), and all flows are assigned identical weights, then round-robin is a reasonably good approximation of GPS. If flows have different weights, the more general weighted round-robin (WRR) scheduler may be used. WRR serves a number of packets from a flow in proportion to the flow's weight.

Despite its simplicity, the WRR scheduler has several limitations. In networks with variable packet sizes, emulating GPS correctly requires advance knowledge of each flow's mean packet size. Specifically, to allocate bandwidth fairly, the WRR scheduler must serve the flows according to a set of normalized weights obtained by dividing the weight of each flow by its mean packet size. In practice, however, the mean packet size for a flow may be difficult or even impossible to predict, e.g., when the flow carries compressed video whereby the packet sizes are strongly dependent on the nature of the video scenes. If mean packet sizes cannot be accurately predicted, the bandwidth allocation under WRR may be significantly different than under GPS.

A second drawback of WRR is that each backlogged flow is served exactly once within every frame. As a result, the WRR scheduler is fair only over time scales longer than the frame size: once a flow is served, it must wait for $n-1$ other flows before it gets service again. If the number n of flows is large, as is the case in core routers, this may lead to long periods of unfairness. A related issue has to do with burstiness: within each frame, a flow transmits all its packets at once, which will arrive at the downstream router in a burst. As a result, WRR has poor delay and burstiness properties.

Deficit Round-Robin (DRR)

Deficit round-robin (DRR) [14] is an improved version of WRR that makes it possible to allocate bandwidth fairly in a network with variable packet sizes without advance knowledge of each flow's mean packet size. To this end, the DRR scheduler maintains a quantum and a deficit counter for each flow that it serves. The quantum of each flow i is proportional to its weight ϕ_i. The deficit counter of each flow i is initialized to zero, and it is used to keep track of the currently unused portion of its allocated bandwidth.

Within each frame (round), the DRR scheduler visits each backlogged flow exactly once, and serves a number of packets such that the sum of their lengths does not exceed the sum of the flow's quantum and deficit counter. The value of the deficit counter is then updated to the unused portion (if any) of this latter amount, and carried over to the next round. As a result, each flow receives its fair share of the bandwidth without the need for the scheduler to estimate the mean packet size for each flow it serves.

The DRR scheduler performs a constant amount of work every time it visits the queue of a flow, its operation is easy to implement in hardware, and variants of this scheme are employed in commercial high-speed routers. However, within each frame, a flow transmits its entire quantum at once, and must then wait for all other flows to transmit before it is served again. Consequently, DRR has the same drawbacks as WRR, namely, unfairness at short time scales and poor delay and burstiness properties.

2.2.3 Hybrid Schedulers

As we have seen, the design of packet schedulers involves tradeoffs between algorithmic complexity, on the one hand, and performance bounds and fairness, on the other hand. Specifically, frame-based schedulers are simple to implement but provide poor delay bounds and suffer from short-term unfairness, while timestamp-based schedulers have good delay and fairness properties but high implementation complexity. More recently, hybrid designs have been proposed that incorporate some elements of timestamp-based schedulers into a frame-based scheme, the latter usually being a version of DRR. In doing so, hybrid schedulers attempt to combine the best of both worlds, i.e., improve the delay, fairness, and output burstiness properties of the frame-based scheduler while maintaining low implementation complexity.

Smoothed Round-Robin (SRR)

We note that the limitations of DRR in terms of short-term fairness and high output burstiness are due to the fact that the scheduler visits each flow exactly once within each frame (round), and transmits an amount of data from its queue equal to its quantum (plus the deficit counter, if a smaller amount was served in the previous round). Hence, the service each flow receives consists of long periods of inactivity (whose length is proportional to the number n of flows served by the scheduler) followed by short periods of service that result in back-to-back data transmissions (bursts). Clearly, if the service that a flow receives could be spread over the entire frame, then the fairness and output burstiness of the scheduler would be improved.

The smoothed round-robin (SRR) scheduler [15] addresses this shortcoming of DRR by spreading the quantum of service allocated to a flow over the frame using a technique based on a “weight spread sequence.” The SRR scheduler needs $O(1)$ time to select a packet for transmission, and has better delay bounds than DRR. On the other hand, the worst-case delay that a packet may experience is proportional to the number n of flows served by the scheduler.

Bin Sort Fair Queuing (BSFQ)

Bin sort fair queuing (BSFQ) [1] takes a different approach to reducing the complexity of timestamp-based schedulers. In particular, BSFQ is designed to reduce the computational effort required for the two main operations of a timestamp-based scheduler: computing the virtual finish time (timestamp) of each arriving packet, and sorting packets in increasing order of timestamps. To this end, virtual time is divided into slots (bins) of length Δ, where Δ is a configurable parameter, and the scheduler maintains a virtual system clock that is equal to the left endpoint of the current slot. Arriving packets are assigned a virtual finish time using an expression similar to the one for SCFQ in (2.8), that can be computed in constant time. Packets with finish times that fall within the same slot, are inserted in a first-in, first-out (FIFO) queue associated with this slot. In other words, there is no sorting of packets that have finish times "close" to each other, as determined by the length Δ of a slot. Therefore, this "bin sorting" operation takes $O(1)$ time. When the virtual clock is equal to the left endpoint of slot i, the scheduler serves all the packets in the FIFO queue associated with slot i. When all the packets of the queue have been transmitted, the virtual clock is incremented by Δ and the scheduler serves the FIFO queue of the next slot $i + 1$.

The BSFQ scheduler is scalable and is easy to implement in hardware. Its fairness and delay guarantees depend strongly on the value of parameter Δ. When Δ is large, BSFQ reduces to FCFS, while when Δ is small, its operation is similar to that of SCFQ. While smaller Δ values result in better fairness and delay guarantees, the amount of state information that the scheduler needs to maintain increases and its efficiency decreases as the value of Δ decreases. Therefore, determining an appropriate value for Δ is a complex task that involves several tradeoffs.

Stratified Round-Robin (S-RR)

Stratified round-robin (S-RR) [16] operates by grouping ("stratifying") flows into *flow classes* based on their weights. An exponential grouping is used, such that the k-th flow class consists of flows i with weights such that: $\frac{1}{2^k} \leq \phi_i < \frac{1}{2^{k-1}}$. S-RR has two scheduling components: an *intra-class* schedulerand an *inter-class* scheduler.The inter-class scheduler assigns a *scheduling interval* to each flow class such that the k-th class is assigned an interval of length 2^k slots. Within a class, flows are scheduled in the associated scheduling intervals using a variant of DRR that gives each flow a quantum that is proportional to its weight.

S-RR has low complexity and provides delay and fairness guarantees similar to those of DRR. However, it improves the worst-case delay a single packet may experience to a small constant, whereas under DRR and BSFQ this value is proportional to the number n of flows served by the scheduler.

Fair Round-Robin (FRR)

The fair round-robin (FRR) scheduler [17] is similar to S-RR in that flows are grouped into classes using the same exponential grouping, and has the same structure in that it employs both an intra-class and an inter-class scheduling component. The inter-class scheduler is timestamp-based, and determines the time a packet from each class is to be scheduled by taking into account the time-varying weight of each class (which changes over time as flows within a class become active or inactive). FRR assigns finish times to *flow classes*, not individual flows, by keeping track of the corresponding GPS system. This scheduler always serves the *eligible* flow class with the smallest finish time; eligibility of a flow class is defined as a generalization of the eligibility criterion introduced by the WF^{2}Q scheduler. Since the inter-class scheduler operates on the basis of flow classes, emulating GPS (i.e., computing the virtual time function) takes time proportional to the number m of classes, which, for a given system is a small constant (determined by the exponential grouping employed) that is independent of the number n of flows.

The intra-class scheduler has two functions. First, it needs to compute the class weight to pass to the inter-class scheduler; the latter uses these weights to determine the order in which each class is served, as we explained above. Second, it must decide the order in which packets from the various flows within the class will be transmitted whenever the inter-class scheduler serves this class. The intra-class scheduler uses a frame-based approach similar to DRR, but with a modification to account for the weight differences among the flows within the same class.

The FRR scheduler has $O(1)$ algorithmic complexity, is worst-case fair, and provides a single packet delay bound that is equal to a small constant, similar to S-RR.

Chapter 3

Tiered-Service Fair Queuing (TSFQ)- The Fixed-Size Packet Case

As we explained in the previous chapter, the high algorithmic complexity of timestamp-based schedulers is due to two fundamental operations: (1) computation of the virtual time function to track the corresponding GPS system, and (2) packet sorting to select the packet with the smallest timestamp to serve next. The WF^{2}Q+ scheduler we described in Section 2.2.1 implements both operations in time $O(\log n)$, where n is the number of flows served. Importantly, WF^{2}Q+ closely emulates the ideal GPS fluid-flow scheduler, and thus it provides tight delay bounds to all flows and achieves a constant worst-case fair index. Although several timestamp-based or hybrid scheduler variants have been developed with lower complexity, these schedulers provide looser delay and fairness guarantees than WF^{2}Q+. As a result, the service received by individual flows under these simpler schedulers may be significantly different than the service they would receive under GPS.

We also note that the packet schedulers we reviewed in the previous chapter were designed under the assumption that both flow weights and packet sizes may take arbitrary values. This fundamental assumption underlies the high complexity of the virtual time computation and packet sorting operations. However, two important observations regarding

Internet traffic characteristics suggest that the implementation of packet schedulers may be simplified significantly without compromising their delay and fairness properties.

- *Flow weights.* First, traffic flows are unlikely to have arbitrary weights. For instance, flows of guaranteed-service applications may be grouped into a small set of classes depending on the nature of the application (e.g., "voice", "video," "game,", etc) with the flows in each class having similar bandwidth and delay requirements. On the other hand, whereas best-effort applications have elastic requirements that adjust to the available rate, their bandwidth requirements are typically limited to the access bandwidth available to the user. Recall also that most Internet service providers offer some type of tiered service in which users may select only from a small set of bandwidth tiers. The practical implication of this fact is that the rates requested by flows (equivalently, the flow weights in the fair queuing system) are not arbitrary, but are limited to a small set of values that are typically known in advance. As we explain shortly, it is possible to speed up considerably the computation of the virtual time function if the scheduler is designed so as to handle only a small set of discrete flow weights.

- *Packet sizes.* The second observation is that in the Internet, the vast majority (i.e., up to 90%) of packets have a fixed length that takes one of a small number of values [18, 19]. Therefore, the scheduler may employ simple queuing structures that simplify, or even completely eliminate the need for, packet sorting operations.

The main contribution of this chapter is a new implementation of WF^{2}Q+ that exploits the above observations to ensure that the two main scheduling operations, namely, computing the virtual time function and selecting the next packet to be transmitted, are performed in time that is independent of the number n of flows, while at the same time maintaining the excellent delay and fairness properties of the original scheduler. We refer to this scheduler as *tiered-service fair queuing (TSFQ)* [20].

In this chapter we make the additional assumption that *all packets of all flows have constant size* L (i.e., $L_i^k = L \ \forall \ i, k$). We will remove this assumption in the next chapter; however, we note that the implementation we present in this chapter is of practical importance to ATM networks. In the remainder of this chapter, we first explain the operation of TSFQ, we study TSFQ fairness analytically, and we present experimental performance results.

3.1 Tiered Service Fair Queuing (TSFQ)

We consider a link scheduler which serves n flows and employs per-flow queuing, i.e., it allocates a FIFO buffer to each flow, as shown in Figure 2.1. The scheduler supports p distinct tiers of service, where $p \ll n$ is a small constant (e.g., $p \approx 10 - 15$). The l-th tier is characterized by a positive real weight $\phi_l, l = 1, \ldots, p$. Each flow i is mapped to one of the p service levels, i.e., it is assigned one of the p weights ϕ_l; we assume that this assignment remains fixed throughout the duration of the flow. The mapping of flows to service tiers is performed at the time the flow enters the network by taking into account the QoS requirements of the application or the bandwidth tier to which the user subscribes. We make the assumption that the link is configured with the number p of service tiers and the associated weights ϕ_j. These parameters may be determined in advance by the network provider as part of the network planning process, by using empirical information regarding the user demands and employing the techniques we developed in earlier chapters of this book.

Before proceeding, we emphasize that our assumption regarding flow weights is different from the approach taken by the stratified round-robin (S-RR) [16] and fair round-robin (FRR) [17] schedulers. Specifically, S-RR and FRR allow flows to have *arbitrary* weights, but "stratify" them into a small number of classes using exponential grouping. In contrast, we assume that *all flows* within a service tier are assigned the *same* weight. To make the distinction clear, we use the term *flow tier* to refer to a set of flows with the same weight, instead of the term *flow class* that was used in [16, 17] to refer to a group of flows with similar weights as determined by the specific exponential grouping method employed.

3.1.1 Logical Operation

The tiered service fair queuing (TSFQ) scheduler operates in a manner similar to WF^{2}Q+ in that:

- it uses the same virtual time function shown in expression (3.6);
- it maintains a single pair of values, S_i and F_i for each flow i, corresponding to the

virtual start and finish times, respectively, of the packet at the head of the FIFO queue of flow i; these values are updated according to expressions (2.6) and (2.7); and

- it employs the SEFF policy to serve packets.

The TSFQ scheduler logically consists of two components, as illustrated in Figure 3.1. The first component comprises of p identical *intra-tier schedulers*, while the second component is a single *inter-tier scheduler.* The main function of each component is as follows:

- *Intra-tier scheduler.* The l-th intra-tier scheduler uses the SEFF policy to select, *among the flows of the l-th service tier, $l = 1, \ldots, p$, only*, the flow i with the minimum virtual finish time F_i. The structure and operation of the intra-tier scheduler are described in detail in the following sections.

- *Inter-tier scheduler.* The inter-tier scheduler simply serves the packet at the head of the queue with the smallest virtual finish time among the p flows selected by the corresponding intra-tier schedulers. Since p is a small constant for the given link, the packet to be transmitted next can be determined in time that is independent of the number of flows, and in fact, this operation can be performed in constant time in hardware. Hence, the implementation of the inter-tier scheduler is straightforward and does not require any priority queue data structure to be maintained.

We note that the logical structure of the TSFQ scheduler is similar to the structure of the S-RR and FRR schedulers which both consist of *intra-class* and *inter-class* scheduling components. However, there are significant differences in the functionality and operation of these schedulers. Specifically, the inter-class scheduler of S-RR assigns scheduling intervals to each flow class, while the inter-class scheduler of FRR assigns weights to flow classes, not individual flows, and serves the class with the smallest timestamp. The TSFQ inter-tier scheduler, on the other hand, simply serves the flow with the smallest finish time among the p such flows across the p tiers, hence its operation is much simpler. The intra-class scheduler of S-RR serves flows within its assigned scheduling interval using a variant of DRR, while the intra-class scheduler of FRR also uses a (different) variant of DRR. In contrast, the intra-tier scheduler of TSFQ (described shortly) uses simple queuing structures to maintain the flows within its tier sorted in decreasing order of finish time.

