A Dynamic, Tunable, QoS-fair Scheduling Scheme for Mutlimedia Streams

Michael S. Boykin[†]

Taieb F. Znati[†,§]

[†]Department of Computer Science

[§]Telecommunications Program

University of Pittsburgh

Pittsburgh, PA 15260

{*boykin1, znati*} *@cs.pitt.edu*

Abstract: A great deal of packet scheduling research in high speed integrated, multimedia services networks (*ISN*s) aims to efficiently emulate the *Generalized Processor Sharing* (*GPS*) service discipline. The approach used by these schemes has been, first, to maximize the emulation accuracy through the use of packet-by-packet scheduling decisions and then to minimize the complexity involved in making such decisions. While reductions in algorithmic complexity are useful, we believe the straight-forward *GPS* emulation prescribed by these schemes is not necessary in *ISN*s. It is the fulfillment of *quality of service QoS* guarantees, not *GPS* accuracy, which represents the primary metric for success in *ISN*s. In this paper, we present a new scheduling approach, *QoS-Aware Fair Sharing* (*QFQ*), designed to reduce packet scheduling overhead while guaranteeing the *QoS* requirements of supported applications. Because the central goal of the *QFQ* discipline lies in meeting the *QoS* requirements of backlogged flows and not guaranteeing a weighted-fair portion of the server's output to such flows, the *QFQ* approach provides *ISN* server's with a great deal of flexibility in determining how to service contending application flows. This paper presents this new and innovative scheme which seeks to exploit an excess capacity allocation policy in order to achieve a reduction in service overhead. This paper describes how such an objective can be achieved and shows how excess capacity can be identified for this purpose despite the presence of non-empty service queues. A set simulation experiments demonstrate the capabilities of the *QFQ* server and compare its performance to other *ISN* scheduling schemes.

Key Words: Multimedia, Multi-service network, Integrated service network, Quality of service, packet scheduling, Fair packet service.

1 Introduction

The explosive growth of multimedia and other bandwidth-intensive applications over the World Wide Web has resulted in a rapid increase in the size of the traffic loads to be supported by modern networks. At the same time, there is a growing need for these networks to provide a wide-range of substantive *Quality of Service* (*QoS*) guarantees to applications transmitting time sensitive audio and video data. Further, the support of such applications requires the development of an effective scheduling strategy which enables a wide-range of *QoS* requirements to be met while simultaneously allowing an efficient implementation in high speed networks. These requirements pointed out the need for the development of a new integrated, multimedia services network (*ISN*) architecture. Many of the questions relating to this architecture have already been addressed with the standardization of the *Asynchronous Transfer Mode* (*ATM*) protocol and the efforts taking place within the *Internet Engineering Task Force* (*IETF*). However, the task of packet scheduling remains an open issue. As a result, several schemes which address traffic scheduling in *ISN* architectures have been proposed.

In [Parakh et al, 1993, Abhay et al, 1994], Parekh and Gallagher suggested *Generalized Processor Sharing* (*GPS*) as a service discipline to provide a wide range of heterogeneous application requirements in a fair manner. While *GPS* only represents an idealized service model that can not be implemented practically, several approximation schemes have been developed which attempt to emulate the behavior of a *GPS* server in a discrete manner. These disciplines, col-

lectively referred to as fair queuing disciplines, have as their primary goal to maximize the accuracy of their ideal *GPS* server emulation.

Despite their highly accurate emulation of a *GPS* server, the proposed fair queuing disciplines were found to be difficult to implement efficiently in high speed environments. Several follow-up schemes [Bennett et al, March, 1996, Golestani 1994, Subash et al, 1996, Varma et al, 1996] attempted to address this shortcoming and generally focused on improving one of two main aspects of *GPS* server emulation. The first focal point was on reducing the overhead involved in mapping real time to virtual time in order to reference the emulated server's behavior. The second focal point was on reducing the algorithmic complexity required to select the flow[1] which is to be serviced next. These improvements, while maintaining high levels of emulation accuracy, led to a considerable reduction in the scheduling complexity of fair queuing algorithms

However, further analysis of fair queuing implementations in high speed environments, reveals that a major source of scheduling overhead is due to the relative impact of the number of service decisions performed by the server per unit time. To illustrate this, consider an *ATM* network environment characterized by a link transmission rate of 1 Gbps and a cell size of 53 bytes. In this environment, an active fair queuing server may perform over 2.3 million scheduling decisions during a 1 second interval. As a result, even when the number of instructions required per scheduling decision is small, the total overhead required to schedule all cells may be nonnegligible relative to the total throughput of the link. Accordingly, performing *packet-by-packet* scheduling decisions in order to achieve an accurate emulation of a fluid *GPS* server often leads to excessive overhead. Considering further that human senses generally require time on the order of a few milliseconds to respond to a given stimulus, over two thousand *ATM* cells could be transmitted prior to the human receiver becoming aware of the first transmission. Thus, in addition to its potential for excessive overhead, emulating a fluid *GPS* server within a single cell also may not achieve any noticeable service improvement for the user.

As a result, the achievements of the follow-up fair queuing schemes are limited due to their goal of accurate *GPS* emulation. Furthermore, their efforts are also limited in light of the fact that application *QoS* guarantees rarely require an emulation server to allocate unused bandwidth according to the strict approach prescribed by *GPS* [Clark et al, 1992, Parekh et al, 1992]. Consequently, a *packet-by-packet* emulation of a *GPS* server is rarely needed in order to ensure *QoS* guarantees. In fact, the work in [Clark et al, 1992] suggests that, although fair queuing schemes reduce scheduling complexity, such schemes execute more scheduling decisions, and therefore incur more overhead, than is required to ensure *QoS* requirements are met.

More recently, a number of schemes have recognized that highly accurate *GPS* emulation is rarely required to ensure *QoS* guarantees [Diffield et al, 1998, Goyal et al, 1997]. The focus of these schemes shifted from precisely emulating *GPS* to achieving alternative service goals using the available excess capacity at the server. As opposed to the fair queuing policy of assigning the unused bandwidth of idle flows to active ones in proportion to their long term relative service weights, these approaches attempt to distribute excess service capacity in a manner that is cognizant of each flow's instantaneous resource requirements. In doing so, these schemes often increase the complexity of the scheduling discipline by having the server maintain state information with respect to multiple service goals. The added overhead emanating from the increased state maintenance adds to the complexity of these disciplines making them even more difficult to implement in high speed environments.

In this paper, we will demonstrate an innovative approach which, unlike previous attempts, aims to exploit excess service capacity in order to facilitate an efficient fair queuing implementation that guarantees application *QoS* requirements. To this end, the proposed *QoS-Aware Fair Queuing* (*QFQ*) service discipline is developed to retain the beneficial characteristics of a fair, *work conserving* discipline while "tuning" its decision making strategy to accommodate a high-speed *ISN* environment. Rather than performing scheduling decisions on a *packet-by-packet* basis as prescribed by other fair queuing schemes, the *QFQ* discipline performs scheduling decisions on a *quantum-by-quantum* basis, where a quantum represents a dynamically computed number of packet transmission times. The points in time at which these scheduling decisions occur are referred to as *service synchronization points*. Therefore, the *QFQ* discipline's primary goal rests in lengthening the server's service quantum by exploiting the availability of excess bandwidth occurring as a result of statistical multiplexing. The gains attributed to this approach can function in concert with the algorithmic

---

[1] For the purposes of this paper, a flow is defined as a sequence of packets originating from a common source and traversing the same path through the network.

improvements of earlier work as this approach can be interpreted as an attempt to reduce the number of scheduling decisions performed without impacting the computational complexity of the scheduling decision itself.

Initially, one may think that by performing scheduling decisions less often, the QFQ server's emulation accuracy will be impacted negatively and lead to adverse consequences with regard to the server's admissibility constraints. However, the QFQ discipline's QoS-$Aware$ process of identifying the excess bandwidth used to expand the server's service quantum is based upon the service requirements of the currently backlogged flows. This implies that in times when the server must provide highly synchronous service due to stringent QoS demands, the quantum is shrunk by inserting a larger number of $service$ $synchronization$ $points$ so as to meet these demands. Conversely, the quantum size is expanded by reducing the number of $service$ $synchronization$ $points$ when the QoS demands of the supported applications allow it. Dynamically adjusting the server's quantum size in this way enables a QFQ server to seek out performance gains without affecting the server's admissibility constraints. Thus, the level of emulation accuracy supported by the QFQ discipline is said to be QoS-$fair$ as it adapts to the QoS requirements of the server's workload. The term "tuning", within this context, refers to the process of selecting the largest service quantum which minimizes scheduling overhead while ensuring that the QoS requirements of all backlogged flows are met.

In addition, it may also seem that the QFQ server's QoS-$fair$ approach may not adequately support applications whose required QoS is lower than its desired QoS level. For example, a voice application may function acceptably with a delay as high as 100 milliseconds. However, given a delay of 10 milliseconds, the observed application quality may be much higher. In this situation, it is important to remember that a QFQ server is $work$ $conserving$ in nature and is active whenever some flow is backlogged. Thus, even though the QFQ server attempts to construct service quanta which ensure that the application's 100 millisecond delay bound is met in the worst-case, lower actual delays can be experienced if the other contending flows do not behave according to their most demanding specification. Consequently, the QFQ approach can provide such applications with enhanced levels of service as long as the server first achieves its goals of QoS fulfillment and reduced overhead.