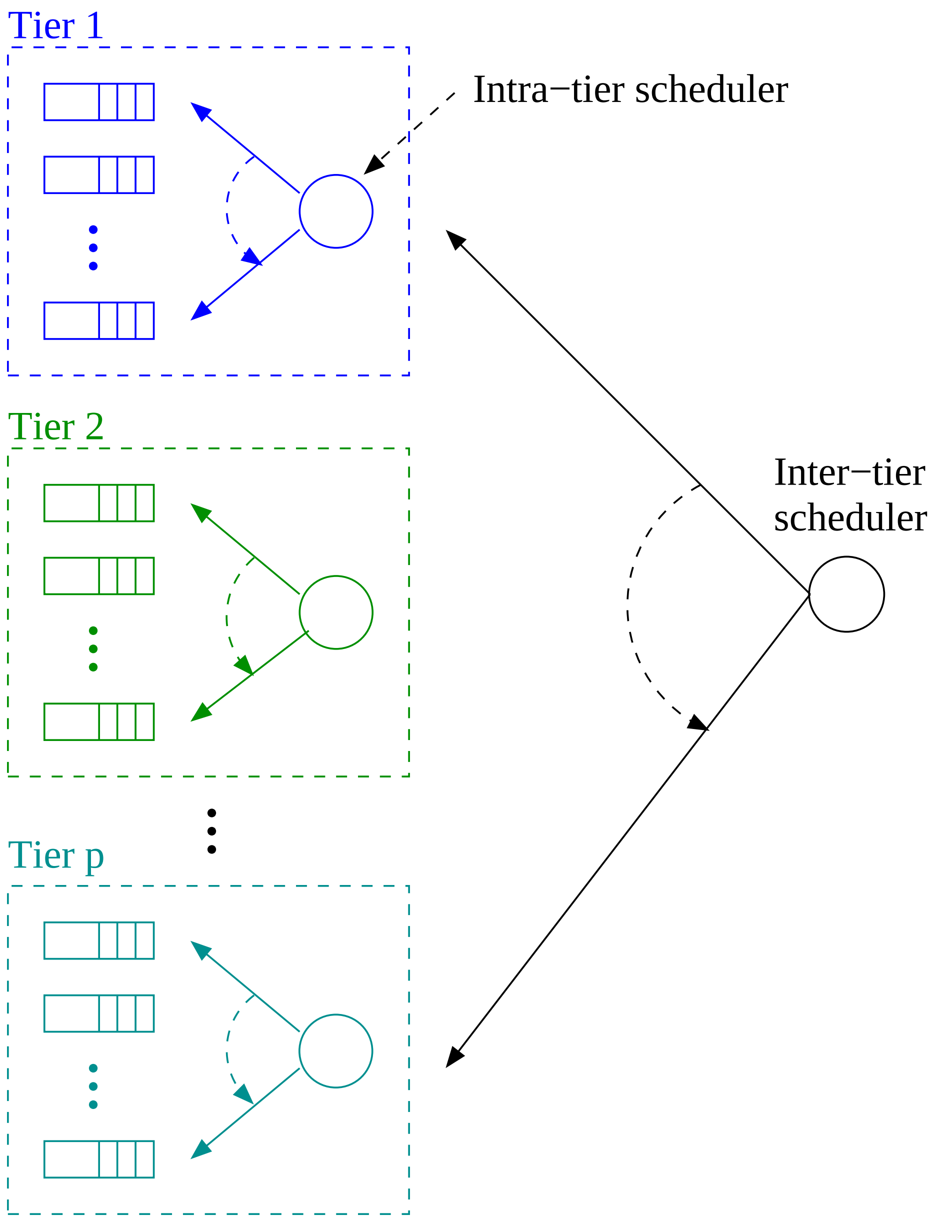


Figure 3.1: Logical diagram of the TSFQ scheduler with p service tiers

3.1.2 Virtual Time Computation

Let $S^{(l)}(t), l = 1, \cdots, p$, denote the virtual start time of the flow with the smallest finish time among the flows of the l-th service tier at time t. Then, we may rewrite the expression (3.6) of the virtual time function as:

$$V_{TSFQ}(t+\tau) = \max\left\{V_{TSFQ}(t) + \tau, \min_{l=1,\ldots,p}\left\{S^{(l)}(t)\right\}\right\}. \tag{3.1}$$

Assuming that at any time t each intra-tier scheduler keeps track of the flow within its tier with the smallest finish time, the minimum operation in the right-hand side of expression (3.1) can be implemented in $O(1)$ time. Hence, the virtual time computation takes time that is independent of the number n of flows.

So far, we have shown that both the virtual time computation and the inter-tier scheduling operations take time that depends only on the number p of tiers, which is a small constant for a given scheduler. Therefore, the critical component of TSFQ is the intra-tier scheduler which is responsible for identifying (selecting) the flow with the minimum virtual finish among the flows in its tier. In the next two sections we show that by employing simple queuing structures this selection operation can be implemented efficiently.

3.2 Intra-Tier Scheduler

The l-th TSFQ intra-tier scheduler, $l = 1, \ldots, p$, serves flows belonging to the l-th service tier and have been assigned the same weight ϕ_l. The p intra-tier schedulers are identical and operate independently of each other. Therefore, in this and the next section we consider the operation of a single intra-tier scheduler in which all flows have identical weights. For simplicity, we let ϕ denote the weight assigned to all the flows served by the scheduler.

As we stated in the beginning of this chapter, we make the additional assumption that *all packets of all flows have constant size* L (i.e., $L_i^k = L \; \forall \; i, k$). We will remove this assumption in the next chapter; however, we note that the implementation we present in this chapter is of practical importance to ATM networks.

In a system with fixed-size packets and flows of identical weight, sorting flows according to their virtual start times produces an identical order to sorting them according to their virtual finish times. This property is formally expressed in the following lemma.

Lemma 3.2.1 *Consider flows i and j with $\phi_i = \phi_j = \phi$ and packets of fixed size L. Let S_i be the virtual start time of flow i, and S_j be the virtual start time of flow j. Then:*

$$S_i \;\leq\; S_j \;\Leftrightarrow\; F_i \;\leq\; F_j \tag{3.2}$$

Proof. TSFQ assigns finish times to flows using the expressions (2.6) and 2.7). Under the assumption of fixed packet size and identical weights, we may rewrite these expressions as:

$$S_i \;=\; \begin{cases} F_i, & Q_i(t-) \neq 0 \\ \max\{F_i, V_{TSFQ^+}(t)\}, & Q_i(t-) = 0 \end{cases} \tag{3.3}$$

$$F_i \;=\; S_i \;+\; \frac{L}{\phi}. \tag{3.4}$$

Therefore, we have that:

$$S_i \;\leq\; S_j \;\Leftrightarrow\; S_i + \frac{L}{\phi} \;\leq\; S_j + \frac{L}{\phi} \;\Leftrightarrow\; F_i \;\leq\; F_j. \tag{3.5}$$

■

3.2.1 Queue Structure and Operation

The intra-tier scheduler for fixed-length packets consists of a simple FIFO queue, as illustrated in Figure 3.2. The scheduler maintains a single *token* κ_i for each flow i that it serves. Initially (i.e., at time $t = 0$, before any packet arrivals to the system), the FIFO queue is empty. Tokens are inserted at the tail of the FIFO queue representing the order in which flows will be served, and a token is removed from the head of the FIFO queue whenever it is selected for service by the inter-tier scheduler.

The operation of the scheduler is fully described by the actions taken whenever a relevant event takes place. The relevant events occur when (1) a packet arrives, (2) a flow becomes eligible for service, or (3) a packet departs (is served).

- *Packet arrival.* A packet of flow i arriving at time t is inserted at the tail of this flow's queue. If flow i was active just prior to the arrival t (i.e., its queue was non-empty, hence $Q_i(t-) \neq 0$, using the notation of Section 2.2.1), then no other action is taken. If, on the other hand, flow i was inactive prior to the arrival (i.e., $Q_i(t-) = 0$),

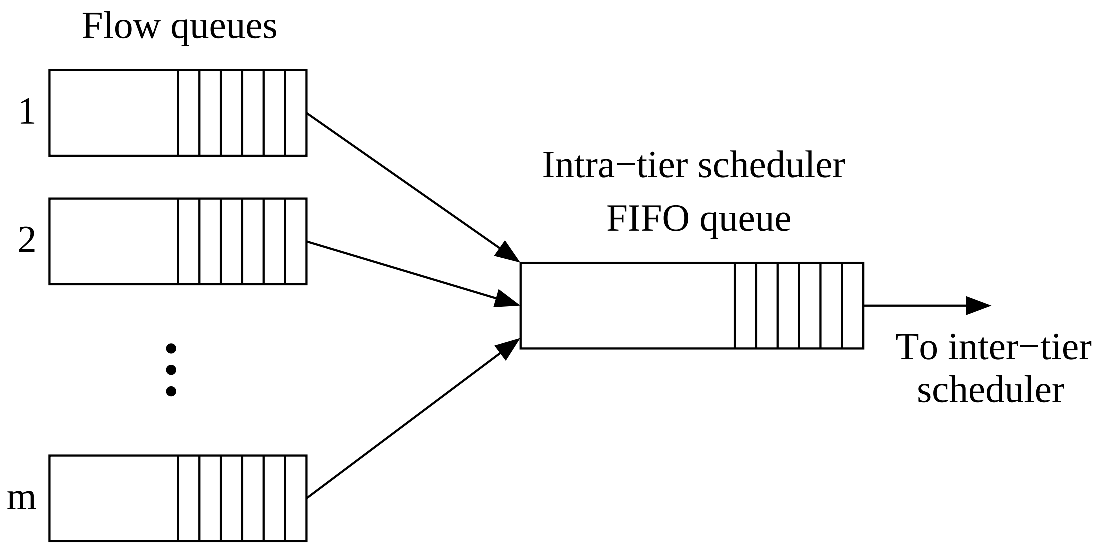


Figure 3.2: Queue structure of the intra-tier scheduler for fixed-size packets

then this arriving packet reaches the head of this flow's queue at time t, and the start time S_i and finish time F_i of flow i are updated according to expressions (3.3) and (3.4), respectively. If this previously inactive flow i becomes eligible at time t (i.e., $S_i \leq V_{TSFQ}(t)$ after the update), then the next event is triggered, otherwise no other action is taken.

- *A flow becomes eligible for service.* When a flow i becomes eligible at time t (i.e., $S_i = V_{TSFQ}(t)$), then the token κ_i corresponding to this flow is inserted at the tail of the scheduler's FIFO queue.

- *Packet departure.* Let κ_i be the token at the head of the scheduler's FIFO queue at the time the inter-tier scheduler selects this tier to serve. Then, the packet at the head of the queue of flow i is served and token κ_i is removed from the scheduler's FIFO queue. If flow i becomes inactive, then no other action is taken. Otherwise, a new packet reaches the head of this flow's queue, and the start time S_i and finish time F_i are updated according to expressions (3.3) and (3.4), respectively. If the flow becomes eligible, then the corresponding event above is triggered, otherwise no action is taken.

Based on these actions, it is easy to see that token κ_i is in the scheduler's FIFO queue *if and only if* flow i is eligible for service. Therefore, we have the following results.

Lemma 3.2.2 *Considering only the flows of a given tier, the intra-tier scheduler of Figure 3.2 is identical to the WF^{2}Q+ scheduler [2].*

Proof. Since tokens are inserted into the FIFO queue at the moment the corresponding flows become eligible for service (i.e., at the moment their virtual start time becomes equal to the current time), tokens in the FIFO queue are sorted in increasing order of the corresponding flows' virtual start times. Because of Lemma 3.2.1, the queue is sorted in increasing order of the virtual finish times, which is the order in which flows are served under WF^{2}Q+. Since (1) token arrivals to the FIFO queue take place at exactly the same instants that the corresponding head-of-line packets are considered for service under WF^{2}Q+, and (2) the order of service is identical, the two schedulers are identical under the assumption of flows with fixed-size packets and identical weights. ■

Lemma 3.2.3 *The TSFQ scheduler consisting of p intra-tier schedulers and one inter-tier scheduler is identical to WF^{2}Q+.*

Proof. Each of the p intra-tier schedulers maintains a FIFO queue that sorts the flows in its tier in increasing order of their start (equivalently, finish) times, identical to the order in which they are considered under WF^{2}Q+. The inter-tier scheduler serves the p flows with tokens at the head of the p intra-tier FIFO queues in increasing order of their virtual start (finish) times. Consequently, the TSFQ scheduler overall is identical to WF^{2}Q+. ■

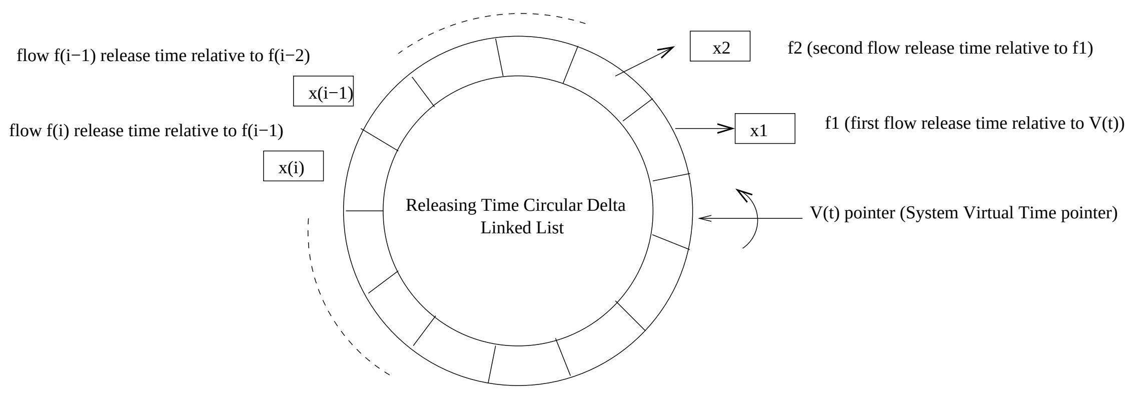
Based on these results and the discussion in Section 3.1.2, we conclude that the TSFQ (intra- and inter-tier) scheduler achieves the worst-case fairness and delay properties of WF^{2}Q+ with an algorithmic complexity of $O(1)$. Note that this conclusion does not contradict the findings of [5] which suggest that the $O(\log n)$ time complexity is fundamental to achieving good delay bounds. The analysis in [5] assumes that flow weights and packet sizes can take arbitrary values, whereas the result of Lemma 3.2.3 only holds under the specific assumptions of fixed flow weights and packet lengths.

3.2.2 Delta List Implementation

In TSFQ, we need to delay releasing the packets until they become eligible. This task can be accomplished efficiently using a delta linked list. Each item of the delta list corresponds to an event that is to take place at some future time; in this case, an event corresponding to the release of some packet to the FIFO queue. The first item of the delta list records the amount of time remaining until the first event is to take place, while the i-th item, $i > 1$, records the amount of time for an event i to occur after event $i - 1$ has taken place. At the instant the head-of-line packet of some flow i leaves the system, an item with the appropriate time value is inserted at the tail of the delta list. A background process periodically decrements the time value of the first item in the list based on the amount of virtual time that has elapsed; this amount is determined by the amount of real time elapsed since the last update and the rate of change in (2.1). Once the time value of the first item reaches zero, the item is removed from the list and the corresponding packet is released to the FIFO queue.

In TSFQ, we assign a circular delta linked list for each intra-tier queue as shown in Figure 3.3. We maintain a pointer that represents the current system virtual time $V(t)$ value on the circular delta list. The algorithm uses the special characteristics for fixed-size packet case, i.e., finishing time sorting is the same as start time sorting. If a packet P_i^1 was received as the first packet in the current busy time of the scheduler, this packet will be released directly to the FIFO queue since its start time is equal to the current $V_{TSFQ}(t)$ value and an entry will be inserted for this flow at the circular delta list, i.e., relative to the current virtual time pointer. The value of delta entry x_i for this flow in this case is $\frac{L}{\phi}$. This comes from the fact that the finishing time of this packet is the start time of the next packet for this flow. In this case the next packet will only be released once $V_{TSFQ}(t)$ passes this entry.

If any other flow f_j receives its first packet P_j^1 at time A_j^1, this packet will be released directly to the FIFO queue since its start time is equal to the current $V_{TSFQ}(t)$ value and an entry will be added at the end of the delta list with delta value x_i equal to the difference between the two finishing times $F_j^1 - F_i^1$ since $\frac{L}{\phi}$ is a constant value for all flows. If a packet arrives to a flow and there is an active entry in the circular delta list, this packet will have to be kept in the flow's queue. Scheduler will not release this packet until



flow f(i−1) release time relative to f(i−2)
x(i−1)
flow f(i) release time relative to f(i−1)
x(i)
Releasing Time Circular Delta Linked List
x2
f2 (second flow release time relative to f1)
x1
f1 (first flow release time relative to V(t))
V(t) pointer (System Virtual Time pointer)

Figure 3.3: Intra-Tier Releasing Time Circular Linked List

it becomes the head of the line packet and the current $V_{TSFQ}(t)$ pointer passes through its flow's entry.

The current $V_{TSFQ}(t)$ pointer will not move from one point to another in the delta linked list until the change in system virtual time is more than the next delta value. When the pointer passes an entry of a flow, the next packet of that flow is released to the common FIFO queue. If the pointer moved over the entry while there was no more packets, this entry will be removed from the delta list and the flow will be considered inactive. If the flow become active again, a new entry will be inserted at the end of the delta linked list.

Notice that due to identical length and weight properties, the only factor affecting the order of these flows' entries is the order of their arrival. That is why inserting the new entries at the end of the list is sufficient to maintain the correct order.

As we mentioned before, the virtual time calculation uses the same expression as WF^{2}Q++ [2]:

$$V_{WF^2Q+}(t+\tau) = \max\left\{V_{WF^2Q+}(t)+\tau, \min_{i\in B(t)}\left\{S_i^{h_i(t)}\right\}\right\}. \tag{3.6}$$

where $B(t)$ is the set of sessions backlogged in system at time t, and S_i is the virtual start time of the packet. We can observe there is a minimum operation that can introduce $log(n)$ complexity on the scheduler implementation. However, using the circular delta linked list we can, in constant time, find the minimum active starting time $\min_{i\in B(t)}\left\{S_i^{h_i(t)}\right\}$. In this case, it is the first entry from the current virtual time pointer that has an active queue.

3.2.3 Fairness Analysis

In this section, we will study the fairness of the TSFQ scheduler. First we will study the fairness of the intra-tier scheduler alone, then we will conclude by giving the overall fairness for both intra-tier and inter-tier schedulers combined. Fairness will be assessed using two standard measures: proportional fairness and worst-case fairness.

Lemma 3.2.4 (Proportional Fairness for Intra-Tier TSFQ Scheduler) *In any time period (t_1, t_2) during which flows i and j are backlogged, we have that:*

$$\left| \frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j} \right| \quad \leq \quad \frac{6\ L}{r_i} \tag{3.7}$$

where $S_i(t_1,t_2)$ is the amount of work received by flow i during (t_1,t_2)

Proof.

Let us assume that two flows f_i and f_j were continuously backlogged at time interval (t_1, t_2). Let P_k^i represent the first packet that departed in this time interval from flow f_i at time L_i^k and assume P_i^l be the last packet to depart in this time interval from flow f_i at time L_i^l as shown in Figure 3.4.

Since flow f_i is continually backlogged in interval (t_1, t_2), the starting time

$$S_i^h \ = \ max(F_i^{h-1}, V(A_i^h)) \ = \ F_i^{h-1} \tag{3.8}$$

for all $h = k+1, k+2, ..., l-1, l$.

From this we can say that

$$\sum_{x=k+1}^{l} L \ \leq \ s_i(t_1,t_2) \ \leq \ \sum_{x=k}^{l+1} L \tag{3.9}$$

where L is the packet length.

Since L and r_i are constant for all flows

$$(l \ - \ k)\ L \ \leq \ s_i(t_1,t_2) \ \leq \ (l \ - \ k \ + \ 2)\ L \tag{3.10}$$

Let us first find the upper limit. Since the virtual time $V(t)$ is non-decreasing function of time, we have

$$V(t_2) \ - \ V(t_1) \ \geq \ V(L_i^l) \ - \ V(L_i^k) \tag{3.11}$$

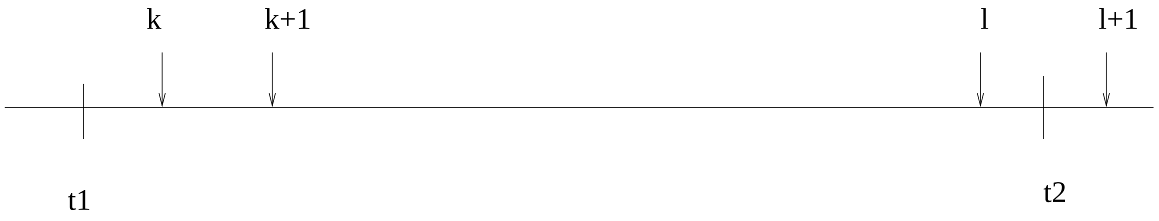


Figure 3.4: Packets in time interval (t_1,t_2).

Since TSFQ does not release packets to the common queue before the virtual time is equal to its start time, packet P_l is not served at time L_l unless

$$V(L_i^l) \geq S_i^l \tag{3.12}$$

Since the flow is backlogged at this time.

$$V(L_i^l) \geq F_i^l - \frac{L}{r_i} \tag{3.13}$$

Since $\Sigma_{allflows}\ r_i \leq r$, the packet will be serviced before or at its theoretical finishing time, hence

$$V(L_i^k) \leq F_i^k \tag{3.14}$$

$$V(t_2) - V(t_1) > F_i^l - \frac{L}{r_i} - F_i^k \tag{3.15}$$

Since flow f_i is continuously backlogged at least since packet k

$$F_i^l = F_i^k + \sum_{x=k+1}^{x=l} \frac{L}{r_i} = F_i^k + (l - k)\frac{L}{r_i} \tag{3.16}$$

since L and r_i are constant for all flows.

$$V(t_2) - V(t_1) > F_i^k + (l - k)\frac{L}{r_i} - \frac{L}{r_i} - F_i^k \tag{3.17}$$

$$V(t_2) - V(t_1) > (l - k - 1)\frac{L}{r_i} \tag{3.18}$$

from this equation and equation (3.10) we obtain:

$$s_i(t_1, t_2) < V(t_2) - V(t_1) + \frac{3\ L}{r_i} \tag{3.19}$$

$$s_i(t_1, t_2) \ \leq \ V(t_2) \ - \ V(t_1) \ + \ \frac{3 \ l_i^{max}}{r_i} \tag{3.20}$$

To obtain the lower bound we need to consider two cases.

Case I: $F_i^{k-1} \ \geq \ V(A_i^k)$

Since there is no departure in (t_1, L_i^k), we have $L_i^{k \ - \ 1} \ \leq \ t_1$ then

$$V(t_2) \ - \ V(t_1) \ \leq \ V(L_i^{l \ + \ 1}) \ - \ V(L_i^{k \ - \ 1}) \tag{3.21}$$

Since we assume that the flows are not oversubscribed i.e. $\Sigma_{allflows} r_i \leq r$ we have

$$V(L_i^{l \ + \ 1}) \ \leq \ F_i^{l \ + \ 1} \tag{3.22}$$

Since TSFQ only releases packets when their start time started in virtual time, we have $V(L_i^{k \ - \ 1}) \ > \ S_i^{k \ - \ 1}$ or $V(L_i^{k \ - \ 1}) \ > \ F_i^{k \ - \ 1} \ - \ \frac{L}{r_i}$.

$$V(t_2) \ - \ V(t_1) \ \leq \ F_i^{l \ + \ 1} \ - \ F_i^{k \ - \ 1} \ + \ \frac{L}{r_i} \tag{3.23}$$

Since flow f_i is continually backlogged since packet k

$$F_i^{l \ + \ 1} \ = \ F_i^k \ + \ \sum_{x=k \ + \ 1}^{l \ + \ 1} \frac{L}{r_i} \tag{3.24}$$

from case assumption $F_i^k \ = \ F_i^{k \ - \ 1} \ + \ \frac{L}{r_i}$ then

$$F_i^{l \ + \ 1} \ = \ F_i^{k \ - \ 1} \ + \ \sum_{x=k}^{l \ + \ 1} \frac{L}{r_i} \tag{3.25}$$

$$V(t_2) \ - \ V(t_1) \ < \ \sum_{x=k}^{l \ + \ 1} \frac{L}{r_i} \ + \ \frac{L}{r_i} \tag{3.26}$$

i.e

$$V(t_2) \ - \ V(t_1) \ < \ (l \ - \ k \ + \ 3) \ \frac{L}{r_i} \tag{3.27}$$

from this equation and equation (3.10)

$$s_i(t_1, t-2) \ > \ V(t_2) \ - \ V(t_1) \ - \ \frac{3 \ L}{r_i} \tag{3.28}$$

Case II: $F_i^{k \ - \ 1} \ < \ V(A_i^k)$

Since flow f_i is backlogged since t_1 and there were no packet departures in (t_1, L_i^k), hence $A_i^k \ \leq \ t_1$. Therefore

$$V(t_2) \ - \ V(t_1) \ \leq \ V(L_i^{l \ + \ 1}) \ - \ V(A_i^k) \tag{3.29}$$

$$V(t_2) \; - \; V(t_1) \; \leq \; F_i^{l \, + \, 1} \; - \; V(A_i^k) \tag{3.30}$$

$$V(t_2) \; - \; V(t_1) \; \leq \; F_i^k \; + \; \sum_{x=k \, + \, 1}^{l \, + \, 1} \frac{L}{r_i} \; - \; V(A_i^k) \tag{3.31}$$

from case assumption $F_i^k \; = \; V(A_i^k) \; + \; \frac{L}{r_i}$. Then,

$$V(t_2) \; - \; V(t_1) \; \leq \; \sum_{x=k}^{l \, + \, 1} \frac{l_i^x}{r_i} \tag{3.32}$$

i.e.,

$$V(t_2) \; - \; V(t_1) \; \leq \; (l \; - \; k \; + \; 2) \; \frac{L}{r_i} \tag{3.33}$$

$$s_i(t_1, t_2) \; > \; V(t_2) \; - \; V(t_1) \; - \; \frac{2 \; L}{r_i} \tag{3.34}$$

Since the first case is lower than the second case, we will use (5.30) and from (3.10)

$$\left| \frac{s_i(t_1, t_2)}{r_i} \; - \; \frac{s_j(t_1, t_2)}{r_j} \right| \; \leq \; \frac{6 \; L}{r_i} \tag{3.35}$$

■

The worst-case fairness index (WFI) [9, 2] measures the maximum time D_i that a packet arriving at an empty flow queue needs to wait before it receives its guaranteed service rate r_i. We have the following lemma.

Lemma 3.2.5 (Worst-Case Fairness)

$$D_i \;\; \leq \;\; \frac{Q_i}{r_i} \; + \; \frac{2L}{r_i} \tag{3.36}$$

where Q_i is the total backlog of flow i and L is the packet length.

Proof. If a packet from flow f_i arrives at time t_1 and creates a total backlog of Q_i. Let t_2 be the time when this Q_i is transmitted. Since this scheduler releases one packet to the FIFO queue when virtual time v is greater than the packets's start time, this means at least one packet is released every packet service time or counter time C_i then

$$D_i \;\; \leq \;\; \sum_{1}^{X} C_i \; + \; C_{i \; start} \; + \; C_{i \; end} \tag{3.37}$$

where X is the number of flow i packets, $C_{i\ start}$ and $C_{i\ end}$ is the residual counter times at the start and end of the time period (t_1, t_2) respectively. Since, each flow is served at least with r_i

$$C_i \leq \frac{L}{r_i} \tag{3.38}$$

From these two equations:

$$D_i \leq \sum_{1}^{X} \frac{L}{r_i} + \frac{L}{r_i} + \frac{L}{r_i} \tag{3.39}$$

$$D_i \leq \frac{Q_i}{r_i} + \frac{2L}{r_i} \tag{3.40}$$

■

3.2.4 TSFQ Intra-Tier and Inter-Tier Fairness Analysis

Previously we studied the fairness of a single intra-tier scheduler. Now, we will study the fairness of the whole scheduler which includes p intra-tier schedulers plus the inter-tier scheduler. For a single TSFQ intra-tier scheduler, we have found that the proportional fairness is $\frac{6\ L}{r_i}$ but as we mentioned before that the inter-tier scheduler selects the smallest finishing time among the p Head of line (HOL) packets. Considering this fact we have the following lemma for proportional fairness.

Lemma 3.2.6 (Proportional Fairness) *In any time period (t_1, t_2) during which flows i and j are backlogged, we have that:*

$$\left| \frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j} \right| \leq \frac{3\ L}{r_i} + \frac{3\ L}{r_j} \tag{3.41}$$

Proof. Each intra-tier scheduler has a bound in proportional fairness of $\frac{6\ L}{r_i}$ from Lemma 3.2.4. Using the same proof but for two flows with different rates r_i and r_j ■

Similarly, worst-case fairness is defined by the following lemma

Lemma 3.2.7 (Worst-Case Fairness)

$$D_i \leq \frac{Q_i}{r_i} + \frac{2\ L}{r_i} \tag{3.42}$$

Proof. From lemma 3.2.3 , TSFQ fixed-size packet is identical to WF^{2}Q+. Hence, TSFQ retains WF^{2}Q+ strong service bound property across all intra-tier schedulers and the proof of the previous lemma hold for p intra-tier schedulers. ■

Chapter 4

Tiered-Service Fair Queuing (TSFQ)- The Variable-Size Packet Case

In this chapter, we will remove the assumption we made in the previous chapter that all packets have a fixed size. As in the previous case, we consider the problem of scheduling flows within a given service tier, therefore we assume that all flows are assigned the same weight ϕ. In a network with variable-size packets, the statement of Lemma 3.2.1 is no longer true, since the second term in the right-hand side of (3.4) is not constant. Hence, in such a network, fair queuing schedulers in general require some form of packet sorting.

In the Internet, however, it is well known that certain packet sizes dominate [18, 19]. Specifically, the study in [18] found that packets of one of three common sizes make up more than 90% of all Internet traffic; the three common packet sizes identified in the study were 40, 576, and 1500 bytes, corresponding to TCP acknowledgments, the default IP datagram size, and maximum-size Ethernet frames, respectively. A more recent study [19] shows that (1) Internet traffic is mostly bimodal at 40 and 1500 bytes, (2) there is a shift away from 576 bytes due to the proliferation of Ethernet, and (3) a new mode is forming around 1300 bytes which the authors theorize is due to widespread use of VPNs. Similar studies, which can be found on CAIDA's web site (http://www.caida.org), confirm that the length of the vast majority of Internet packets takes one of a small number of constant values. In the remainder of this section we show how we can exploit these facts regarding the Internet packet length distribution to modify the intra-tier TSFQ scheduler we presented in the previous section so that it can handle Internet traffic efficiently, i.e., by performing a number of packet sorting operations independently of the number of flows.

4.1 Queue Structure and Operation

Instead of maintaining a single FIFO queue, as is the case for fixed-size packets shown in Figure 3.2, the intra-tier scheduler for variable packet size networks maintains a small number k of queues. The queue structure of this scheduler is illustrated in Figure 4.1 for the trimodal packet length distribution reported in [18]; the queue structure can be modified in a straightforward manner to reflect any similar distribution. In this case, the scheduler maintains $k = 7$ queues. Three of the queues are dedicated to packets of a common size, i.e., 40, 576, and 1500 bytes, respectively, which define the three modes of the distribution in [18]. The other four queues are for packets of sizes other than the common values; as seen in Figure 4.1, there is one queue for packets of size less than 40 bytes, one for packets of size 41-575 bytes, one for packets of size 577-1499 bytes, and one for packets of size greater than 1500 bytes.

4.2 Intra-Tier Scheduler: The Variable-Size Packet Case

The operation of the intra-tier scheduler is very similar to the one we described in Section 3.2.1, with only one difference. In particular, the actions taken at packet arrival and departure events are identical to those in the fixed-packet case listed in Section 3.2.1. The only difference is in the actions taken at instants when a flow becomes eligible for service:

- *A flow becomes eligible for service.* When a flow i becomes eligible at time t, then the token κ_i corresponding to this flow is inserted into the queue corresponding to the size of the packet at the head of the queue of flow i.