In Section 2, we will discuss related work from the literature regarding fair queuing and its most well-known emulation servers. Next, a discussion of the QFQ service discipline's implementation and design issues is presented in Sections 3 and 4. To do this, we will first describe an ideal QFQ server's behavior which provides a fluid service model. The introduction of this server is not provided to advocate its use over other schemes which strive to attain good emulation accuracy. Instead, it is given in order to serve as a reference model upon which the "tuned" QFQ server is built. The discussion of the "tuned" QFQ server will specifically describe the QFQ scheduling policy and methodology for reducing scheduling overhead while maintaining adequate application QoS support. We will discuss the QFQ discipline's design parameters which allow a QFQ server to dynamically "tune" its service quanta based upon the QoS requirements of the currently backlogged flows. Further, the discussion will demonstrate how such "tuning" enables the QFQ server to strike a balance between its desire to perform as few scheduling decisions as possible and the need to meet the QoS requirements of all backlogged applications. In the discussion of the scheme, we will describe how servers in high speed ISN environments can exploit the QoS tolerances of its active applications in order to identify excess bandwidth and enable multiple packets to be serviced per scheduling decision. Subsequently in Section 5, we will describe a simulation experiment contrasting the QFQ discipline's performance results to those of other well-known schemes from the literature. Lastly, Section 6 will summarize the contributions of the paper and will suggest possible extensions for future work.

2   Related Work

There have been several schemes presented in the literature which seek to maintain a high level of fairness while reducing the complexity of GPS emulation. A class of these schemes, [Subash et al, 1996, Geoffrey et al, 1996], has sought to simplify the scheduling algorithm by employing improved data structures for storing the set of flow priorities. In one such scheme [Geoffrey et al, 1996], the authors describe an adaptive heap data structure which is used to decide when new flow priorities, originally maintained in a linked list, should be inserted into the server's heap. The authors argue the resulting data structure adapts to its environment providing good average case performance and a low worst-case complexity of $\mathcal{O}(\log N)$; where N denotes the number of supported flows. In yet another scheme [Subash et al, 1996], Suri et. al propose the $Leap$ $Forward$ $Virtual$ $Clock$ $(LFVC)$, which reduces the scheduling complexity to $\mathcal{O}(\log \log N)$ requiring only an extra maximum-sized packet's transmission time above the worst-case delay bound

for *WFQ*.

Still other approaches have reduced the complexity of their emulation by improving upon the virtual time computation used to simulate the ideal server's behavior. In [Bennett et al, April 1996], a *Worst-Case Fair Weighted Fair Queuing Plus* (WF^{2}Q$^+$), a scheme that executes in $\mathcal{O}(\log N)$ time while providing the same worst-case delay bound as *WFQ*, is described. Other schemes such as *Self-Clocked Fair Queuing (SCFQ)* [Golestani 1994] and *Frame-based Fair Queuing (FFQ)* [Varma et al, 1996] simplify the server's virtual time computation by approximating its value. The *SCFQ* scheme, however, suffers from the fact that its worst-case delay bound can grow as the number of supported flows increases. Alternatively, the *FFQ* scheme suffers from the fact that its fairness bound depends upon the minimum rate allocated by a server. As a result, the usefulness of the latter two schemes depends on the extent to which their approximation has an impact on admissions decisions.

The algorithmic improvements presented by the schemes discussed here are compatible with the *QFQ* approach. Where these schemes have focused on reducing the cost of a scheduling decision, the *QFQ* scheme's focus has been to reduce overhead in terms of the number of decisions required to ensure *QoS* guarantees are met. Consequently, although efforts to reduce the algorithmic complexity of *GPS* emulation are useful in tailoring fair queuing disciplines for use in high speed environments, alone, they may not go far enough. As demonstrated in [Clark et al, 1992, Yates et al, 1993], high levels of fairness are rarely required to ensure application *QoS* guarantees are fulfilled. Even more importantly, though, technological advances which permit server rates to increase dramatically not only reduce the time available to implement a given scheduling policy, they also facilitate the server supporting a larger number of application flows. This fact is especially true given the large efficiency gains being acheived through the use of new compression algorithms. Hence, benefits gained through a reduction in algorithmic complexity can be offset by increases in the number of flows to be scheduled.

In light of this, there is a growing body of research which endeavors to exploit this fact in order to enhance applications' observed *QoS*. Most of these schemes, however, do not address the issue of unacceptable levels of overhead in high speed environments. For example, although the techniques discussed in [Diffield et al, 1998] address how to distribute excess bandwidth in order to optimize loss probabilities or control delay distributions, the ap-

proach presented requires the server to emulate two ideal disciplines; one to enforce *QoS* guarantees and the other to distribute excess bandwidth. Therefore, the server must maintain the necessary state required for both disciplines. As state maintenance represents the most expensive scheduling discipline task, such an approach may not be appropriate for high speed environments.

A class of scheduling disciplines, referred to as Fair Airport scheduling algorithms, was introduced by Cruz in [Cruz 1992] and later in [Goyal et al, 1997] by Goyal and Vin. In these disciplines, every packet of a flow enters a rate regulator and and an auxiliary service queue upon arrival. Once a packet is allowed through the rate regulator, it joins a guaranteed service queue if it has not been serviced by the auxiliary service queue server already. To satisfy the desired objectives, a Fair Airport server may employ a different service discipline for each of the two service queues maintained. The resulting server is *work conserving* with priority being given to packets of the guaranteed service queue over those in the auxiliary service queue. In their paper [Goyal et al, 1997], Goyal and Vin discuss a scheduling discipline which attempts to define a fair queuing server which achieves a separation of rate and delay allocation. The algorithm they present uses a *non-work conserving* server to enforce delay bounds in conjunction with the guaranteed service queue and a fair, *work conserving* discipline, *Start-Time Fair Queuing (SFQ)*, to provide fairness in the auxiliary service queue.

Like Goyal and Vin's Fair Airport scheme, the *QFQ* scheme also attempts to achieve a measure of separation between rate and delay allocation. By dynamically controlling the level of fairness to be maintained, the *QFQ* server provides a flow with differing worst-case delay bounds given the same guaranteed rate. In so doing, however, the *QFQ* scheme does not seek to extend an application's delay bounds by more than the application can tolerate. Thus, the *QFQ* server seeks to strike a balance between straightforward *GPS* emulations which restrict excess allocation strategies and the Fair Airport algorithms which require the state for multiple disciplines to be maintained. By providing a unified scheduling strategy which integrates a fair queuing discipline and an excess capacity allocation algorithm, the *QFQ* server attempts to balance its need to lower overhead along with a desire to more effectively allocate excess capacity. Unlike the schemes presented in [Diffield et al, 1998, Goyal et al, 1997] though, *QFQ* employs an excess capacity algorithm which permits multiple packets to be serviced between *service synchroniza-*

tion points, and in turn, reduces overhead. Thus, rather than striving to implement an ideal excess capacity allocation scheme which requires *packet-by-packet service synchronization points*, the *QFQ* approach trades a reduced level of flexibility in terms of its excess capacity allocation algorithm choices, in order to provide a dynamic, but bounded, level of fairness with lower overhead.

The Burst Scheduling scheme presented in [Lam et al, 1995] employs a *non-work conserving* scheduling discipline to ensure successive burst from a given application source are provided with the required level of service to ensure an acceptable *QoS*. To reduce the complexity of the server's process of selecting the packet to be serviced next, a Burst Scheduling server combines the use of an adaptive heap with a group priority mechanism. A group's size is computed so as not to increase the flow's worst-case delay. The idea behind the use of group priority is to avoid labeling each packet from a given flow with its own priority level. Instead, a group of packets from the same burst and the same flow are labeled with the same priority as the last packet in the group. In this way, the state of the priority queue will require updating once per group rather than once per packet. The shortcoming of this approach is that the scheme employs a *non-work conserving* discipline to manage observed delays and that packet group sizes are computed so as not to increase the worst-case delay.

The approach prescribed by *QFQ* differs in that it is *work conserving* and, therefore, inherits all the benefits of such approaches over *non-work conserving* ones. To manipulate delays, then, the *QFQ* approach sets the level of service fairness, via its service quantum, so as not to exceed an application's worst-case delay bound. Thus, *QFQ* can allow groups of packets to be serviced between *service synchronization points* despite the fact that the packets may belong to different bursts. Finally, the dynamic approach employed by the *QFQ* server better enables it to take advantage of statistical multiplexing allowing larger quantum sizes.

3 An Ideal QFQ Server

In this section, we will define a new service model which endeavors to build upon the service model specified by *GPS* by employing a more flexible resource sharing policy definition. In this new approach, resource sharing is defined according to an individual application's *QoS* requirements (i.e throughput and delay) rather than the relative service weights of all active flows. By defining a resource

sharing policy in this manner, we attempt to allow the server greater flexibility in determining how to distribute excess resource capacity as compared to the *GPS* definition. Later, it will be shown how this flexibility can be used to achieve reduced service overhead. In addition, we expect the model presented here to more readily accommodate future improvements in traffic specification accuracy and call admission control.

A real-time flow supported by an ideal *QFQ* server (*i-QFQ*) is characterized by a set of specification parameters which include the flow's average rate, r, its maximum burst size, σ, and its delay bound, Δ. The server is *work conserving* and operates at a fixed rate R. Let $N_m(t_0, t_1)$ represent the *neediness deviation rate* of flow m over the interval $[t_0, t_1]$. A flow's *neediness deviation rate* is a weighted measure of the rate at which the flow's *allocated bit rate* is deviating from its *equipartition bit rate*. A flow's *allocated bit rate* denotes the actual service rate received from an *i-QFQ* server. On the other hand, a flow's *equipartition bit rate* represents the fair share service rate the flow should receive from an *i-QFQ* server over a given interval of time. The *equipartition bit rate* associated with a given flow can fluctuate over time as a function of both the flow's *QoS* guarantees and the current number of active flows at the server. Consequently, an *i-QFQ* server can be defined as one for which:

$$N_m(t_0, t_1) = 0$$

for all flows, m, and intervals, $[t_0, t_1]$.

Let $\mathcal{E}_m(t_0, t_1)$ denote flow m's *equipartition bit rate* over interval $[t_0, t_1]$. Further, let $\mathcal{A}_m(t_0, t_1)$ represent m's *allocated bit rate* during $[t_0, t_1]$. Flow m's *neediness deviation rate* over the interval $[t_0, t_1]$ can then be define as:

$$N_m(t_0, t_1) = \frac{\int_{t_0}^{t_1} (\mathcal{E}_m(t_0, x) - \mathcal{A}_m(t_0, x)) \, dx}{(t_1 - t_0)} \quad (1)$$

To ensure a proper level of service is provided for each flow, an *i-QFQ* server maintains three parameters with respect to each flow's service: the length of time the flow has been eligible for service, the share of service it should have received during that time and the share it actually received. These parameters include the flow's *busy_start_time*, its *equipartition bit share* and its *allocated bit share*, respectively. The first parameter, *busy_start_time* denoted by t_0, represents the start time of a flow's most recent busy interval at the server. To prevent idle flows from accumulating service credits[2] during periods of inac-

---

[2] The term service credits denotes the fact that the value of a flow's *equipartition bit share* is greater than its *allocated bit share*.

tivity, the *busy_start_time* parameter is updated only when the flow becomes active.

The second parameter, *equipartition bit share*, provides a measure of the amount of service which should be provided to a particular flow over a given interval of time. This measure can be used to describe the behavior of any ideal server, fluid or discrete, which guarantees a specified service level. All that is assumed is that a flow's *equipartition bit share* ensures the flow's *QoS* will be satisfied and that the sum of all flows' *equipartition bit shares* is bound by the maximum service provided by the server at all times. Since a particular flow's *equipartition bit rate* fluctuates over time, the flow's *equipartition bit share* which is derived from the flow's *equipartition bit rate*, also fluctuates. As a result, a flow's *equipartition bit rate* as well as its *equipartition bit share* values can range from zero to as high as the server's maximum allowed value, which is constrained by its rate, R. Flow m's *equipartition bit rate*, $\mathcal{E}_m(t_0, t_1)$, over $[t_0, t_1]$ is used to derived its *equipartition bit share*, $E_m(t_0, t_1)$, over that interval as follows:

$$E_m(t_0, t_1) = \int_{t_0}^{t_1} \mathcal{E}_m(t_0, x) \, dx \qquad (2)$$

where t_0 denotes m's *busy_start_time*.

The definition of the *equipartition bit rate* which is guaranteed by an *i-QFQ* server differs from the guaranteed service rate provided by a *GPS* server. In *GPS*, a flow m's guaranteed service rate is based upon its statically defined service weight, ϕ_m. In contrast, the service rate guaranteed to a flow supported by an *i-QFQ* server is dynamically-defined based upon the flow's *QoS* requirements and is reflected in the flows varying *equipartition bit rate*. Furthermore, where a *GPS* server's guaranteed rate with respect to an active flow is based upon that flow's relative service weight in comparison to the service weights of other active flows, an *i-QFQ* server's guaranteed rate with respect to an active flow is based upon that flow's *QoS* requirements at that time. This difference provides an *i-QFQ* server with a great deal of flexibility in allocating service as *QoS* requirements over a particular interval can be determined in a multitude of ways.

Finally, the third parameter, *allocated bit share*, quantifies the actual number of bits an *i-QFQ* server has serviced on behalf of flow m during the interval $[t_0, t_1]$. Similar to the *equipartition bit share* parameter, the *allocated bit rate* parameter, $\mathcal{A}_m(t_0, t_1)$, can be used to derive a flow m's *allocated bit share*, $A_m(t_0, t_1)$, as described in Equation 3.

$$A_m(t_0, t_1) = \int_{t_0}^{t_1} \mathcal{A}_m(t_0, t_1) \, dx \qquad (3)$$

A few observations can be made regarding an *i-QFQ* server. First, when the sum of all flows' *equipartition bit rate* is bound by the server's maximum rate R, it can be noted that an *i-QFQ* server provides each flow with an *allocated bit rate* equal to its *equipartition bit rate*. Thus, an *i-QFQ* server is, indeed, characterized as a server in which the *neediness deviation rate* of all flows is zero at all times. In addition, a closer look at Equation 1 reveals that the value computed reflects the server's rate of deviation with respect to the service provided to a given flow and not the number of service credits accumulated by that flow. Using the rate, rather than the number of accumulated service credits, prevents flows from monopolizing a server's service by obtaining a large number of service credits during periods of idleness. Thus, a potential cause of server instability is avoided [McCann et al, 1998].

Based on its definition, it should be clear that an *i-QFQ* server defines a fluid discipline which is not readily implementable in practice. Ensuring that all flows' *neediness deviation rates* remain equal to zero at all times implies the server must be capable of servicing all backlogged flows simultaneously. This can be difficult not only because of the large number of flows which may be backlogged at a high-speed *ISN* link server, but also the fact that the number of backlogged flows varies dynamically over time as a function of the flow's input rate and the server's service rate. Despite these facts, insight into the server's behavior can be gained through the use of a *Virtual Parallel Processor* (*VPP*) server.

A *VPP* server consists of N virtual processors, ϱ_k, $1 \leq k \leq N$. Virtual processor ϱ_m continually transmits bits on behalf of active flow m at m's *allocated bit rate*, $\mathcal{A}_m(t_0, t_1)$. The sum of the $|\mathcal{N}(t_1)|$ active processors' execution rates is denoted with the term, $\mathcal{R}^{vpp}(t_1)$. While the *GPS* discipline's allocations would imply that $\mathcal{R}^{vpp}(t_1) = R$ for all times t_1, ensuring all flows' *neediness deviation rates* remain zero only implies that:

$$\sum_{j \in \mathcal{N}(t)} \mathcal{E}_j(t_0, t) = \mathcal{R}^{vpp}(t) \leq R$$

holds for all t in an *i-QFQ* server. Therefore, unlike the *GPS* server, an *i-QFQ* server need not proportionately improve a flow's *QoS* when the requirements of the active flows do not consume all of the available processor capacity. This leaves the *i-QFQ* server with remarkable flexibility regarding how it will allocate its excess processor capacity. The approach we shall prescribe in this paper calls for the

i-QFQ server to implement an integrated strategy which allocates excess capacity to best-effort traffic. Although a more general approach can be formulated to accommodate a wider range of service goals for other non-guaranteed traffic types (e.g. statistical, predictive, ABR, etc.), we shall only describe this one approach in the interest of both simplicity and space.

We conclude this section with the statement of two lemmas and a theorem describing the service provided by the i-QFQ server. The two lemmas shall be offered without proof as they follow directly from the i-QFQ server definition.

Lemma 1 *Given a stable system where the sum of the committed equipartition service rates of all active flows is bound by the server's capacity for all t, each flow's allocated bit rate is lower bound by its equipartition bit rate for all times, t.*

Lemma 2 *A new flow may be admitted into the network at a node k without violating any flow's QoS guarantee if, after the admission of the new flow, the sum of the equipartition bit rates of the supported flows, including the new one, remains bound by the server's capacity for all t.*

While these two lemmas have focused on the server's ability to provide the required service rate to supported flows, the theorem which follows next addresses the maximum service delay to be experienced by a packet, p_m^n. The variable p_m^n represents the nth packet from flow m which is said to have arrived at time a_m^n. The number of bits which have arrived from flow m during the interval $[t_0, t]$ is denoted by the variable $\alpha_m(t_0, t)$. Thus, $\alpha_m(0, a_m^n)$ can be used to describe the total number of bits, including the number of bits in p_m^k, which have arrived from flow m since the server began.

Theorem 1 *Given that the sum of the equipartition bit rates of all active flows is bound by the server's capacity for all times, t, the delay suffered by packet p_m^n is given by:*

$$\{ \min t \ (a_m^n < t \leq t') \quad | \quad \int_0^t \mathcal{A}_m(0, x) \ dx$$
$$\geq \int_0^{t'} \mathcal{E}_m(0, x) \ dx$$
$$\geq \alpha_m(0, a_m^n) \} - a_m^n \ (4)$$

Proof. This theorem states the latest departure time experienced by a flow m packet arriving at time a_m^n is upper bound by the t', where t' denotes the earliest time in which the value of $\mathcal{E}_m(0, t')$ requires

$\alpha_m(0, a_m^n)$ or more bits to have been transmitted on behalf of flow m. As the i-QFQ server's definition dictates, $\mathcal{A}_m(0, x)$ is equal to $\mathcal{E}_m(0, x)$ for all x in this interval. Thus, the actual delay, $(t - a_m^n)$, suffered by packet p_m^n must be equal to m's delay bound, Δ_m, which is enforced implicitly by $\mathcal{E}_m(0, t)$.

4 An Efficient QFQ Server

In this section, our purpose is to define both a realizable and efficient server implementation scheme for the i-QFQ discipline. The scheme's objective is to define an efficient virtual time based approach which borrows the notion of packet starting and finishing times from the body of work regarding *non-work conserving* frameworks in order to better attempt to cause a flow's actual delay performance to match its specified delay bound. However, the QFQ discipline defines a *work conserving* server which, instead of allowing the server to remain idle when none of the active flows have an eligible packet, causes the QFQ server to "logically" advance its virtual time clock to the earliest starting time of some currently queued packet. Thus, the scheme attempts to gain some of the control *non-work conserving* schemes provide while retaining the high resource utilization and fairness of work conserving schemes.

Before beginning the scheme description, we would like to describe the server data structures in general. It is assumed that the server maintains a separate service queue for each of its supported flows. Arriving packets are queued in $FIFO$ order into their respective flow's service queue. The server uses the variable $H_m(t)$ to denote the packet which is at the head of the flow m service queue. Throughout this section, flow m's starting and finishing times shall be assumed to refer to $H_m(t)$'s starting and finishing times. In addition, it is assumed that the server maintains a priority queue data structure to facilitate efficient ExtractMin, Insert and Delete operations. The elements of the priority queue data structures are pointers to the respective flow service queues. When a flow is selected to receive service, its packets are serviced in $FIFO$ order using Dequeue operations on the flow's service queue until its quantum expires.

The focus of this paper, however, is not to reduce the complexity of any of the operations used to emulate GPS (although these improvements are compatible with the scheme described here). Instead, the focus of this paper is to reduce the number of times the ExtractMin operation is used in the process of selecting the flow to be serviced next. The discussion presented shall first introduce the server's algorithm for computing packet starting and finishing

times. Next, the scheduling policy algorithms will be presented and discussed. Following this, some proofs will be developed about the QoS support provided by the QFQ server. The section ends with the presentation and discussion of further revisions aimed at enhancing service to best-effort or other non-guaranteed traffic classes.