Similar to the fixed-packet case, token κ_i is in one of the intra-tier scheduler's queues if and only if flow i is eligible for service.

Since each of the p inter-tier schedulers maintains k distinct queues, the inter-tier scheduler selects the flow to serve next as the one with the smallest virtual finish time

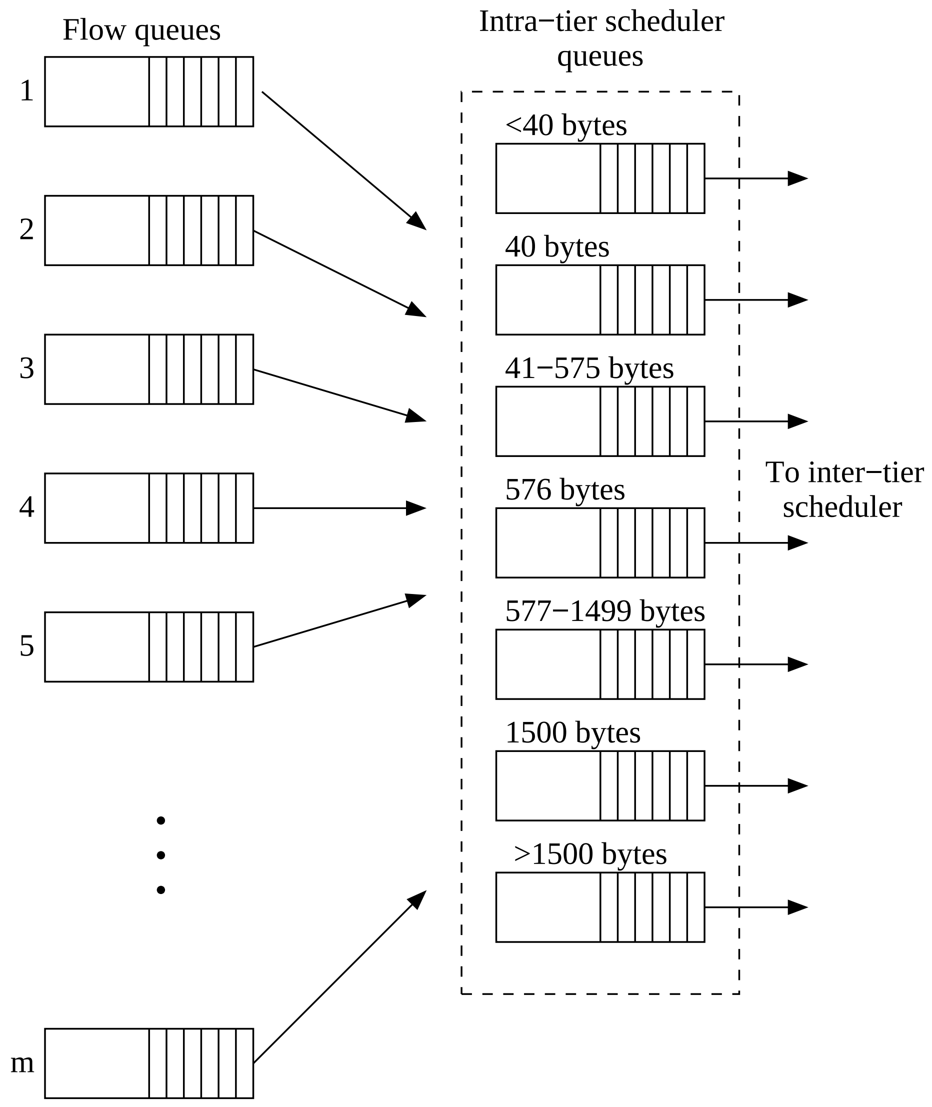


Figure 4.1: Queue structure of the intra-tier scheduler for Internet packet traffic

among the pk candidate flows whose tokens are at the head of the pk queues. Since both p and k are small integers and their values are constant for a given system, this operation of the inter-tier scheduler takes constant time, as in the fixed-size packet case.

4.2.1 Packet Sorting Operations

Note that Lemma 3.2.1 holds true for packets of a common size. Hence, the queues dedicated to these packets operate in a FIFO manner, and packets are simply inserted at the tail of these queues. Since packets of a common size make up more than 90% of Internet traffic [18], no sorting operations are necessary for the vast majority of packets. On the other hand, queues dedicated to packets of sizes between the common values must be sorted appropriately at the time of a packet insertion. These sorting operations take place infrequently (less than 10% of the time), and involve relatively short queues (since less than 10% of the packets are spread over several such queues at p different service levels). Moreover, the time complexity of the sorting operations is *independent* of the number m of flows in the given service tier, and is a function only of the network load and the ratio of packets with a non-common size.

We have the following results.

Lemma 4.2.1 *The TSFQ scheduler for variable packet sizes, consisting of p intra-tier schedulers as in Figure 4.1 and one inter-tier scheduler, is identical to WF^{2}Q+.*

Proof. The proof of Lemmas 3.2.2 and 3.2.3 also holds in this case, hence the scheduler is equivalent to WF^{2}Q+. ■

Finally, we note that, although consecutive packets of the same flow i may be inserted into different queues in Figure 4.1, they will always be transmitted in order: not only does the second packet have a larger virtual finish time than the first one, but since there is exactly one token for each flow, the second packet cannot be considered for service until the first one has departed from the scheduler.

4.2.2 Elimination of Packet Sorting Operations

The operation of the intra-tier scheduler may be further simplified by eliminating packet sorting even for queues holding packets of sizes between the common values. Doing so may cause some packets to be served in incorrect order of virtual finish time, hence introducing a small degree of unfairness. However, the overall impact is likely to be small. Indeed, observe that packets of a non-common size represent only a small fraction of the overall traffic seen by the server, and are distributed over a number of different queues across p service tiers. Consequently, the arrival rate to each of these queues is likely to be low, especially under typical operating conditions when the load offered to the server is not too high. Now note that, since all flows within a service tier have the same weight ϕ in expression (3.4), the order of packets in such a queue will depend on the relative values of their virtual start time and length. Therefore, even when a small packet arrives to find larger packets in the queue (i.e., packets with a larger value for the second term in the right-hand side of (3.4)), the elapsed time since the previous arrival (which affects the first term of (3.4)) may be sufficiently large so that the queue remains sorted.

This intuition is further supported by the coarse manner in which the leap forward virtual clock [21] algorithm computes timestamps, and the mechanism employed by the bin sort fair queuing (BSFQ) discipline [1] to sort packets. The results in [21, 1] indicate that approximate sorting can be as good as exact sorting; moreover, in the case of our TSFQ scheduler, approximate sorting is limited to a small fraction of all packets.

We emphasize that the queue structure shown in Figure 4.1 is for illustration purposes only and is simply meant to convey the idea underlying the structure of the scheduler for Internet packet traffic; we do not imply that routers have to be configured in exactly this manner. Network operators may configure this queue structure to reflect the specific packet distribution observed in their networks, and update it over time as traffic conditions evolve. Similarly, they may optimize the number of service tiers and the flow weights associated with them (e.g., using the techniques presented in earlier chapters) by taking into account the prevailing user demands. Therefore, this framework of fair queuing schedulers for tiered-service networks is quite flexible. Network providers may adapt the specific elements of the framework to differentiate their offerings, and to provide users with a menu of customized services.

4.3 Fairness Analysis

In this section, we will study the fairness of the TSFQ scheduler for variable-size packet case. First, we need to notice that each intra-tier scheduler in the variable-size packet case is only an array of fixed packet size intra-tier scheduler. Each element of this array represents a different packet size. From this, we can use the same proofs and results from the previous chapter to analyze fairness of variable-size packet case.

The TSFQ for the variable-size packet case consists of p intra-tier scheduler. Each one of these intra-tier schedulers consists of k queues as shown in Figure 4.1. The inter-tier scheduler has to choose the smallest finishing times among these pk queues. In the previous chapter, we found that the proportional fairness for TSFQ intra-tier scheduler is $\frac{6\ L}{r_i}$. The following lemma calculates the proportional fairness for the TSFQ variable-size packet case.

Lemma 4.3.1 (Proportional Fairness) *In any time period (t_1, t_2) during which flows i and j are backlogged, we have that:*

$$\left| \frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j} \right| \leq \frac{3\ L_i}{r_i} + \frac{3\ L_j}{r_j} \tag{4.1}$$

Proof. Each intra-tier scheduler has a bound in proportional fairness of $\frac{6\ L}{r_i}$ from Lemma 3.2.4. Using the same proof but for two flows with different rates and packet-sizes (r_i, L_i) and (r_j, L_j). ■

Similarly, worst-case fairness is defined by the following lemma.

Lemma 4.3.2 (Worst-Case Fairness)

$$D_i \leq \frac{Q_i}{r_i} + \frac{2\ L_{max}}{r_i} \tag{4.2}$$

Proof. From lemma 4.2.1 , TSFQ variable-size packet is identical to WF^{2}Q+. Hence, TSFQ retains WF^{2}Q+ strong service bound property across all intra-tier schedulers and the proof of the previous lemma hold for p intra-tier schedulers. Taking L_{max} as the worst-case among the k packet-size queues. ■

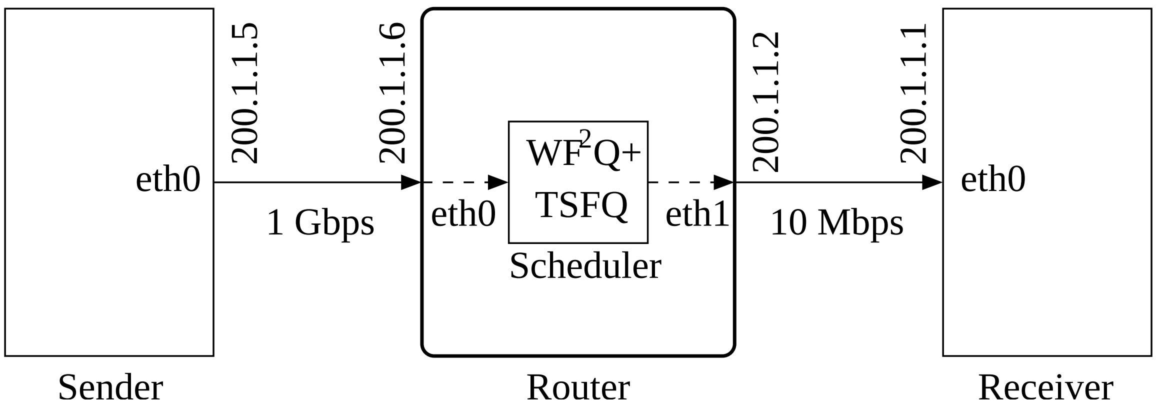


Figure 4.2: Testbed setup

4.4 Experimental Evaluation of TSFQ

We have developed implementations of the TSFQ scheduler for the *ns-2* network simulator and in the Linux kernel. The details of the *ns-2* implementation are reported in [22], along with a comprehensive set of simulation experiments that validate the operation of TSFQ. In this section we present network experiments with the Linux kernel implementation which is fully described in [23].

The TSFQ scheduler was implemented as a Linux kernel loadable module. The Linux kernel version 2.6.26.2 [24] was used, the latest kernel available at the time of the implementation in early 2008. The WF^{2}Q+ discipline [2] was also implemented as a separate loadable module for comparison purposes, since a Linux kernel implementation of the WF^{2}Q+ scheduler did not exist at the time.

4.4.1 Testbed and Experimental Setup

The experiments were carried out using a testbed consisting of three Linux machines connected as shown in Figure 4.2. The leftmost machine acts as the “sender” of UDP traffic that is destined to the rightmost machine, the “receiver.” The middle machine is configured as a “router” that receives packet traffic from the sender and forwards it to the receiver. The Ethernet link from the sender to the router is configured to run at 1 Gbps, while the link from the router to the receiver is configured to run at 10 Mbps. Consequently, the latter link becomes the bottleneck, causing the queues at the router to build up.

UDP traffic at the sender is generated by multiple simultaneous flows, each transmitting to a different destination port on the receiver. The router implements the TSFQ and WF^{2}Q+ disciplines to schedule packets received from the sender for transmission on the outgoing 10 Mbps link. It also employs per-flow queuing, assigning a separate FIFO queue to each UDP flow. The router uses the port information carried by the packets to determine the flow to which they belong and insert them into the appropriate queue. The TSFQ and WF^{2}Q+ schedulers use pre-configured weights to serve the queues of the various flows.

The UDP flows at the sender continuously transmit packets to the receiver without any form of flow control. Packet sizes L are randomly generated from the following discrete distribution:

$$Pr[L = x] \;=\; \begin{cases} 0.3, & x = 40 \\ 0.3, & x = 1200 \\ 0.3, & x = 1500 \\ 0.1, & 1 \leq x \leq 39, 41 \leq x \leq 1199, 1201 \leq x \leq 1499 \end{cases} \tag{4.3}$$

This distribution generates traffic dominated by a small number (in this case, three) of packet sizes, and is similar to the packet size distributions observed in [18, 19]. Consequently, the intra-tier schedulers of TSFQ are configured with six queues, similar to the structure shown in Figure 4.1 (the seventh queue of Figure 4.1 for packets of size greater than 1500 bytes is not used here, as no such packets are generated).

A number of experiments were carried out to investigate the behavior of the schedulers under three scenarios:

- *Scenario I.* Several flows of different weights are started at the same time. After the system reaches steady state, the flows are terminated one by one. This scenario explores how the schedulers allocate excess bandwidth to the remaining flows.

- *Scenario II.* A small number of flows are started at the same time. After the system reaches steady state, new flows of different weights are started. Once the system reaches steady state again, the newly introduced flows are terminated. The start and termination instants of the new flows are spread over time. These experiments are used to investigate the impact of new flows on the bandwidth share of existing ones, as well as the allocation of excess bandwidth.

- *Scenario III.* Many flows spanning a small number of service levels are run for a long time. This scenario is used to evaluate the fairness of each scheduling discipline. Specifically, we use Jain's fairness index [25] to compare the WF^{2}Q+ and TSFQ schedulers. In a system with n competing flows and flow i having throughput share $f_i, i = 1, \ldots, n$, Jain's fairness index (FI) is defined as:

$$FI = \frac{\left(\sum_{i=1}^{n} f_i\right)^2}{n \sum_{i=1}^{n} f_i^2}, \tag{4.4}$$

  such that a value of 1 represents perfect fairness with all flows receiving an equal share ($= 1/n$) of the available bandwidth.

4.4.2 Performance Results

In this section we present a set of illustrative experiments for the three different scenarios described above.

Scenario I: Allocation of Excess Bandwidth

Figures 4.3 and 4.4 present the results of an experiment to investigate the relative behavior of the WF^{2}Q+ and TSFQ schedulers in allocating excess bandwidth. This experiment involves four flows: flows 1 and 2 each have weight 0.15, while flows 3 and 4 each are assigned weight 0.35. For this experiment, the TSFQ scheduler was configured with $p = 2$ tiers, one with weight 0.15 and the other with weight 0.35; hence, flows 1 and 2 were assigned to the first tier, and flows 3 and 4 were assigned to the second tier. All four flows start transmission simultaneously at time $t = 0$, and are terminated one-by-one, in reverse order of their index, at 10-second intervals.