4.1 QFQ virtual time computation

The computation of the virtual time, if not done carefully, can be one of the most expensive operations involved in GPS emulation. However, it has been shown in [Bennett et al, April 1996] that this computation could be done efficiently as follows:

$$
\begin{aligned}
\mathcal{V}(t) &= \text{real time, for all } t \text{ where } \mathcal{B}(t) = \emptyset \\
\mathcal{V}(t+\tau) &= max\left(\mathcal{V}(t) + \tau, min_{m \in \mathcal{B}(t+\tau)} S_m^{H_m(t+\tau)}\right)
\end{aligned}
$$

In the equation above, $\mathcal{B}(t + \tau)$ denotes the set of backlogged or active flows at time t and $S_m^{H_m(t+\tau)}$ denotes the starting time of the packet at the head of the flow m service queue. Note that due to the inclusion of τ in the first term of the maximization, $\mathcal{V}(t)$ moves at least as fast as the real time. Therefore, a server's virtual time at a time t is greater than or equal to the real time. Also note that the second term of the maximization ensures that, if some flow is backlogged, at least one flow is always eligible for service. This fact causes the server to be *work conserving*. Finally, virtual time is reset to the real time on each of the server's transitions from idle to active state.

4.2 Computing starting and finishing times

Yet another goal of the QFQ server is to exploit the fact that many times a GPS emulation server (WFQ, $PGPS$ or WF^2Q, etc.) delivers flow m packets earlier than needed due to GPS's strict fairness definition. To better understand this, recall that a GPS emulation server normally assigns each real-time flow m a minimum guaranteed rate $\mathcal{E}_m$ which is greater than or equal to its average rate, r_m[3]. Consequently, it is possible that with $\mathcal{E}_m$, the GPS emulation server provides flow m with a worst case delay, δ_m, that is less than m's specified delay bound, Δ_m, in the worst case.

$$
\Delta_m \geq \delta_m = \frac{\sigma_m}{\mathcal{E}_m} + \frac{S_{max}}{C} \tag{5}
$$

---

[3] To understand the reasoning for this see [Parekh et al, 1992].

To address the undesirable side-effect of GPS's strict fairness definition, a QFQ server computes e_m, the minimum earliness of flow m packets in the worst case. The value of e_m is determined by subtracting m's worst case delay bound, δ_m, from its specified delay bound Δ_m.

$$
e_m = \Delta_m - \delta_m \tag{6}
$$

The QFQ server uses e_m to compute the starting time of the packet which begins each flow m burst identified by a transition from idle to active state. As a result, it determines how long the server can postpone service to flow m without violating it QoS guarantee. Given a flow m *busy_start_time* of t_0, the QFQ server attempts to wait until time s_m before allowing flow m packets to compete for service. The value of s_m is established as shown in Equation 7. Given that flow m packets normally would have arrived e_m seconds early in the worst case, postponing service to m does not cause m's delay bound, Δ_m, to be violated. The purpose of the minimum earliness parameter is to assist the server in lengthening its service quantum. The details of this approach will be explained later in this section

$$
s_m = t_0 + e_m \tag{7}
$$

A particular flow's starting and finishing times form a monotonically increasing sequence during each flow m active period. Many schemes emulating GPS compute the starting time of a flow's arriving packet to equal that packet's arrival time unless the packet arrived prior to the expected departure time of the flow's previous packet. However, the QFQ server's approach differs in that it computes packet starting time as the maximum of s_m and the expected finishing time of the flow's previous packet. In this computation, s_m denotes the latest time the server can begin servicing the packet without potentially violating one of it QoS commitments.

To illustrate how a QFQ server computes packet starting times, assume p_m^n refers to the nth flow m packet and a_m^n denotes its arrival time. Finally, let $\mathcal{L}_m^n$ denote the length of p_m^n. Given this, p_m^n's starting time, S_m^n, and its finishing time, F_m^n, are computed as:

$$
S_m^n = \begin{cases} V(a_m^n) + e_m & \text{if } (a_m^n > F_m^{n-1}) \\ F_m^{n-1} - \frac{S_{max}}{C} & \text{otherwise} \end{cases} \tag{8}
$$

$$
F_m^n = S_m^n + \frac{\mathcal{L}_m^n}{\mathcal{E}_m} + \frac{S_{max}}{C} \tag{9}
$$

where $F_m^0 \to -\infty$.

The QFQ service definition of S_m^n denotes p_m^n's GPS starting time moved forward in virtual time by e_m seconds. The equation for S_m^n when a_m^n is less than or equal to F_m^{n-1} accounts for the fact that the finishing time definition gives the worst case finishing time which is at most $\frac{S_{max}}{C}$ later than the GPS finishing time. Because successive packets from each flow are serviced in order and finishing times of successive packets from a given flow are monotonically increasing, the actual sarting time, $\hat{p}_m^n$, p_m^n can begin to compete for service is given by:

$$\hat{p}_m^n = \max(S_m^n, f_m^{n-1}), where$$
$$\left(S_m^{n-1} + \frac{\mathcal{L}_m^{n-1}}{C}\right) \le f_m^{n-1} \le F_m^{n-1} \quad (10)$$

where f_m^n denotes the value of the virtual clock at the actual time p_m^{n-1} completed service. The leftmost term in the inequality bounding f_m^{n-1} denotes the earliest time at which packet p_m^{n-1} may depart the server. This value is determined by adding p_m^{n-1}'s service time to its starting time, S_m^{n-1}. The right hand side of the inequality is p_m^{n-1}'s worst case finishing time. Recall that, S_m^n is less than F_m^{n-1} as F_m^{n-1} must account for potential service unfairness which may result from the discrete nature of the server. Thus, the equation illustrates the earliest time p_m^n may begin service is, indeed, the greater of S_m^n and f_m^{n-1}.

4.3   QFQ service policy

With the definition of each packet's starting and finishing time, we will now discuss the QFQ server's service policy. The server begins by selecting the flow, c_1, which is to be serviced next. To be selected, a flow must have the minimum finishing time of all those flows which are eligible at the time of the scheduling decision. By eligible it is meant that the packet at the head of the flow's service queue has a starting time less than or equal to the current virtual time. After selecting c_1, the server selects another flow, c_2, having the minimum starting time from all the unselected flows. If c_1 is the only active flow, c_2 and its starting time $S_{c_2}^{H_{c_2}(t)}$ are both set to infinity.

QFQ Server Algorithm
Step 1: while $\mathcal{A}(t) = \emptyset$
$\qquad c_2 = -\infty$;
$\qquad$ processor_status = idle;
Step 2: Let $c_1 = \{\min\ c \mid S_c^{H_c(t)} \le \mathcal{V}(t)$ and $F_c^{H_c(t)} \le F_j^{H_j(t)}\} \ \forall \ j$;

Step 3: Select flow with minimum start time
$$c_2 = \begin{cases} \infty, \text{if } \mathcal{A}(t) = \{c_1\}, \text{else} \\ \{\min\ c \ne c_1 \mid S_c^{H_c(t)} \le S_j^{H_j(t)} \ \forall j \ne c_1\} \end{cases}$$
Step 4: if $(c_2 = \infty)$ $S_{c_2}^{H_{c_2}(t)} = \infty$
Step 5: Let start $= \mathcal{V}(t)$;
Step 6: Let num_tx $= 0$ $\qquad$;
Step 7: do
$\qquad$ a.) $size_i = \mathcal{L}_{c_1}^{H_{c_1}(t)}$
$\qquad$ b.) num_tx $+= size_i$
$\qquad$ c.) Service $\ H_{c_1}(t)$
$\qquad$ d.) Update $\ \mathcal{V}(t + \frac{size_i}{C})$
$\qquad$ while $((Q_{c_1}(t) > 0)$ and
$\qquad\qquad (S_{c_2}^{H_{c_2}(t)} \ge (start + \frac{num\_tx + \mathcal{L}_{c_1}^{H_{c_1}(t)}}{C})))$
Step 8: Goto 1

Examining the QFQ server pseudocode, notice that the flow c_1 packet referred to in the while statement conditional of Step 7 is not the one transmitted in Step 7c. The $\mathcal{L}_{c_1}^{H_{c_1}(t)}$ term in the while conditional refers to the length of the packet which was subsequently queued in flow c_1's service queue. As coded, a minimum of one packet from c_1 is serviced with each selection. However, the QFQ server attempts to stretch c_1's service quantum by continuing to service c_1's packets until either its service queue becomes empty or servicing the next packet will prevent flow c_2 from competing for service when it should. The use of e_m in computing starting times assists the server in lengthening the service quantum by denoting the latest time a packet must begin competing for service without jeopardizing the flow's QoS guarantee. Thus, the QFQ server is able to reduce the number of scheduling decisions made to every τ units of time where τ is dependent upon the size of the selected flow's service queue and the QoS requirements of the other active flows. In contrast, schemes seeking to emulate GPS must make scheduling decisions after each packet transmission.

The effect of lengthening a flow's service quantum is twofold. First, it reduces the number of scheduling decisions which must be made for a given workload. The second effect is that flows may experience greater service unfairness in comparison to GPS emulation schemes. Multipacket service quanta cause the server to get ahead of a GPS server with respect to the flow receiving service. Likewise, the server may fall behind a GPS server with respect to the other flows not being serviced. However, because the selected flow's quantum is determined based upon the QoS requirements of all active flows, QoS-fair service quanta do not interfere with the guarantees of unselected flows. Consequently, the service flexibility and reduce overhead facilitated by QoS-fair service

quanta do not negatively impact the on-time reliability of a flow's packets since the QFQ discipline seeks only to exploit active flows' delay tolerances rather than imposing additional delay. This fact shall be developed further in the next section where the approach is validated with proofs of worst-case delay and throughput support.