Figures 4.3 and 4.4 plot the throughput of each flow (in Mbps) as a function of time for the WF^{2}Q+ and TSFQ schedulers, respectively. Recall that the bottleneck link in the experimental setup was set to 10 Mbps, and this latter value represents the bandwidth that is shared among the four flows. We observe that during the first 10 seconds of the experiment when all four flows are active, both schedulers allocate the available bandwidth in proportion to the flow weights, such that flows 1 and 2 (respectively, flows 3 and 4) capture approximately 15% (respectively, 35%) of the total bandwidth each. When flow 4 terminates, the bandwidth share of each of the three flows that remain active increases proportionally to its weight. In particular, flow 3 with the highest weight (.35) captures most of the bandwidth that becomes available, while flows 1 and 2 of the same but lower weight (0.15) capture an equal share of the excess bandwidth. The same behavior is observed at the time the other flows are terminated. Importantly, the throughput curves of a given flow are comparable across the two figures, implying that the TSFQ and WF^{2}Q+ schedulers perform similarly in terms of allocating bandwidth to flows in proportion to their weights.

Scenario II: Impact of New Flows

In order to demonstrate the performance of the two schedulers when flows both arrive and depart, we run an experiment with the same flows as in Scenario I, i.e., two flows of weight .35 and two of weight .15. In this case, the flows of lower weight (flows 1 and 2) both become active at time $t = 0$. At time $t = 10$ seconds (respectively, $t = 20$ seconds) flow 3 (respectively, flow 4) of higher weight becomes active. All four flows remain active until time $t = 30$ seconds, at which time flow 3 departs, followed by flow 4 at time $t = 40$ seconds.

The results of this experiment are shown in Figures 4.5 and 4.6, which again plot the time-varying throughput of each flow under the WF^{2}Q+ and TSFQ scheduler, respectively. During the first ten seconds of the experiment, the two active flows receive an equal share of the available bandwidth despite their low weights, as expected. As the other two flows are introduced, the bandwidth share of existing flows is reduced accordingly; on the other hand, the bandwidth share of active flows increases as flows depart (i.e., are terminated). Overall, we make three important observations: (1) at any point in time the available bandwidth is shared among active flows in proportion to their weights; (2) as flows arrive or depart, the bandwidth share of all the flows in the system quickly reaches a new equilibrium; and (3) there is good agreement in the behavior of the two schedulers.

Scenario III: Long-Term Fairness

The first experiment of this section investigates qualitatively (i.e., graphically) the long-term fairness of the WF^{2}Q+ and TSFQ schedulers, and involves 32 flows that all start at time $t = 0$ and remain active throughout the duration of the experiment. Two of the flows have a weight of .35, ten flows have a weight equal to 0.05, and the remaining twenty

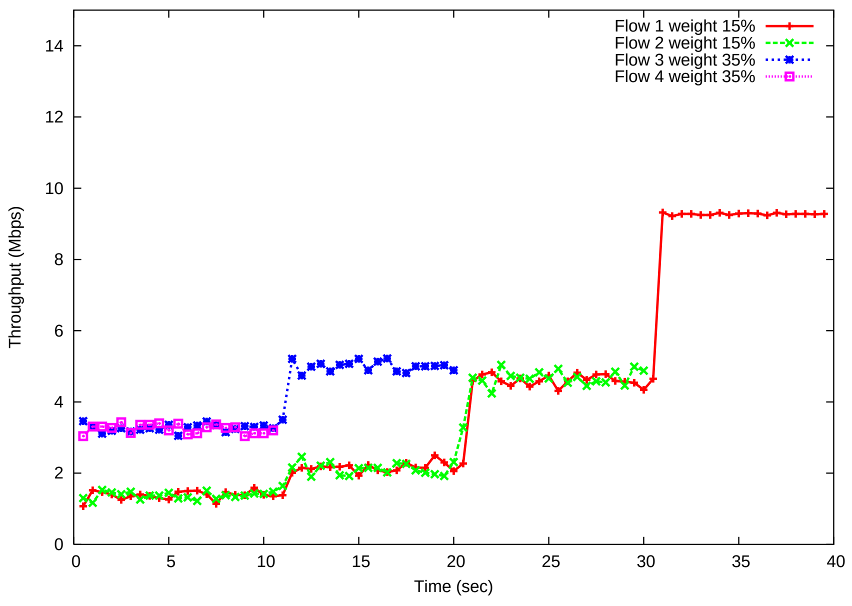


Figure 4.3: Scenario I, four flows, WF^{2}Q+ scheduler

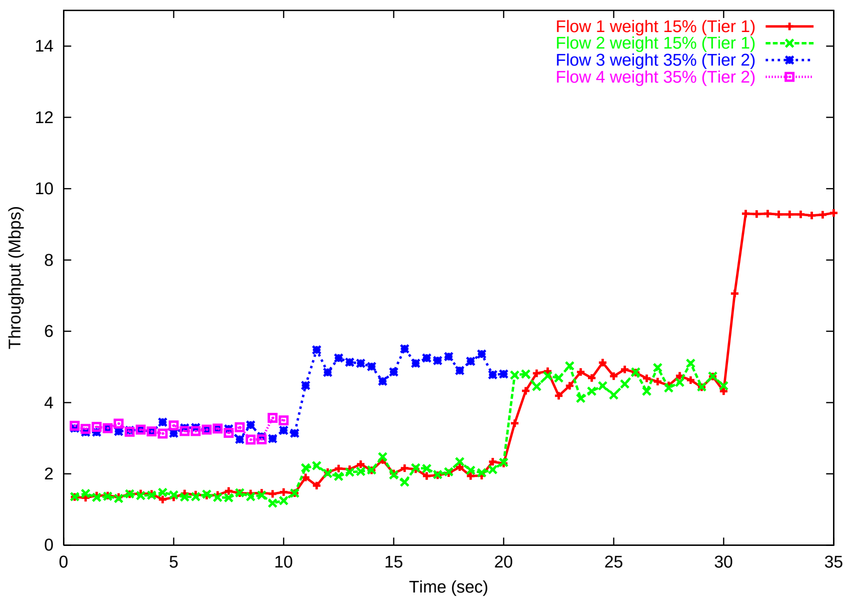


Figure 4.4: Scenario I, four flows, TSFQ scheduler

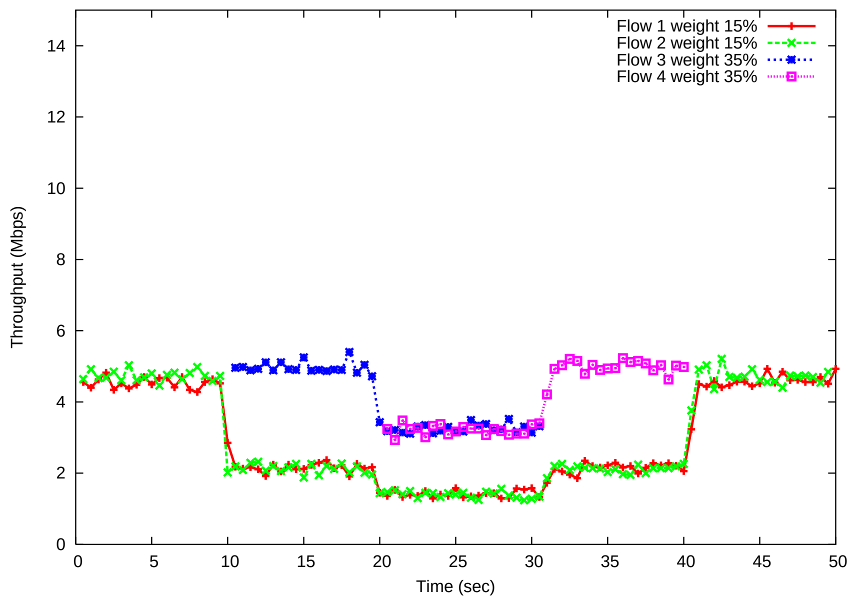


Figure 4.5: Scenario II, four flows, WF^{2}Q+ scheduler

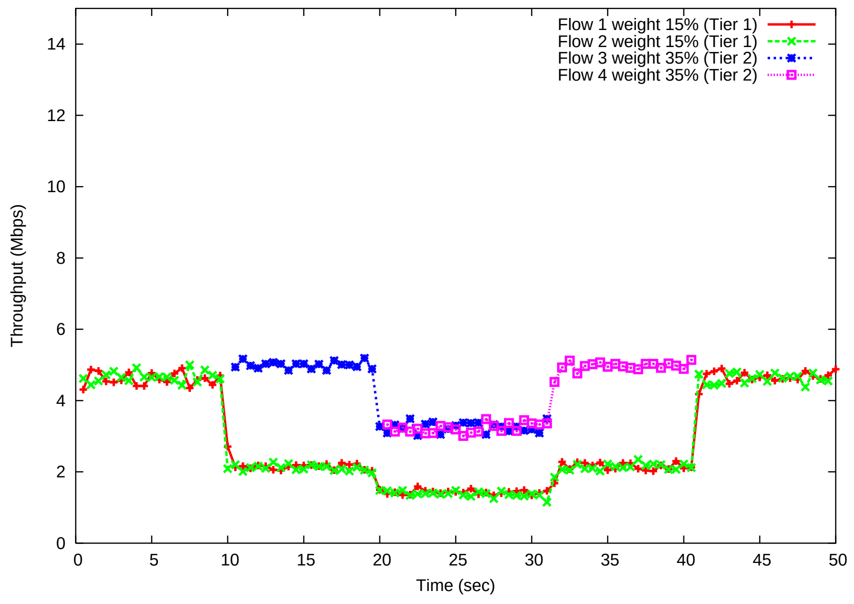


Figure 4.6: Scenario II, four flows, TSFQ scheduler

flows have a weight of 0.01. Hence, for this experiment, the TSFQ scheduler was configured with $p = 3$ tiers with weights of 0.35, 0.05, and 0.01, respectively, and the thirty-two flows were assigned to the appropriate tier according to their individual weights.

Figures 4.7 and 4.8 plot the throughput of the thirty-two flows as a function of time for the WF^{2}Q+ and TSFQ schedulers, respectively. In both figures, the flows are clearly separated into three groups, each corresponding to three TSFQ tiers, with flows within each group receiving a share of bandwidth in line with their weight. Although the throughput of the various flows shows more short-term variations under the TSFQ scheduler, the overall behavior is similar in the two figures. In order to quantify the long-term fairness of the two schedulers, we computed Jain's fairness index from expression (4.4), using the long-term throughput of the thirty-two flows, and normalizing these values by the corresponding flow weight. The fairness index values are plotted in Figure 4.9 as a function of time. The fairness index is about 10% higher under the WF^{2}Q+ scheduler, reflecting the lower throughput variations in Figure 4.7. Nevertheless, the curves of both schedulers are relatively stable across the duration of the experiment, indicating that the two schedulers have similar fairness characteristics.

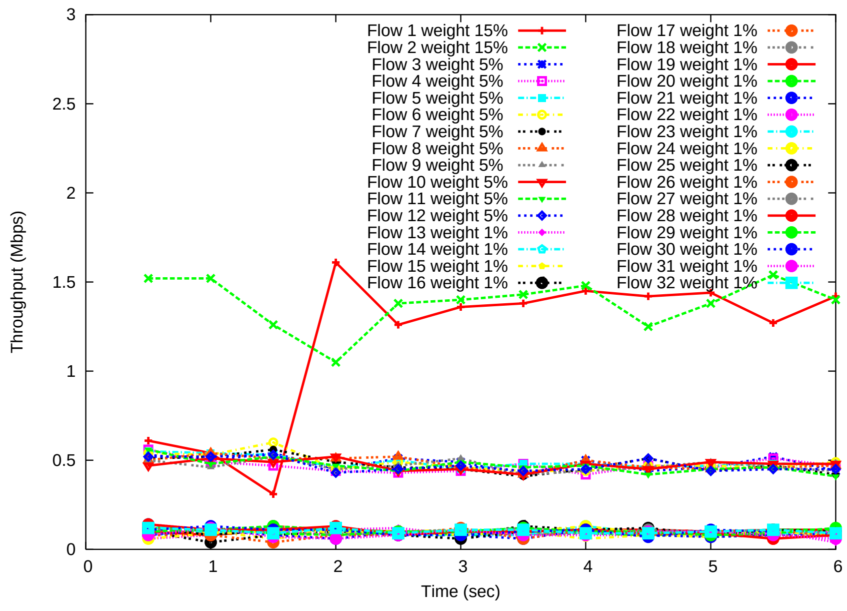


Figure 4.7: Scenario III, thirty two flows, WF^{2}Q+ scheduler

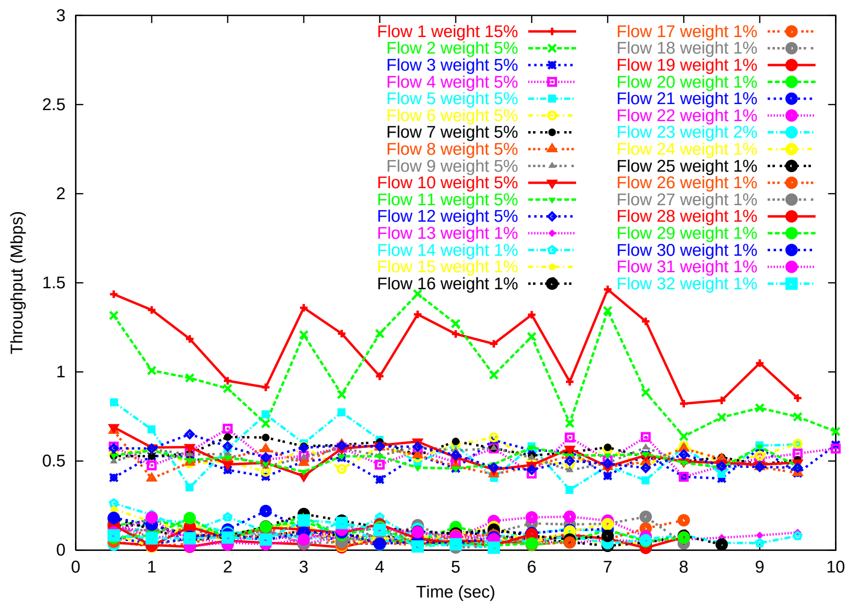


Figure 4.8: Scenario III, thirty two flows, TSFQ scheduler

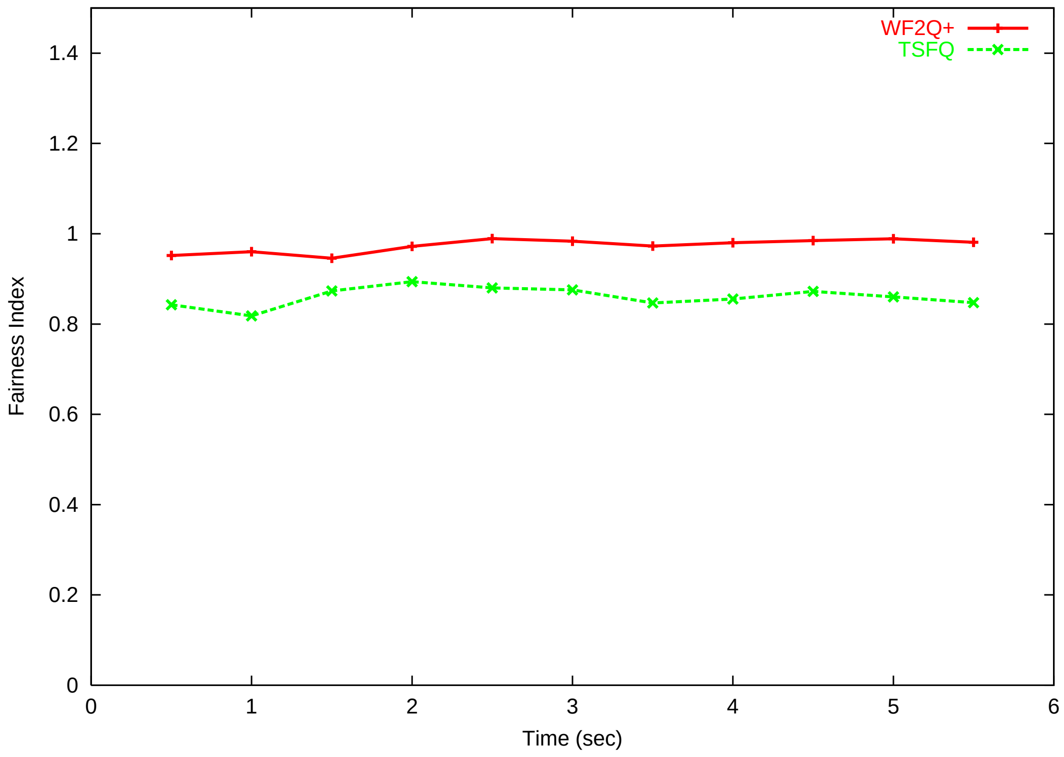


Figure 4.9: Scenario III, thirty-two flows, fairness index

Chapter 5

The Worst-Case Bin Sort Queuing (WBSQ)

In this chapter we present a new noble packet scheduler called the Worst-Case Bin Sort Queuing (WBSQ). Similar to the TSFQ scheduler, WBSQ uses quantization to achieve good fairness properties while maintaining low complexity. However, the quantization in WBSQ is used in a different dimension than the one used in TSFQ. As we discussed before, TSFQ quantize the service offered by the scheduler into p service tiers. WBSQ on the other hand quantizes virtual time into buckets of time or bins to eliminate the need for sorting. In addition, WBSQ utilizes this type of quantization in a special queue structure to achieve constant worst-case fairness.

The virtual time stamp calculation in WBSQ is similar to the ones used by most of the WFQ schemes. Suppose that the k^{th} packet of a flow i arrives at time t, has weight of ϕ_i, and has length L_i^k. Let S_i^k and F_i^k denote the virtual times at which this packet begins and completes service, respectively. Letting $F_i^0 = 0$ for all flows i, we have [6]:

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\} \tag{5.1}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i} \tag{5.2}$$

where a_i^k is the arrival of packet k of flow i. For each arriving packet, its finish time is computed according to equations (5.1) and (5.2). The scheduler serves packets in order of their finish times for those which have their start time already started in GPS at time t

i.e when system virtual time $V(t)$ passes their starting time S. For system virtual time $V(t)$ calculation, WBSQ uses the simplified calculation proposed by WF^{2}Q+ as in equation (5.3).

$$v(t+\tau) = \max(v(t)+\tau, \min(S_i)) \tag{5.3}$$

for each flow $f_i, (i = l, ...N)$.

WBSQ scheduler employs the concept of bin sorting to eliminate the need for sorting where the virtual time space is divided into equal intervals or bins. Each bin represents an interval of virtual time between t and $t+\delta$ where δ is the bin width. Packets are assigned virtual time stamps and inserted into their corresponding bins. The queuing order within a bin is First In First Out, FIFO. WBSQ starts serving from the first bin and gets the first eligible packet in that bin. A packet is eligible when the system virtual time $V(t)$ is greater than its start time $S_i^k(t)$.

Notice that there are two major differences between WBSQ and Bin Sort Fair queuing (BSFQ) as proposed in [1]. First, WBSQ uses WF^{2}Q+ system virtual time $V(t)$ calculation (5.3) whereas BSFQ use the current bin finishing time value. The second difference is that WBSQ serves only eligible packets i.e when system virtual time $V(t)$ is greater than packets starting time. BSFQ on the other hand does not have a mechanism to check eligibly. Thats the reason why WBSQ has constant worst-case fairness whereas BSFQ suffers from $O(N)$ worst-case fairness. However, both schedulers serve packets in FIFO order within a single bin. That's why bin width δ is an important design parameter as we will see when we talk about fairness properties of WBSQ.

In order to keep constant implementation complexity, WBSQ needs to avoid using the minimum operation that exists in the calculation of system virtual time (3.6). In fact, WBSQ needs to emulate two minimum operations to keep constant complexity. The first minimum operation is needed to select the packet with minimum finishing time F equation (5.2) i.e sorting. The second minimum operation is needed to calculate virtual system time $V(t)$ according to WF^{2}Q+ equation (5.3). These two operations normally cost $log(N)$ where N is the number of flows. In order to maintain constant time implementation, WBSQ uses a new idea that utilizes the bin sort concept, circular buffers and linked lists to represent and sort the finishing time and starting time of the flows.

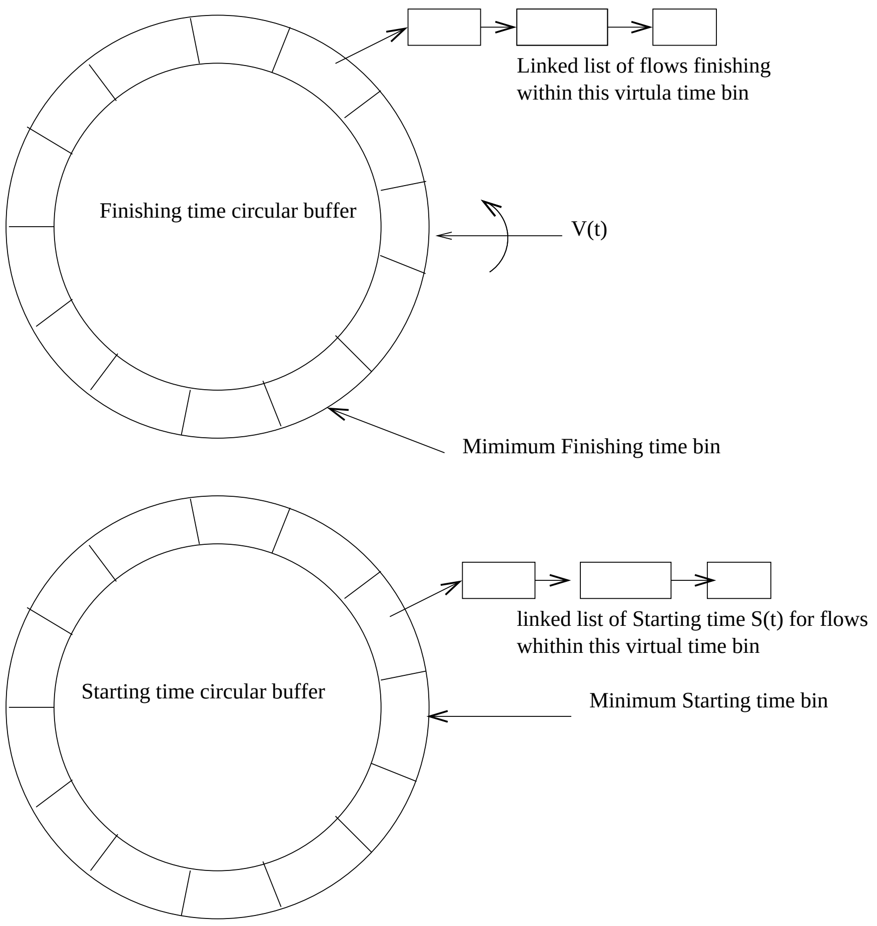

Linked list of flows finishing
within this virtula time bin
Finishing time circular buffer
V(t)
Mimimum Finishing time bin
linked list of Starting time S(t) for flows
whithin this virtual time bin
Starting time circular buffer
Minimum Starting time bin

Figure 5.1: WBSQ scheduler.

5.1 Queue Structure and Operation

As shown in Figure 5.1, WBSQ uses two circular buffers that are divided into virtual time bins. One buffer represents the finishing time and the second buffer represents the starting time. Both buffers are organized into N^{bin} bins where N^{bin} is the number of bins in the circular buffer. N^{bin} is a system design parameter and must be set to a value that is equal to, or greater than a certain threshold to allow scheduler to use all available buffers. Each bin is implicitly labeled with a virtual time interval and each interval has length δ. The parameter δ is another system design parameter of the scheduler and its value has a significant impact on performance. The intervals of the bins are disjointed and their union spans a continuous range of the virtual time space that represents finishing and starting times.

Each bin in these two circular buffers maintains a pointer to a linked list. In the finishing time circular buffer, each bin points to a linked list of flows that have finishing times corresponding to the bin virtual time interval. Similarly, in the starting time circular buffer, each bin points to a linked list of flows that have starting times corresponding to the bin virtual time interval. In each circular buffer, there is a moving starting pointer that points to the bin with the minimum value. In the finishing time circular buffer case, the pointer P_{min_f} points to the bin with time interval $(t^{min_{bin}}, t^{min_{bin}} + \delta)$ that holds the minimum finishing time. Similarly, the pointer P_{min_s} points to the bin with time interval $(t^{min_{bin}}, t^{min_{bin}} + \delta)$ that holds the minimum starting time of all flows. Notice that the N^{bin} bins in each circular buffer will be used to represent virtual time interval from $(t^{min_{bin}}, t^{min_{bin}} + (N^{bin} - 1) * \delta)$ where N^{bin} is the number of bins in the circular buffer. Since the finishing time is an increasing function of time, the minimum finishing time is always moving forward. This makes the represented virtual time range $(t^{min_{bin}}, t^{min_{bin}} + (N^{bin} - 1) * \delta)$ a forward sliding window. There is another moving pointer in the finishing time circular buffer called P_v that points to the bin which has the current system virtual time $V(t)$ value as shown in Figure 5.1.

When a packet becomes the head of the flow, two entries will be assigned to it. One in the finishing time linked list corresponding to its finishing time and one in the starting time linked list corresponding to its starting time. The bin number P_f corresponding to the packet finishing time $F_i(t)$ will be calculated relative to the bin P_v represent the current $V(t)$ as follows.

$$P_f = P_v + \left\lfloor \frac{F_i(t) - V(t)}{\delta} \right\rfloor \tag{5.4}$$

If this value is equal to P_v, flow finishing time will be represented in the current $V(t)$ bin P_v, and otherwise, it is represented in the P_f bin relative to the P_v. If P_f is greater than N_m, the packet is discarded. In the same manner, the starting time entry will be inserted into the starting time circular buffer using a similar equation but with reference to the current minimum starting time min_S.

$$P_S = P_{min_S} + \left\lfloor \frac{S_i(t) - min_S}{\delta} \right\rfloor \tag{5.5}$$

where P_{min_S} is the bin holding the minimum starting time value of all active flows.

When choosing the next packet to serve, WBSQ always starts from the starting point reference P_{min_F} bin and serves the first flow in the FIFO linked list that has a starting time $S_i(t)$ less than or equal to $V(t)$. If no flow was found with this criteria at a certain bin, WBSQ will move to the next bin and so on. Notice here that even in the case of serving a packet from a bin that might be different than the starting point P_{min_F} reference, WBSQ will start from the same start point P_{min_F} in the next service cycle. However, there is some unfairness introduced in packet selection due to using FIFO within a bin. As we will show in the next section, this unfairness is proportional to bin width δ.

All pointers of WBSQ's circular buffers are continuously updated at arrivals and departures. Such that, whenever the $V(t)$ value changes, the pointer P_v will be advanced to the bin that holds the new value of $V(t)$. Similarly, P_{min_S} and P_{min_F} also shift according to the new value of S_{min} and F_{min} respectively. Also, when the next packet of a flow comes to the head of the line, the old entries for previous packet starting time and finishing time are removed and the new values are inserted.

However, when the starting time entry for the flow that has the minimum starting time min_S are removed, the starting pointer P_{min_S} will choose the next minimum starting time. The new S_{min} will be chosen from the FIFO linked list in the current P_{min_S}. If no more entries exist in that bin, WBSQ will choose from the next bin and so on. Note here that there is a possibility that the new selected S_{min} is not the actual minimum starting time. This error is introduced due to lack of sorting within a single bin. This could lead to having S_{min} with δ greater than the actual minimum starting time. We will discuss the effect of this error in the scheduler analysis section. If the entry for the minimum finishing time were removed, the next finishing time entry with same or next bin is chosen for P_{minF} pointer. On the other hand, if a packet with a finishing time less than the existing F_{min} were inserted, P_{minF} pointer will point to this new packet bin.

All of these operations can be accomplished at constant time, and by updating the two circular buffers and their pointers, we can get the minimum finishing time and minimum starting time without expensive minimum operations. Hence, we can keep constant time complexity for packet sorting and $V(t)$ calculations. In the next section, we will prove that WBSQ scheduler provides an end-to-end delay and a fairness guarantee. Also we will discuss its worst-case fairness.

5.2 Scheduler Analysis

5.2.1 Accuracy of Virtual Time Calculation

The time stamps calculations in WBSQ is not performed using the exact virtual time equation (5.3) that W^{2}FQ+ uses. Instead, these calculations are done through approximation methods to avoid the $log(N)$ complexity of minimum operation. However, this approximation introduces some error in the calculations. Errors in calculations are introduced from two main sources. The first is introduced when selecting the next minimum starting time S_{min} from a bin in the starting time circular buffer. Since there is no sorting within a bin, this minimum could be δ greater than the actual minimum starting time. This could advance $V(t)$ by δ ahead as in equation (5.3). As a result, errors are introduced when packets time stamps are calculated using the new value of $V(t)$ in (5.1) and (5.2). This could advance finishing and starting time by δ for the new traffic that arrives after this error in $V(t)$ is introduced. However, in the worst-cast, this error may give an advantage for up to $\delta\ r$ worth of packets only.

The second source of error comes from selecting the next packet to service within a bin in the finishing time bin array. Since packets are not sorted within a bin, this can lead to selecting a packet other than the packet with the minimum finishing time. However, WBSQ would not service a packet if it not eligible for service. This condition limits the worst-case difference to $\Sigma_{allflows}\ \frac{L_{max}}{r_i}$ or $\frac{L_{max}}{r}$.

Lemma 5.2.1 *The Maximum total service W_δ that all flows f_j may receive within a virtual time bin $[t, t\ +\ \delta]$ before serving an eligible flow f_i is bound by:*

$$W_\delta \quad \leq \quad r\ \delta\ +\ L_{max} \tag{5.6}$$

where $j = 1, 2, ..., N$ where $j\ \neq\ i$

Proof.

From our discussion earlier in this section, the worst-case error in finishing time calculation could lead up to δ in advantage to old packets. Let assume that flow i actual finishing time $F_i^{actual}(t)$ was increased by δ due to the introduced virtual time error. If flow j has finishing time $F_j(t)\ >\ F_i^{actual}(t)$ but due the introduced error it becomes

$F_j(t)\ >\ F_i^{actual}(t)\ +\ \delta$

The only packets that can be served from flow j are those that have a finishing time

$F_j(t) \ < \ F_i^{actual}(t) \ + \ \delta.$

Since flow j finishing time increase is proportional to r_j, the maximum service that flow j could have before flow i is $r_j \ \delta$.