4.4   QFQ regulator

Because all flow's may not be active at the time of the scheduling decision, it may be possible for a new flow to become active prior to the end of the computed scheduling quantum defined for c_1. If this new flow's starting time is less than c_2's, the quantum previously computed for c_1 can be too long.

To compensate for this, the QFQ server employs a regulator process. The process is capable of handling packet arrivals, updating virtual time and, potentially updating c_2 if required.

QFQ Regulator Algorithm
Let p_{new}^n be the arriving packet from flow new
Step 1: Update $\mathcal{V}(t + \tau)$
Step 2: Compute S_{new}^n
Step 3: Compute F_{new}^n
Step 4: If $(c_2 \neq -\infty)$
    $(S_{new}^n < S_{c_2}^{H_{c_2}(t)})$? $c_2 = new : c_2 = c_2$;

With each packet arrival, the server regulator updates virtual time, computes the packet's starting and finishing times and determines whether or not to update c_2. If c_2 equals negative infinity, indicating the server is idle, c_2 need not be modified as the new flow is the only active flow. Otherwise, the value of c_2 is set to the starting time of the flow having the minimum starting time of all active flows. As the value of c_2 identifies the unselected flow with the minimum starting time of the currently active set, the QFQ regulator need only compare the newly active flows starting time with that of c_2's starting time. Recall too that if the server is active with only one active flow, $S_{c_2}^{H_{c_2}(t)}$, is set to positive infinity. Thus, in this case, c_2 will take on the identity of the new flow and the conditional in the server algorithm will be updated appropriately. Finally, because the service loop requires at most $\frac{S_{max}}{C}$ seconds per iteration, the QFQ server checks for updates to c_2 at least every $\frac{S_{max}}{C}$ seconds as well. Thus, the lack of information at the time c_2 was first computed does not cause the server's worst case delay bounds to increase beyond its minimum.

4.5   Scheme validation

In this section, we show that the use of e_m to offset the starting time of flow m packets does not interfere with the QFQ server's ability to satisfy the QoS requirements of the supported flows. As a result, we show first that the computation of the packet starting and finishing times allows sufficient time to service the packet. Next, we show that given a burst from flow m of the maximum size, σ_m, the server has sufficient time to service the selected flow's burst by the delay bound, Δ_m.

4.5.1   Validation of S_m^n and F_m^n values

Here we show that with the computation of S_m^n and F_m^n, sufficient time is provided to service each arriving packet from flow m. Before beginning the proof, however, note that if a flow's backlog consists of more than one packet, only the first packet of the burst may arrive too late to be scheduled correctly. To understand this, recall subsequent packets begin to compete for service at the later of the previous packet's actual finishing time, f_m^1, and their starting time, S_m^n. This implies that a backlog can only occur when packets p_m^n arrive prior to F_m^{n-1} for $(n > 1)$.

In addition, recall that given a maximum flow m burst of size σ_m, the sum of the excess delay introduced by a discrete emulation server onto each of the packets is bound by $\frac{S_{max}}{C}$. Thus, we would like to first show that the time between each flow's starting and finishing time allows for sufficient time to service the packet currently at the head of the flow m queue. We do this by showing sufficient time is provided between p_m^1's starting and finishing time and then show the same to be true for p_m^n in general.

Lemma 3 *The value $(F_m^1 - S_m^1)$ equals $\frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C}$ which allows the QFQ server to service packet p_m^1 at its guaranteed rate $\mathcal{E}_m$ and suffer its worst case unfairness; where p_m^1 denotes the packet at the head of the flow m service queue.*

Proof. To prove this lemma we show:

$$(F_m^1 - S_m^1) \leq \left(\frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C} \right)$$

Re-writing F_m^1, the following equation is obtained.

$$\left(\left[S_m^1 + \frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C} \right] \right) - S_m^1 \leq \left(\frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C} \right)$$

Simplifying both sides, we obtain:

$$\left(\frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C} \right) \leq \left(\frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C} \right)$$

which is true by identity. To prove the lemma holds for the more general case of $(n > 1)$ we need only show:

$$\left(F_m^n - S_m^n\right) \leq \left(\frac{\mathcal{L}_m^n}{\mathcal{E}_m} + \frac{S_{max}}{C}\right)$$

which simplifies to:

$$\left(\left[S_m^n + \frac{\mathcal{L}_m^n}{\mathcal{E}_m} + \frac{S_{max}}{C}\right] - S_m^n\right) \leq \left(\frac{\mathcal{L}_m^n}{\mathcal{E}_m} + \frac{S_{max}}{C}\right)$$

which is true as well.

4.5.2 Burst validation

Continuing, we seek to show that the overlap of the S_m^n and F_m^{n-1} values does not prevent the server from fulfilling its service commitments to real-time flows. Further, we must demonstrate that offsetting the starting time of a burst by e_m seconds does not cause a flow's delay bound to be violated. Hence, we must prove the following:

$$\forall m \ \ \forall n \ \ \left(F_m^n - \mathcal{V}(a_m^n)\right) \leq \Delta_m$$

The above equation states that each packet from each flow suffers a delay less than or equal to its permissible queuing delay bound.

To set up this proof, let us assume the worst case where there are σ_m instantaneous packet arrivals from a given flow m over an infinite capacity link. Assume the arrivals are numbered $p_m^1 \ldots p_m^{\sigma_m}$. We first show in Case 1 that p_m^1 suffers a queuing delay which is bounded by Δ_m, the flow's tolerable delay bound. Next, we show in Case 2 that $p_m^{\sigma_m}$ suffers a queueing delay that is bound by Δ_m as well. Since intervening packets from flow i all suffer a delay greater than p_m^1 and less than $p_m^{\sigma_m}$ the assertion is shown to hold.

Theorem 2 *Assuming flow m is guaranteed a service rate of $\mathcal{E}_m$, each flow m packet experiences a delay which is upper bound by m's delay bound, Δ_m. In other words, we show:*

$$\forall m \ \ \forall n \ \ \left(F_m^n - \mathcal{V}(a_m^n)\right) \leq \Delta_m$$

Case 1- $Q_m(a_m^1) = 0$(i.e., n = 1): In this case, we show that the delay suffered by the first packet in a flow m burst is bound by flow m's delay bound Δ_m. To do this, we prove:

$$F_m^1 - \mathcal{V}(a_m^1) \leq \Delta_m$$

$$\left[S_m^1 + \frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C}\right] - \mathcal{V}(a_m^1) \leq \Delta_m$$

$$\left[\left[\mathcal{V}(a_m^1) + e_m\right] + \frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{R}\right] - \mathcal{V}(a_m^1) \leq \Delta_m$$

$$\left[\left[\mathcal{V}(a_m^1) + \Delta_m - \frac{\sigma_m}{\mathcal{E}_m} - \frac{S_{max}}{R}\right] + \frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{R}\right] - \mathcal{V}(a_m^1) \leq \Delta_m$$

$$\frac{\mathcal{L}_m^1 - \sigma_m}{\mathcal{E}_m} \leq 0$$

$$\mathcal{L}_m^1 \leq \sigma_m$$

which is true by definition of the leaky-bucket constraint all real-time flows m are expected to obey.

Case 2- $Q_m(a_m^1) <> 0$(i.e., $n = \sigma_m$): In this case, we complete the proof by showing that the delay suffered by the last packet in a flow m burst is also bound by flow m's delay bound Δ_m. To do this, we prove:

$$F_m^{\sigma_m} - \mathcal{V}(a_m^{\sigma_m}) \leq \Delta_m$$

$$\left[F_m^1 + \sum_{j=2}^{\sigma_m} \frac{\mathcal{L}_m^j}{\mathcal{E}_m}\right] - \mathcal{V}(a_m^{\sigma_m}) \leq \Delta_m$$

$$\left[\left(S_m^1 + \frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C}\right) + \sum_{j=2}^{\sigma_m} \frac{\mathcal{L}_m^j}{\mathcal{E}_m}\right] - \mathcal{V}(a_m^{\sigma_m}) \leq \Delta_m$$

$$\left[\left(\left(\mathcal{V}(a_m^1) + e_m\right) + \frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C}\right) + \sum_{j=2}^{\sigma_m} \frac{\mathcal{L}_m^j}{\mathcal{E}_m}\right] - \mathcal{V}(a_m^{\sigma_m}) \leq \Delta_m$$

$$\left[\left(\left(\mathcal{V}(a_m^1) + \Delta_m - \frac{\sigma_m}{\mathcal{E}_m} - \frac{S_{max}}{C}\right) + \frac{\mathcal{L}_m^1}{\mathcal{E}_m} + \frac{S_{max}}{C}\right) + \sum_{j=2}^{\sigma_m} \frac{\mathcal{L}_m^j}{\mathcal{E}_m}\right] - \mathcal{V}(a_m^{\sigma_m}) \leq \Delta_m$$

$$\underbrace{\left[\mathcal{V}(a_m^1) - \mathcal{V}(a_m^{\sigma_m})\right]}_{1} + \underbrace{\left[\left(\sum_{j=1}^{\sigma_m} \frac{\mathcal{L}_m^j}{\mathcal{E}_m}\right) - \frac{\sigma_m}{\mathcal{E}_m}\right]}_{2} \leq 0$$

1. As the virtual arrival times of the packets in m's burst form a monotonically increasing function,

$$\mathcal{V}(a_m^1) \leq \mathcal{V}(a_m^{\sigma_m}) \implies \left[\mathcal{V}(a_m^1) - \mathcal{V}(a_m^{\sigma_m})\right] \leq 0$$

2. The leaky-bucket arrival constraint implies:

$$\left(\sum_{j=1}^{\sigma_m} \mathcal{L}_m^j\right) \leq \sigma_m \implies \left(\sum_{j=1}^{\sigma_m} \frac{\mathcal{L}_m^j}{\mathcal{E}_m}\right) - \frac{\sigma_m}{\mathcal{E}_m} \leq 0$$

As the value of both the 1st and 2nd terms in the LHS of the above equation have been shown to each have a non-positive value, their sum is clearly non-positive and the inequality is proven true. Further, since the server's virtual time moves as fast or faster than real-time, the bound is shown to hold in real-time as well as virtual time.