Hence, the total service that all flows can have before flow i is

$$\Sigma_{allflows} \ r_j \ \delta \ = \ r \ \delta \tag{5.7}$$

Now let us consider the case when $F_i^F(t) \ \leq \ F_j^F(t)$ where P_i^F is the first packet of flow i in virtual time $[t, t+\delta]$. Let us assume that the order of packets was reversed due to the lack of sorting within a bin. Since WBSQ serves packet in FIFO within a bin, flow j packet will be considered first. However, only those packets from flow j who started service in GPS $V(t) \ \geq \ S_j^m(t)$ could be served before serving P_i^F packet. Let consider worst-case scenario when flow j restarted being active with packet P_j^F

$S_j^F(t) = \max(V(t), F_j^{F-1}(t)) = V(t)$

Since $F_i^F \ \leq \ F_j^F$

$S_i^F(t) \ - \ \frac{L_i^F}{r_i} \ \leq \ V(t) - \frac{L_j^F}{r_i}$

$V(t) \ \geq \ S_i^F(t) \ + \ \frac{L_j^F}{r_i} \ - \ \frac{L_i^F}{r_i}$

Now, if packet P_j^F is served before P_i^F, the new starting time for flow j is

$S_j^{F+1}(t) = F_j^F(t) \ = \ V(t) \ + \ \frac{L_j^F}{r_i}$

Even if packet P_j^{F+1} has been inserted into the bin's linked list before packet P_i^F, the following condition has to be true for updated $\grave{V}(t)$

$\grave{V}(t) \ \geq \ S_j^{F+1}(t) \ \geq \ V(t) \ + \ \frac{L_j^F}{r_i}$

Since we are using WF^{2}Q+ virtual time calculations. This means that WBSQ virtual time calculations meet the GBT property in terms of the virtual system time and the virtual start time. This means:

$S_i(t) \ - \ \frac{L_{max}}{r_i} \ \leq \ V(t) \ \leq \ S_i(t) \ + \ \frac{L_{max}}{r_i}$

Hence, the maximum value that $V(t)$ can go in advance is $\frac{L_{max}}{r_i}$. This means that the total service that all flows can get before serving packet P_i^F is L_{max}

From this equation and (5.7), the Maximum total service W_δ that all flows f_j may

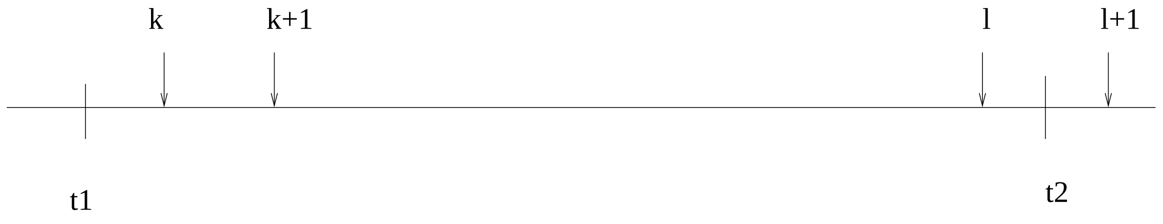


Figure 5.2: Packets in time interval (t_1, t_2).

receive within a virtual time bin $[t, t + \delta]$ before serving an eligible flow f_i is bound by:

$$W_\delta \leq r\,\delta + L_{max} \tag{5.8}$$

■

5.2.2 Proportional Fairness

The fairness measure of Golestani [8] essentially requires that the difference between the normalized service received by any two backlogged flows f_i and f_j, over any time period (t_1, t_2), be bounded by a small constant. This section analyzes the performance of WBSQ scheduler with respect to the fairness measure of Golestani.

Theorem 5.2.1 *(Golestani fairness). In any time period (t_1, t_2) during which flows f_i and f_j are backlogged,*

$$\left| \frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j} \right| \leq 3\left(\frac{l_i^{max}}{r_i} + \frac{l_j^{max}}{r_j} + \delta\right) \tag{5.9}$$

where $S_i(t_1, t_2)$ is the amount of data sent for flow i during the time period (t_1, t_2)

Proof.

Let us assume that two flows f_i and f_j were continuously backlogged at time interval (t_1, t_2). Let P_k^i represent the first packet that depart in this time interval from flow f_i at L_i^k and assume P_i^l be the last packet depart in this time interval from flow f_i at time L_i^l as shown in Figure 5.2.

Since flow f_i is continually backlogged in interval (t_1, t_2), the starting time

$$S_i^h = max(F_i^{h-1}, V(A_i^h)) = F_i^{h-1} \tag{5.10}$$

for all $h = k+1, ..., l+1$.

from this we can say that

$$\sum_{x=k+1}^{l} l_i^x \;\leq\; S_i(t_1, t_2) \;\leq\; \sum_{x=k}^{l+1} l_i^x \tag{5.11}$$

Let us first, find the upper limit. Since $V(t)$ is non-decreasing function of time, we have

$$V(t_2) \;-\; V(t_1) \;\geq\; V(L_i^l) \;-\; V(L_i^k) \tag{5.12}$$

Since WBSQ is worst-case scheduler, packet P_i^l is not served at time L_i^l unless

$$V(L_i^l) \;\geq\; S_i^l \tag{5.13}$$

Since flow is backlogged at this time and there are δ margin of error in finishing time calculation

$$V(L_i^l) \;\geq\; F_i^l \;-\; \frac{l_i^l}{r_i} \;-\; \delta \tag{5.14}$$

Since $\Sigma_{allflows}\ r_i \quad \leq\ r$ the packets will be serviced before or at its theoretical finishing time, then

$$V(L_i^k) \;\leq\; F_i^k \tag{5.15}$$

$$V(t_2) \;-\; V(t_1) \;>\; F_i^l \;-\; \frac{l_i^l}{r_i} \;-\; \delta \;-\; F_i^k \tag{5.16}$$

Since flow f_i is continuously backlogged at least since packet k

$$F_i^l \;=\; F_i^k \;+\; \sum_{x=k\,+\,1}^{x=l} \frac{l_i^x}{r_i} \tag{5.17}$$

$$V(t_2) \;-\; V(t_1) \;>\; F_i^k \;+\; \sum_{x=k\,+\,1}^{x=l} \frac{l_i^x}{r_i} \;-\; \frac{l_i^l}{r_i} \;-\; \delta \;-\; F_i^k \tag{5.18}$$

$$V(t_2) \;-\; V(t_1) \;>\; \sum_{x=k\,+\,1}^{x=l\,-\,1} \frac{l_i^x}{r_i} \;-\; \delta \tag{5.19}$$

from this equation and equation (5.11)

$$S_i(t_1, t_2) \;<\; V(t_2) \;-\; V(t_1) \;+\; \frac{l_i^k}{r_i} \;+\; \frac{l_i^l}{r_i} \;+\; \frac{l_i^{l\,+\,1}}{r_i} \;+\; \delta \tag{5.20}$$

$$S_i(t_1, t_2) \;\leq\; V(t_2) \;-\; V(t_1) \;+\; \frac{3\; l_i^{max}}{r_i} \;+\; \delta \tag{5.21}$$

To get the lower bound we need to consider to cases.

Case I: $F_i^{k-1} \;\geq\; V(A_i^k)$

Since there is no departure in (t_1, L_i^k), we have $L_i^{k\,-\,1} \;\leq\; t_1$ then

$$V(t_2) \;-\; V(t_1) \;\leq\; V(L_i^{l\,+\,1}) \;-\; V(L_i^{k\,-\,1}) \tag{5.22}$$

Since we assume that flows are not oversubscribed i.e. $\Sigma_{allflows} r_i \leq r$ we have

$$V(L_i^{l\,+\,1}) \;\leq\; F_i^{l\,+\,1} \tag{5.23}$$

Since WBSQ only serves packets when their start times started in virtual time and since there is δ margin of error in finishing time calculations, we have $V(L_i^{k\,-\,1}) \;>\; S_i^{k\,-\,1} \;-\; \delta$ or $V(L_i^{k\,-\,1}) \;>\; F_i^{k\,-\,1} \;-\; \frac{l_i^{k\,-\,1}}{r_i} \;-\; \delta.$

$$V(t_2) \;-\; V(t_1) \;\leq\; F_i^{l\,+\,1} \;-\; F_i^{k\,-\,1} \;+\; \frac{l_i^{k\,-\,1}}{r_i} \;+\; \delta \tag{5.24}$$

Since flow f_i is continually backlogged since packet k

$$F_i^{l\,+\,1} \;=\; F_i^k \;+\; \sum_{x=k\,+\,1}^{l\,+\,1} \frac{L_i^x}{r_i} \tag{5.25}$$

from case assumption we have

$$F_i^k \;=\; F_i^{k\,-\,1} \;+\; \frac{L_i^k}{r_i} \tag{5.26}$$

then

$$F_i^{l\,+\,1} \;=\; F_i^{k\,-\,1} \;+\; \sum_{x=k}^{l\,+\,1} \frac{L_i^x}{r_i} \tag{5.27}$$

$$V(t_2) \;-\; V(t_1) \;<\; \sum_{x=k}^{l\,+\,1} \frac{L_i^x}{r_i} \;+\; \frac{l_i^{k\,-\,1}}{r_i} \;+\; \delta \tag{5.28}$$

$$S(t_1, t_2) \;>\; V(t_2) \;-\; V(t_1) \;-\; \frac{L_i^k}{r_i} \;-\; \frac{L_i^{l\,+\,1}}{r_i} \;-\; \frac{L_i^{k\,-\,1}}{r_i} \;-\; \delta \tag{5.29}$$

$$S(t_1, t_2) \;>\; V(t_2) \;-\; V(t_1) \;-\; \frac{L_i^k}{r_i} \;-\; \frac{3\; L_i^{max}}{r_i} \;-\; \delta \tag{5.30}$$

Case II: $F_i^{k\,-\,1} \;<\; V(A_i^k)$

Since flow f_i is backlogged since t_1 and there was no packets departures in (t_1, L_i^k), hence $A_i^k \le t_1$. Therefore

$$V(t_2) - V(t_1) \le V(L_i^{l+1}) - V(A_i^k) \tag{5.31}$$

$$V(t_2) - V(t_1) \le F_i^{l+1} - V(A_i^k) \tag{5.32}$$

$$V(t_2) - V(t_1) \le F_i^k + \sum_{x=k+1}^{l+1} \frac{l_i^x}{r_i} - V(A_i^k) \tag{5.33}$$

from case assumption

$$F_i^k = V(A_i^k) + \frac{l_i^k}{r_i} \tag{5.34}$$

Then

$$V(t_2) - V(t_1) \le \sum_{x=k}^{l+1} \frac{l_i^x}{r_i} \tag{5.35}$$

and

$$\sum_{x=k+1}^{l} l_i^x \ge V(t_2) - V(t_1) - \frac{2\, l_i^{max}}{r_i} \tag{5.36}$$

Since first case is lower than second case, we will use (5.30) and from (5.11)

$$\left| \frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j} \right| \le 3\left(\frac{l_i^{max}}{r_i} + \frac{l_j^{max}}{r_j} + \delta\right) \tag{5.37}$$

■

5.2.3 Worst-Case Fairness

Before we start discussing the worst-case fairness properties of WBSQ, we will introduce a corollary that will be used in the analysis.

Since WBSQ will not service a packet from next bin until all eligible packet in the current bin are serviced. The only difference in packet service between WBSQ and WF^{2}Q+ happens within a single bin. In the worst-case all eligible packets will be served before the eligible flow f_i within that single bin only.

This means that if a flow f_i was eligible to be serviced within a bin and was served the last one due to FIFO. From lemma (5.2.1), the maximum service that all flows would get is $W_\delta \le \delta\, r + L_{max}$

Corollary 5.2.1 *The difference Δ between the departure d_i^{WBSQ} in WBSQ and the departure $d_i^{WF^2Q+}$ in the corresponding WF^{2}Q+ system of eligible flow f_i is bound by*

$$\Delta \leq \delta + \frac{L_{max}}{r} \tag{5.38}$$

where δ is the bin width

Proof.

Since WBSQ and WF^{2}Q+ use the same finishing time calculation, the difference comes from scheduling flows that are inserted into the same bin. The discrepancy comes from the fact that WBSQ uses FIFO within a bin while WF^{2}Q+ does not. From (5.6) above, the maximum delay in service that a flow will experience in WBSQ relative to the service it would receive under WF^{2}Q+ is when packets within a bin are serviced before this packet due to the virtual time calculation errors. Hence, from (5.6)

$$\Delta \leq \delta + \frac{L_{max}}{r} \tag{5.39}$$

■

The fairness measure of Bennet-Zhang (also called worst-case fairness) is a more refined notion of fairness. Rather than comparing the relative amounts of service received by two flows f_i and f_j , it compares the service received by a single flow f_i to the service it would receive in the ideal case, i.e., when f_i has exclusive access to an output link of bandwidth r_i . Suppose a packet belonging to flow f_i arrives, creating a total backlog of q_i in f_i's queue. The fairness measure of Bennet-Zhang requires a bound on the maximum time D_i that elapses before the packet is transmitted, thereby "draining" the backlog of q_i. In particular, it is desired to bound how much D_i is in excess of $q_i\ r_i$, which is the amount of time it would take to clear a backlog of q_i with an output link of bandwidth r_i. Theorem (5.2.2) gives a bound for D_i.

Theorem 5.2.2 $D_i^{WBSQ} \leq \delta + 2\ \frac{L_{max}}{r}$

Proof.

The proof follows from the fact that WBSQ and WF^{2}Q+ use the same virtual time calculation. The only difference between them is the fact that WBSQ uses FIFO within

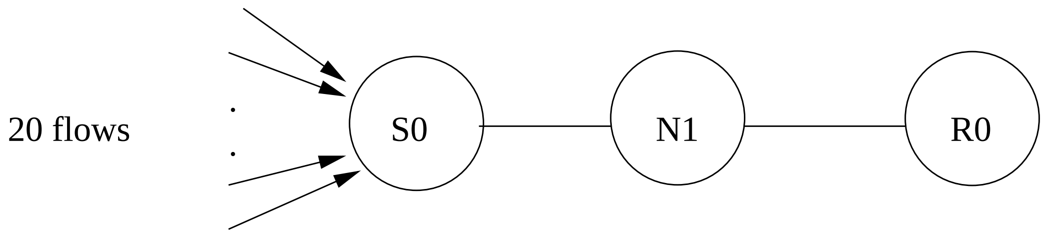


Figure 5.3: Simulated network topology.

a bin. But WBSQ will not service a flow from a bin before serving all eligible flows that belong to preceding bins. So, WBSQ and WF^{2}Q+ only differ in using FIFO within a single bin. From [2] $D_i^{WF^2Q+}$ for WF^{2}Q+ is at most $\frac{L_{max}}{r}$.

From this expression and (5.39), we have

$D_i^{WBSQ} \leq \frac{L_{max}}{r} + \delta + \frac{L_{max}}{r}$

$$D_i^{WBSQ} \leq \delta + 2\frac{L_{max}}{r} \tag{5.40}$$

■

5.3 Experimental Results

In this section we will report the results of our simulation experiments which were designed to investigate WBSQ properties in practical situations and to compare WBSQ with two scheduling disciplines, WF^{2}Q+ and BSFQ. All experiments were performed using ns-2, to which we added WBSQ and BSFQ queuing classes. While we carried out extensive simulations, we will only report the results of two representative experiments, one for short-term throughput and the other one for end-to-end average delay variation. Figure 5.3 shows the network topology used in the experiments. All the links have a bandwidth of 2Mbps and a propagation delay of 1 ms. In all experiments, there are 20 CBR flows from $S0$ node to $R0$ node. The average flow rates range from 10Kbps to 280Kbps. All 20 flows start within interval $(0, 2)$ seconds and stop after ten seconds.

In the first set of experiments, we selected flow number 6 for our measurements. We calculated the average throughput of flow 6 over 100 ms intervals. The x-axis in Figures

5.4,5.5 and 5.6 represents these time intervals and the y-axis represents the throughput in bits per second (bps). These experiments were run for three types of schedulers: BSFQ, WBSQ and WF^{2}Q+. Figure 5.4 shows the results of this set of experiments when $\delta = 1000$ for both BSFQ and WBSQ where δ is the bin width.

As we can see in Figure 5.4, WF^{2}Q+ has the best throughput performance which is almost constant and close to the guaranteed bandwidth of flow 6 which equals $120Kbps$. On the other hand, BSFQ has the largest throughput variation. Hence, BSFQ has the worst throughput performance among the three schedulers. Although our scheduler WBSQ has some throughput variation it is much smaller than BSFQ and most of the time very close to the guaranteed rate of $120Kbps$.

In Figures 5.5 and 5.6, we repeat the same set of experiments for a lower value of bin width (δ). As we can observe, BSFQ still suffers from high throughput variation even with lower values of bin width. On the other hand, WBSQ throughput variation almost reached the same level of performance as that of WF^{2}Q+. From these sets of experiments, we conclude that WBSQ has a much better throughput variation over that of BSFQ. Also, we observe that for small bin width values, the throughput variation performance of WBSQ is very close to that of WF^{2}Q+.