4.6 Service Enhancements

As described so far, the scheme has addressed the issue of loosening the coupling of rate and delay guarantees. Despite having a guaranteed rate lower bound by the average rate and a traffic arrival function which normally would imply a flow's worst case delay would be less than its specified delay bound, the approach presented attempts to provide applications with an actual delay equal to their specified delay bound. By the Law of Conservation [Kleinroock 1976], this reduction in the service provided to real-time applications necessarily results in a improvement in the service provided to best-effort applications in the form of reduced average delay given the server's work conserving nature.

The current specification, however, only reduces the average delay for best-effort traffic. Over-subscribed best-effort traffic must still compete with real-time traffic for service. This is a consequence of the scheme pushing the starting time of over-subscribed best-effort traffic back to its expected time. The reason for the starting times being pushed back is so that the non-compliant traffic does not interfere with the guarantees provided to other flows sharing the server. While correct, this approach is unnecessarily restrictive. Instead of advancing the virtual clock to the minimum starting time of any back-logged packet when no packets have starting times less than the current virtual clock value, the server should attempt to service any best-effort traffic (regardless of its starting time) during the ϵ units of time until the next starting time. The value of ϵ may be defined as:

$$\epsilon = min_{i \in \mathcal{B}(t)} S_i^{H_i(t)} - \mathcal{V}(t)$$

Doing this requires that best-effort traffic be considered eligible immediately upon its arrival. The effect of this is to potentially increase the number of flows competing for service. Nevertheless, recall that the QFQ server's priority decision is made with respect to the minimum finishing time rather than the starting time. As such the service guarantees provided by the QFQ server will not be altered. However, also note that while it may be useful to consider best-effort traffic eligible upon arrival, this should not interfere with the efforts to increase the scheduling quantum and reduce overhead.

To this end, an additional variable, $\dot{S}_m^n$, is introduced to denote p_m^n's earliest allowed starting time, while S_m^n denotes the latest allowed starting time. For all real-time flows, i, $\dot{S}_m^n$ equals S_m^n for all m, where S_m^n is computed as shown earlier in equation 8. For best-effort flows, i, S_m^n is computed as normal, but $\dot{S}_m^n = \mathcal{V}(a_m^n)$ for all k.

$$\dot{S}_m^n = \begin{cases} \mathcal{V}(a_m^n) & \text{if } i \text{ is a best-effort flow} \\ S_i^k & \text{otherwise} \end{cases}$$

With this definition, the server can use $\dot{S}_m^n$ to determine if a packet is eligible for transmission and the desired effect is achieved. Notice that because the computation of F_m^n is still dependent upon S_m^n and not $\dot{S}_m^n$, the minimal service guarantees for all flows can still be maintained.

4.6.1 Algorithm revisions

Before showing the revised server algorithm, we must modify the calculation of the virtual time function. The modification of the virtual time function uses $\dot{S}_m^n$ instead of S_m^n which benefits the best-effort flows since this value is unchanged for real-time ones. This modification is shown below:

$$\mathcal{V}(t) = \text{real time, for all } t \text{ where } \mathcal{B}(t) = \emptyset$$
$$\mathcal{V}(t + \tau) = max(\mathcal{V}(t) + \tau, min_{i \in \mathcal{B}(t+\tau)} \dot{S}_i^{h_i(t+\tau)})$$

In the server pseudocode below, Step 2 of the algorithm is modified to use $\dot{S}_m^n$ instead of S_m^n. Note, that step 3 and the conditional in the *do..while* loop depicted in Step 7 both still use S_m^n to allow for larger service quanta. This approach is valid as S_m^n is the time which flow i must begin competing for service to ensure its deadline is not violated.

Revised QFQ Server Algorithm
Step 1: while $\mathcal{A}(t) = \emptyset$
$$c_2 = -\infty \text{ ;}$$
idle;
Step 2: Let $c_1 = \{min \ c \mid \dot{S}_c^{H_c(t)} \leq \mathcal{V}(t) \text{ and } F_c^{H_c(t)} \leq F_j^{H_j(t)}\} \ \forall \text{ j};$
Step 3: Select flow with minimum start time
$$c_2 = \begin{cases} \infty, \text{if } \mathcal{A}(t) = \{c_1\}, \text{else} \\ \{min \ c \neq c_1 \mid S_c^{H_c(t)} \leq S_j^{H_j(t)} \ \forall j \neq c_1\} \end{cases}$$
Step 4: if $(c_2 = \infty) \ S_{c_2}^{H_{c_2}(t)} = \infty$
Step 5: Let start $= \mathcal{V}(t)$
Step 6: Let num_tx $= 0$
Step 7: do
$$size_i = \mathcal{L}_{c_1}^{H_{c_1}(t)}$$
num_tx $+= size_i$
Service $H_{c_1}(t)$
Update $\mathcal{V}(t + \frac{size_i}{C})$
while $((Q_{c_1}(t) > 0)$ and
$$(S_{c_2}^{H_{c_2}(t)} \geq (start + \frac{num\_tx + \mathcal{L}_{c_1}^{H_{c_1}(t)}}{C})))$$
Step 8: Goto 1

Lastly, the QFQ server's regulator process must be modified to calculate the value of $\dot{S}_m^n$. The reader

should note that the decision of whether or not to update c_2 depends on the value of S^k_{new} and not $\dot{S}^k_{new}$; as S^k_{new} denotes the time p^n_m must should begin competing for service to ensure its delay constraint is met, while $\dot{S}^k_{new}$ only denotes the time it may begin competing for service.

Revised QFQ Regulator Algorithm
Step 1: Let p^k_{new} denote the newly arriving packet.
Step 2: Update $\mathcal{V}(t + \tau)$
Step 3: Compute S^k_{new}
Step 4: Compute $\dot{S}^k_{new}$
Step 5: Compute F^k_{new}
Step 6: $(S^k_{new} < S^{h_{c_2}(t)}_{c_2})$? $c_2 = new : c_2 = c_2;$

Finally, it should be clear that the service enhancements mentioned in this section can be performed with a minimal amount of added state to maintain. Namely, the server must compute $\dot{S}^n_m$ and provide storage for this value. The other algorithm's are only updated as needed to have them refer to $\dot{S}^n_m$ or S^n_m, according to their usage.

5 Simulation and Analysis

To demonstrate the capabilities of the QFQ discipline's approach provides, we implemented several ISN scheduling disciplines using the $CSIM$ discrete-event simulation package. A representative example of the results of the simulation trials conducted is presented below. The results depicted compare and contrast the performance of the QFQ service discipline with that of $FIFO$, a dual static priority scheme referred to as $DUALQ$, WFQ and WF^2Q. The $DUALQ$ scheme functioned by placing all real-time traffic onto queue zero, while all best-effort traffic was placed onto queue one. In a work-conserving manner, the $DUALQ$ scheduler then selected the packet from the lowest numbered non-empty queue.

T	N	RT	AR	PR	MBS	DB
Cst	10	Y	4 c/s	4 c/s	1	0.180 s
B-1	50	Y	1 c/s	200 c/s	50	0.078 s
B-2	50	Y	1 c/s	200 c/s	60	0.075 s
B-3	50	Y	1 c/s	200 c/s	150	0.180 s
B-4	1	N	2 c/s	200 c/s	100	300 s

Table 1: Workload Profile

The server in this simulation was configured to transmit cells at a rate of two-hundred thousand cells per second where a cell was defined to be 424 bits in length. In addition the server was configured with

sixteen thousand buffers which is equivalent to 848 kilobytes. Based upon these values, a cell transmission time equaled five microseconds. The simulated workload consited of consisted of a mix of constant rate real-time flows (Cst), bursty real-time flows (B-1, B-2 and B-3) as well as a bursty best-effort application (B-4). The simulated workload is described in Table 1, where T denotes the flow type, N denotes the number of flow requests of that type which were in the workload, and the term c/s denotes **1000** cells per second. For each flow, constant (Cst), real-time (RT) or best-effort, is characterized by its average rate (AR), its peak rate (BR), its maximum burst size (MBS), and its delay bound (DB). The latter parameter denotes the maximum queuing delay tolerable by the flow at the node being studied. Bursty flows in the workload, including the best-effort one, transmitted was modeled as an interrupted Poisson process, where the means of the "on" and "off" times, denoted Avg_On_Time and Avg_Off_time respectively, are listed in Tables 2, milliseconds.