In the second experiment, we measured the average end-to-end delay variation for the a single flow for different δ values and compared it with WF^{2}Q+ and BSFQ. We calculated the average end-to-end delay variation for flow 6 over the whole run time of 10 seconds. The variation in the arrivals of all packets was measured and divided by the number of packets to get the average delay variation. These measurements were recorded for different bin width values as shown in Figure 5.7. The x-axis in Figure 5.7 represents the bin width while the y-axis represents the average delay variation.

From Figure 5.7, we can see that WF^{2}Q+ has the lowest delay variation which is also constant because WF^{2}Q+ does not use bins. On the other hand, BSFQ has the highest average delay variation and its performance does not improve for lower bin width values. However, WBSQ has much better average delay variation than that of BSFQ. Also, we notice that the average delay variation gets even better for smaller bin width values.

Looking at Figure 5.7, we notice that the average end-to-end delay variation for WBSQ is much closer to that of WF2Q if compared with BSFQ for all delta values. Also as bin width δ becomes smaller, WBSQ's average delay variation gets closer to WF^{2}Q++.

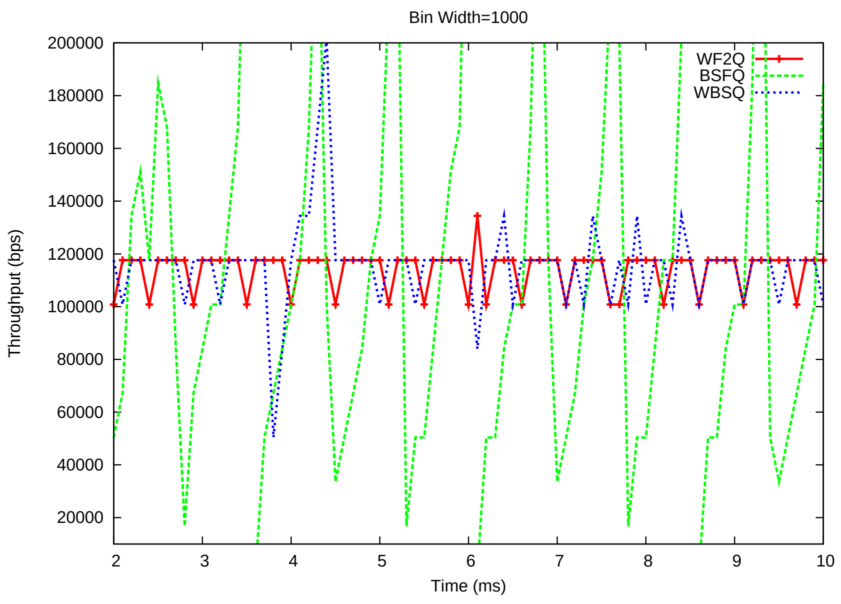


Figure 5.4: Short Term throughput for δ=1000.

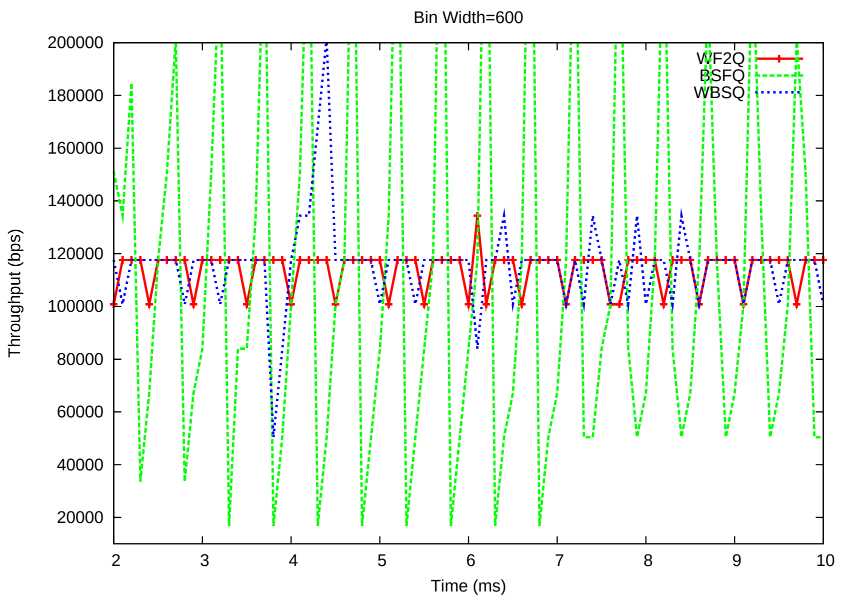


Figure 5.5: Short Term throughput for δ=600.

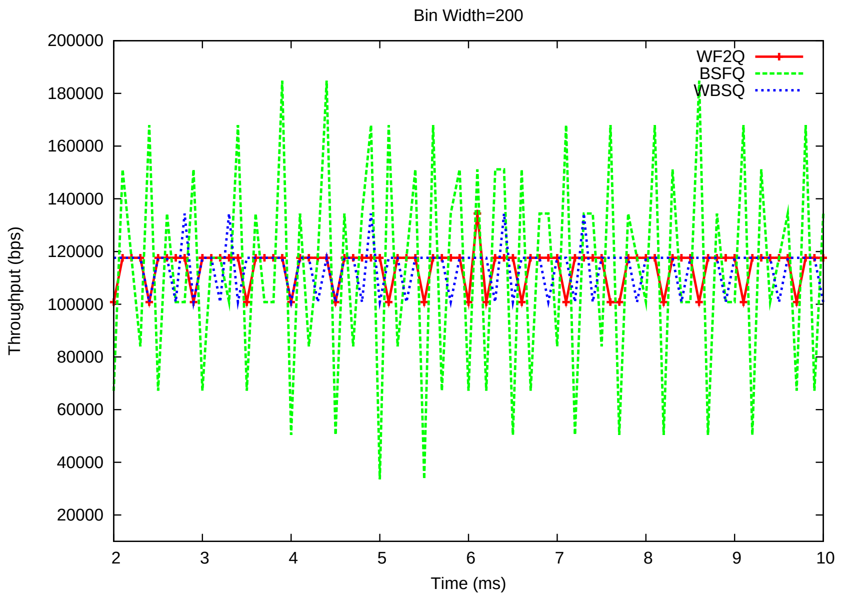


Figure 5.6: Short Term throughput for δ=200.

5.4 Concluding Remarks

We have proposed a new worst-case fair queuing (WBSQ) scheduler that has a constant complexity. The scheduler combines two main simplification ideas. The first idea comes from bin sort concept which represents virtual time in virtual bins of time intervals. The second idea utilizes the simplified virtual time calculation of WF^{2}Q+. Theoretical analysis showed that WBSQ has constant proportional and worst-case fairness indexes. We also compared WBSQ with two other schedulers BSFQ and WF^{2}Q+'s through simulations. The experiments showed that WBSQ was close to WF^{2}Q+ short-term throughput and average delay variation specially for smaller bin size values. WBSQ has constant complexity implementation and can easily be implemented by hardware. Therefore, we believe that employing WBSQ scheduling within high-speed routers will enable network operators to significantly enhance their ability to offer and guarantee a wide range of services.

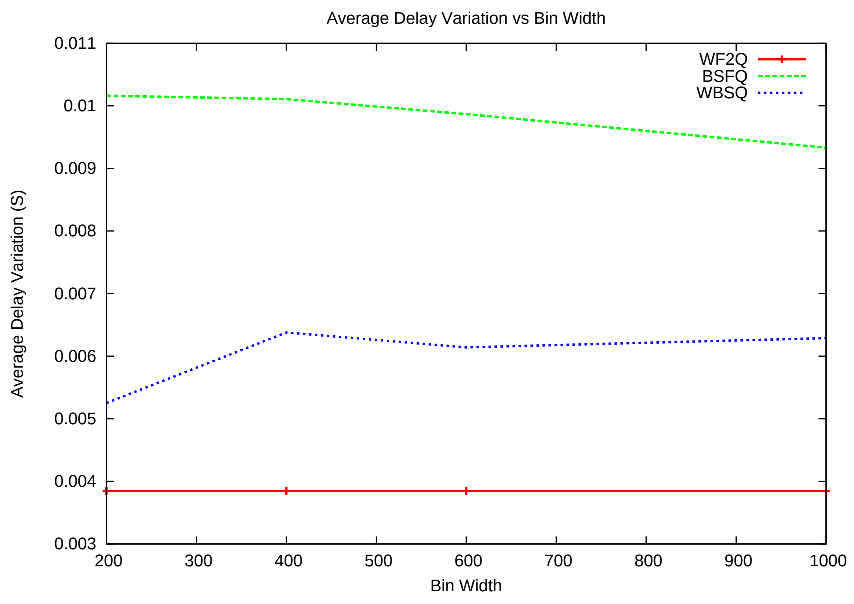


Figure 5.7: Average Delay Variation versus bin width (δ).

Chapter 6

Summary and Future Work

In this thesis we provided a solution for packet schedulers in high speed networks. In such an environment there is a need for both low complexity and fairness. Most schedulers that have been proposed improve one property at the expense of the other. Our solution is based on the quantization approach. In the first part of this thesis, we utilized the quantization of the service offered by the scheduler where we proposed a new family of tiered-service fair queuing (TSFQ) schedulers. TSFQ was motivated by two key observations: that providers typically offer a small number of service levels, and that the Internet packet length distribution exhibits a small number of prominent modes. Within each tier, the schedulers employ a fixed number of queues to handle packets with few or no sorting operations. The intra-tier scheduler simply serves the packet with the smallest timestamp among a constant number of packets at the front of the intra-tier queues. The simple structure and operation of the schedulers are practically realizable and especially attractive for hardware implementation. We have shown that, despite their low complexity, the TSFQ algorithms have good delay and fairness properties; in fact, they are equivalent to WF^{2}Q with the additional property that the virtual time function can be computed in $O(1)$ time. Therefore, we believe that employing TSFQ scheduling within high-speed routers will enable network operators to significantly enhance their ability to offer and guarantee a wide range of services.

In the second part of this thesis, we utilize the quantization of virtual time. A new scheduler was proposed called Worst-Case Bin Sort Queuing (WBSQ). WBSQ scheduler has worst-case fairness property while maintaining a constant complexity. The scheduler combines two main simplification concepts. The first originates from the bin sort idea which represents virtual time in virtual time interval bins. The second idea utilizes the virtual time quantization to calculate system virtual time $V(t)$ of WF^{2}Q+ in constant time. Our analysis showed that WBSQ have constant proportional and worst-case fairness indexes. In addition, we compared WBSQ with two other schedulers BSFQ and WF^{2}Q+ through $ns-2$ simulations. The experiments showed that WBSQ was close to WF^{2}Q+ in short-term throughput and average delay variation, especially for smaller bin size δ values. WBSQ has constant complexity implementation and can easily be implemented by hardware. Therefore, we believe that employing WBSQ scheduling within high-speed routers will enable network operators to significantly enhance their ability to offer and guarantee a wide range of services.

6.1 Future Work

Our work can be extended in several directions.

1. TSFQ as an end-to-end scheduling algorithm. The traffic quantization and (TSFQ) scheduler framework can be utilized to design an end-to-end scheduling algorithm that uses weight metrics to schedule packets across multi hops of the network. The packets can carry two items of information in their headers. The first value represents the weight class or tier that a flow belongs to. The other information represents the total time that the packet had to wait in its local queue before it was released to the common queue. We can come up with an algorithm where core routers give preference to the packets that had to wait the longest in their queues. Also this algorithm can advise a way to guarantee an end-to-end delay for flows among the whole network.

2. Quantizing both service and time. In our approach we only utilize either service or virtual time for quantization. Future approaches could utilize both of these parameters together for quantization. For example, we can use weight as a tier criteria as we did in TSFQ. In addition, we can use virtual time quantization for each of these intra-tier schedulers instead of using separate intra-tier for each packet-size group.

3. UNIT scheduler. In TSFQ, we construct the scheduler from multiple intra-tier schedulers and one inter-tier scheduler. This idea can be extended to find a framework for

a UNIT scheduler. The UNIT scheduler should be simple and generic to represent a building block for the entire scheduler. By combining these UNIT schedulers together, a more complex scheduler can be built. One advantage of using this idea is that it simplifies the fairness analysis of the whole scheduler. For example, knowing the fairness properties of the intra-tier scheduler in TSFQ simplified the fairness analysis of the entire scheduler.

Bibliography

[1] S. Cheung and C. Pencea. Bsfq: Bin sort fair queuing. *IEEE INFOCOM'02*, 2002.

[2] J. C. R. Bennet and H. Zhang. Hierarchical packet fair queueing algorithms. In *IEEE/ACM Trans. Networking, vol. 6*, pages 175–185, April 1998.

[3] S. Keshav. *An Engineering Approach to Computer Networking.* Addison-Wesley, Reading, MA, 1997.

[4] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1992.

[5] J. Xu and R. Lipton. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. In *Proceedings of ACM SIGCOMM '02*, 2002.

[6] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of ACM SIGCOMM '89*, pages 1–12, September 1989.

[8] J. Golestani. A self-clocked fair queuing scheme for broadband applications. In *in Proc. IEEE INFOCOM*, pages 636–646, 1994.

[9] J. C.R. Bennett and H. Zhang. WF^{2}Q: worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM '96*, pages 120–128, 1996.

[10] P. Goyal and H. M. Vin. Generalized guaranteed rate scheduling algorithms: A framework. *IEEE/ACM Transactions on Networking*, 5(4):561–571, August 1997.

[11] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. In *Proceedings of ACM SIGCOMM '96*, pages 157–168, August 1996.

[12] L. Zhang. Virtual clock: A new traffic control scheme for packet switching networks. In *Proceedings of ACM SIGCOMM '90*, 1990.

[13] D. Stiliadis and A. Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.

[14] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of ACM SIGCOMM '95*, 1995.

[15] G. Chuanxiong. SRR, an $O(1)$ time complexity packet scheduler for flows in multi-service packet networks. In *Proceedings of ACM SIGCOMM '01*, pages 211–222, August 2001.

[16] S. Ramabhadran and J. Pasquale. Stratified round robin: A low complexity packet scheduler with banwidth fairness and bounded delay. In *Proceedings of ACM SIGCOMM '03*, pages 239–249, August 2003.

[17] Xin Yuan and Zhenhai Duan. Frr: a proportional and worst-case fair round robin scheduler. *Proceedings of IEEE INFOCOM '05*, March 2005.

[18] G. J. Miller K. Thompson and R. Wilder. Wide-area internet traffic patterns and characteristics. In IEEE Network, pages 11(6):10–23, NOV/DEC 1997.

[19] C. Papadopoulos R. Sinha and J. Heidemann. Internet packet size distributions: Some observations. October 2005.

[20] G. N. Rouskas and Z. Dwekat. A practical and efficient implementation of WF^{2}Q+. In *Proceedings of IEEE ICC*, pages 172–176, June 2007.

[21] S. Suri, G. Varghese, and G. Chandranmenon. Leap forward virtual clock: An $O(\log \log N)$ queueing scheme with guaranteed delays and throughput fairness. In *Proceedings of IEEE INFOCOM '97*, 1997.

[22] Ajay Babu Amudala Bhasker. Tiered-service fair queueing (TSFQ): A practical and efficient fair queueuing algorithm. August 2006.

[23] Shrikrishna Khare. Testbed implementation and performance evaluation of the tiered service fair queueing (TSFQ) packet scheduling discipline. Master's thesis, North Carolina State University, Raleigh, NC, August 2008.

[24] Linux Kernel Organization. The Linux kernel archives. http://www.kernel.org/.

[25] R. Jain, W. Hawe, and D. M. Chiu. A quantitative measure of fairness and discrimination for resource allocation in shared systems. Technical Report TR-301, DEC Research Report, 1984.