T	Avg_On_Time	Avg_Off_time
B-1	0.250 ms	49.9875 ms
B-2	0.300 ms	59.9820 ms
B-3	0.750 ms	149.8875 ms
B-4	0.500 ms	49.9750 ms

Table 2: Bursty Application Profiles

In the first simulation trial which, along with each of the others, lasted for 100 seconds of simulated time, the QFQ server's performance is contrasted with the serviced provided by a $FIFO$ service discipline. To accept the real-time flow requests, the flow's delay bound must be less than the worst-case queuing delay bound supported by the $FIFO$ server. In this trial, the bound was computed to be 79.995 milliseconds which is obtained by dividing the total number of buffers minus one by the server's maximum rate. On the other hand, the QFQ server used the worst-case delay bound as computed by Equation 5 given that the guaranteed rate was determined by the flow's average rate specification. Best-effort flows were automatically accepted by both servers. As a result, the QFQ server was able to accept all 161 flow requests while the $FIFO$ server accepted only 61 requests. Because of the $FIFO$ server's strict service model, it could not accept any of the calls of type B-1 or B-2. Once accepted, both servers guaranteed that no packets which conformed to its flow's traffic specification would be dropped. This fact was confirmed in the simulation data as all accepted flows had an on-time reliability percentage of one.
In Figure 1, the end-to-end delay performance of the two servers are compared. It can be observed that

the traffic under the *QFQ* discipline generally suffered higher end-to-end delay values than did traffic serviced by the *FIFO* sever. However, this is to be expected since the *QFQ* node was supporting one hundred more flows than the *FIFO* node. The intermediate points on the delay curve for the *FIFO* node where the delay was equal to zero resulted from the flows whose flow requests were rejected.
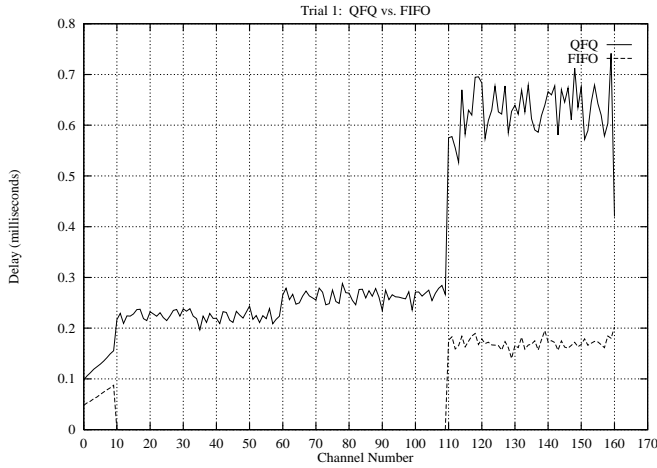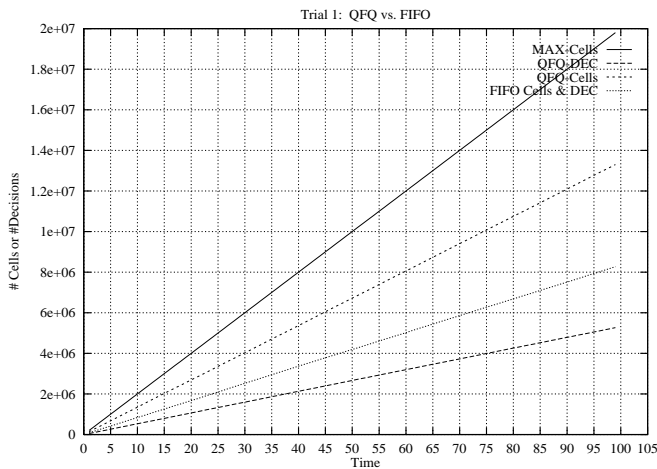

Figure 1: QFQ vs. FIFO Delay Comparison

Figure 2: QFQ vs. FIFO Quantum Comparison

Scheme	Num Synch. Points	Avg. Cells per Quantum	Num Late
QFQ	5,269,072	2.52	0
FIFO	8,267,565	1	0
DUALQ	10,767,751	1	0
WFQ	13,233,412	1	0
WF^{2}Q	13,253,917	1	0

Table 3: Quantum Table

Figure 2 depicts the performance of the two disciplines in terms of the number of *service synchronization points* and the number of cells transmitted per unit time. The abbreviation DEC is used in the subsequent graphs to refer to the number of scheduling decisions. As the *FIFO* discipline services one cell per *service synchronization point*, its two curves are identical. The curve labeled "Max-Cells" denotes the server's maximum attainable throughput rate. As can be seen from the graph, the *QFQ* server was able to service multiple cells between *service synchronization points* without violating the *QoS* requirements of any of the flows.
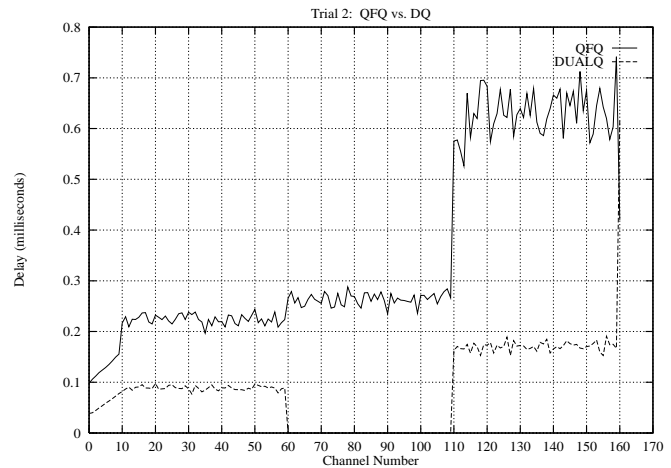

Figure 3: QFQ vs. DUALQ Delay Comparison

Table 3 indicates that the average number of cells transmitted per quantum was 2.22. As opposed to being limited by the *QoS* requirements of the active flows, this number was lower the maximum due to the selected flows' service queues becoming empty prior to the expiration of the quantum. This fact is supported by the difference between the actual server output as compared to the maximum output in terms of cells serviced per second.

In the next trial, the *QFQ* server's performance was contrasted with that of a *DUALQ* server. In this trial, five percent of the buffers (equivalent to the proportion of the best-effort flow's desired average rate compared to the server's rate) were set aside for best-effort traffic while the remaining ninety-five percent were reserved for real-time flows. This equated to 15,200 buffers being reserved for real-time flows while 800 were set aside for best-effort traffic. As a result, real-time flows were accepted only if they tolerated a queuing delay of 75.995 milliseconds. In Figure 3, it can be observed that the average end-to-end delay under the *DUALQ* scheme was again lower

than under the QFQ scheme. However, this is again expected since the scheme was unable to accept any calls of type B-2. These rejected call requests are reflected in the $DUALQ$ scheme's delay curve values of zero for flows 60 through 109. Figure 4 illustrates that although that the $DUALQ$ server serviced fewer cells than the QFQ server, it required more *service synchronization points*. With this, it seems that the added delay and complexity of the QFQ server in comparison to both the $FIFO$ and $DUALQ$ servers seems worthwhile given that fewer scheduling decisions are required and a larger number of flows can be accepted.

Lastly, the WFQ and WF^2Q simulations had similar results due to the similarity of their service algorithms. While all three schemes, WFQ, WF^2Q and QFQ, accepted the same number of calls and service generally the same number of cells, Figure 5 show that the QFQ server required significantly fewer *service synchronization points* than did either the WFQ or WF^2Q servers. This is especially useful given that these results were achieved under moderate loading while it can be expected that the average quantum size would increase with heavier loading given the same QoS requirements. As efficiency becomes of greater concern with increased utilization, this behavior seems desirable for high speed implementations.

the QFQ server is less able to exhaust the service queues of flows with larger maximum burst sizes. In addition, the lower frequency with which the QFQ server visits service queues contributes to this result as well. In contrast, the relatively regular service provided by the WFQ and WF^2Q servers facilitates lower average delays for its supported flows. One notable exception involves the constant rate flows 1 through 9 which, despite their small burst sizes, the WFQ and WF^2Q servers provide with a lower average delay than the QFQ server.

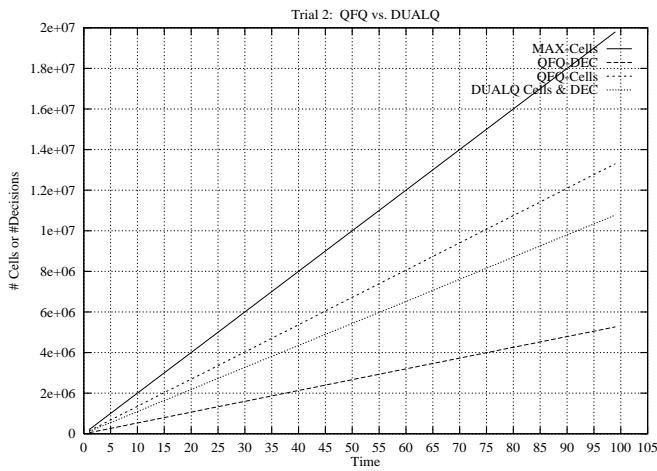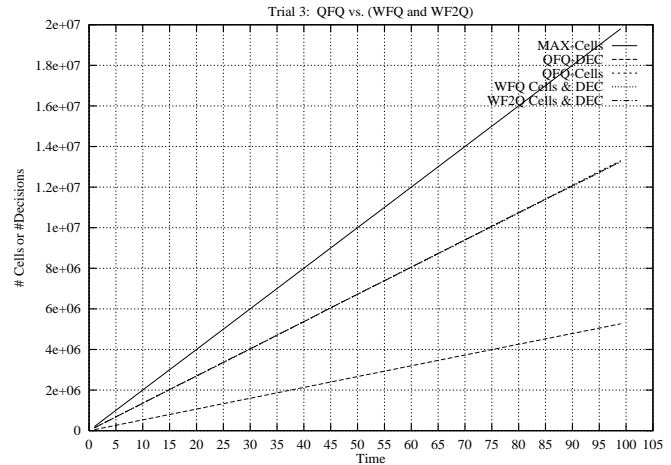Figure 5: Service Synchronization Overhead

Figure 4: QFQ vs. DUALQ Quantum Comparison

In reference to the delay performance, the simulation results are depicted in Figure 7 and Figure 6. In generally, the results show that the smaller a flow's maximum burst size, the better the QFQ server's performance was due to the increased likelihood of the server exhausting the flow's queue when the flow is selected. However, with increasing maximum burst sizes, the WFQ and WF^2Q schemes provide lower average delays. This can be attributed to the fact that
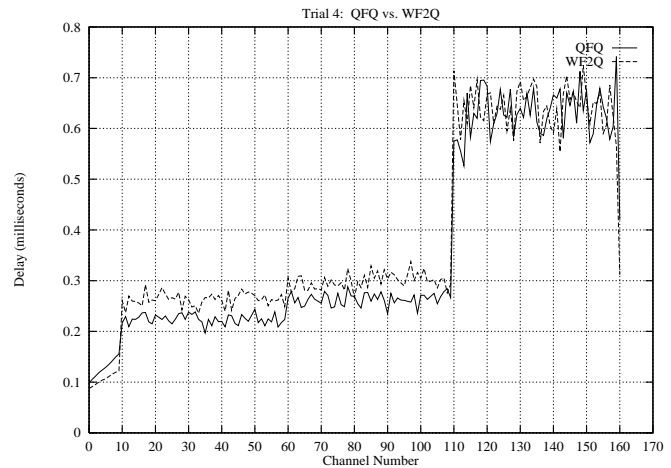

Figure 6: Delay Performance - WFQ vs. WF^2Q

Again this seems to be a factor of the extended service quantum used by the QFQ server which stretches the time between server visits to the flow's queues. Despite the differences in average delay, however, all flow's conforming traffic was provided an on-time reliability percentage of one which lends credibility to the QFQ approach of widening the ser-

vice deviation while sufficiently bounding it to ensure *QoS* guarantees and reduce scheduling overhead.

6 Conclusion

In summary, this paper began by noting that the strict model of fair resource sharing defined in the *GPS* service model is not necessary in *ISN* environments. It was stated that an application's actual *QoS* should not increase, by definition, when the active applications' service requirements do not consume all the server's capacity. As the application and the network's admission control mechanism are usually unable to exploit such service improvements, the benefits of this approach are usually small. Lastly, we stated that while the accuracy of a *GPS* emulation server increases with increases in link speed given a constant packet size, the overhead encumbered by this server increases as well. As a result, we concluded that the packet-by-packet scheduling decisions prescribed by *GPS* emulation schemes were not needed since application *QoS* not *GPS* accuracy, should be the packet scheduling mechanism's main focus.

Further, it is stated that a packet scheduling mechanism's approach should first ensure application *QoS* guarantees can be fulfilled and then use any excess capacity to reduce service overhead and service best-effort on other types of non-guaranteed traffic. In response, a new service model, referred to as *QFS*, is described. The *QFQ* model defines a server which is required to only provide applications with the *QoS* requested or set aside for them at call-setup time. Because the *QFQ* server has only a flow's *QoS* requirements to satisfy, rather than a stricter fair share as defined by *GPS*, it has a high degree of flexibility in determining how to allocate excess service capacity. In this paper, the allocation policy presented sought first, to reduce overhead and secondly, to enhance the service provided to best-effort applications. As best-effort traffic can be over-subscribed and backlogged, the resulting service gain can prove extremely beneficial. Thus, while continuing to meet the application throughput and delay guarantees, the *QFQ* server is able to improve the level of service received by best-effort applications. In addition, where the strict *GPS* model requires an emulation server to make scheduling decisions on a packet-by-packet basis, the *QFQ* service model introduces scheduling decisions only as required by the active flow's *QoS* guarantees and the run-time traffic load. As a result, fewer scheduling decisions are performed and, thus, the server's overhead is lower .
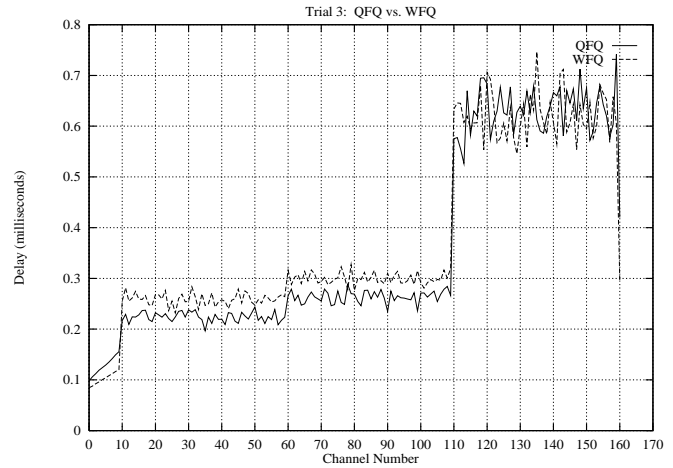

Figure 7: Delay Performance - *QFQ* vs. *WFQ*

The ability to support per-flow management is critical in order to efficiently address network congestion and end-to-end QoS guarantees. Although viable within an edge router in a private network, the complexity of managing resources on a per-flow basis may be prohibitive within a backbone router, where the number of flows can be very high. Aggregate mechanisms to group traffic into different classes have been proposed to addresses the lack of scalability of integrated service models. Differentiated service models require significantly less state information and processing power, but create behavioral dependencies among flows within the same aggregate class. There is a growing consensus, however, that in general per-flow signaling can be combined with aggregate traffic handling to develop QoS service models that offer the robustness and high quality guarantees provided by integrated service models, while at the same time supporting the scalability provided by differentiated service models. In future work, we intend to extend the *QFQ* framework to support traffic aggregation and differentiation. In the new framework, private networks will support per-flow management service models to achieve stronger QoS guarantees, while backbone networks connecting these private networks support differentitated service models and rely on routers ingress to the differentiated service network to classify packets. Queues within the differentiated service network can then be configured to support specific QoS requirements, such as delay bounds and minimum bandwidth. Furtheremore, routers must be able to re-negotiate delays when, due to congestion or node failure along the flow's routing path, the network is no longer able to continue supporting the requested service.

number CISE/ANIR 0073972.

References

J.C.R. Bennett and H. Zhang, "WF^{2}Q: Worst-Case Fair Weighted Fair Queuing", *Proceedings of INFOCOM '96*, March 1996.

J.C.R. Bennett and H. Zhang, "Why WFQ Is Not Good Enough for Integrated Services Networks?", *Proceedings of NOSSDAV '96*, April 1996.

D. C. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", *SIGCOMM '92*, pp. 14–26, 1992.

R. L. Cruz., "Service Burstiness and Dynamic Burstiness Measures: A Framework", *Journal of High Speed Networks*, 2:105–127, 1992.

N.G. Duffield, T.V. Lakshman, and D. Stiliadis, "On Adaptive Bandwidth Sharing with Rate Guarantees", *Proceedings of Infocom 1998*, 1998.

S. Jamaloddin Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications", *Proc. IEEE INFOCOM*, pages 636–646, 1994.

P. Goyal and H. M. Vin, "Fair Airport Scheduling Algorithms", *Proceedings of NOSSDAV '97*, St. Louis, Missouri, May 1997.

Leonard Kleinrock,"Queueing Systems Volume 2: Computer Applications", *John Wiley, New York, 1976*.

Simon Lam and Geoffrey Xie, "Burst Scheduling Networks: Flow Specification and Performance Guarantees", *Proceedings of NOSSDAV '95*, Durham, New Hampshire, April 1995.

Cathy McCann, Raj Vaswani, and John Zahorjan, "A Dynamic Processor Allocation Policy for Multiprogrammed Shared Memory Multiprocessors", *ACM Transactions on Computer Systems*, pages 146–178, 1993.

A. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks", *PhD thesis, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology*, February 1992.

Abhay K. Parekh and Robert G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case", *IEEE/ACM Trans. on Networking*, 1(3):344–357, June 1993.

Abhay K. Parekh and Robert G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case", *IEEE/ACM Trans. on Networking*, 2(2):137–150, April 1994.

Subhash Suri, George Varghese, and Girish Chandranmenon, "Leap Forward Virtual Clock: A New Fair Queueing Scheme with Guaranteed Delays and Throughput Fairness", Technical Report WUCS-96-10, Washington University in St. Louis, June 1996.

Anujan Varma and Dimitrios Stiliadis, "Design and Analysis of Frame-Based Fair Queueing: A New Traffic Scheduling Algorithm for Packet Switched Networks", *Proc ACM SIGMETRICS*, May 1996.

Geoffrey G., Xie J. and L. Simon, "An Efficient Adaptive Search Algorithm for Scheduling Real-Time Traffic", *Proceeding of ICNP '96*, Columbus, Ohio, October 1996.

Yates D., Kurose J., Towsley D., and Hluchyj M. G., "On Per-Session End-to-end Delay Distributions and the Call Admission Problem for Real-Time Applications with QoS Requirements", *Proceedings of ACM SIGCOMM 1993*, pages 2–12, September 1993.

Biographies

Michael S. Boykin Dr. Boykin obtained his Ph.D. in Computer Science at the University of Pittsburgh, Main Campus, in April 1999. He obtained his M.S. degree in 1997 in Computer Science at the University of Pittsburgh as well. His research work and publications have focused on the design of efficient packet scheduling algorithms for use in high-speed integrated services networks. His research interests also include the design of low overhead mechanisms to provide guaranteed quality of service in distributed, real-time multimedia system environments. Dr. Boykin held a variety of positions within the IT department of

a major Fortune 100 company's division. These positions involved various aspects of global, enterprise-wide distributed system deployments, network commincations protocol and distributed system analysis as well as other telecommunication-related areas. He was also part of the senior management team of a systems and network integration firm as the Professional Services Manager. The firm specialized in integrating real-time, manufacturing floor network and systems with new, enterprise-wide, ERP and CRM IT solutions for manufacturing, chemical and pharmaceutical companies. Dr. Boykin has also served as Director of Network Services for a major IT outsourcing and ASP hosting firm. In addition, Dr. Boykin has served as an adjunct faculty member and visiting lecturer in the University of Pittsburgh's Computer Science and Telecommunications departments, respectively. His courses involve network and communications protocol design, analysis and implementation.

Taieb F. Znati Prof. Znati (A' 91) (znati@cs.pitt.edu, tznati@nsf.gov). Prof. Znati obtained a Ph.D. Degree in Computer Science at Michigan State University, East Lansing, in April 1988, and a Master of Science Degree at Purdue University, West Lafayette, Indiana. In 1988, he joined the University of Pittsburgh where he currently serves as a Professor in the Department of Computer Science with a joint appointment in Telecommunications in the Department of Information Science. His current research interests focus on agent-based technology, middleware, the design of network protocols for wired and wireless communication networks to support multimedia applications' QoS requirements, the design and analysis of medium access control protocols to support distributed real-time systems, network simulation, and the investigation of fundamental design issues related to large scale distributed systems. He is frequently invited to present lectures and tutorials and participate in panels related to networking and distributed multimedia topics, in the United and abroad. He is currently on leave from the University of Pittsburgh to serve as Senior Program Director for Networking Research at the National Science Foundation.