Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks

Dimitrios Stiliadis and Anujan Varma
Computer Engineering Department
University of California
Santa Cruz, CA 95064

Abstract

In this paper we introduce and analyze *frame-based fair queueing*, a novel traffic scheduling algorithm for packet-switched networks. The algorithm provides end-to-end delay bounds identical to those of PGPS (packet-level generalized processor sharing), without the complexity of simulating the fluid-model system in the background as required in PGPS. The algorithm is therefore ideally suited for implementation in packet switches supporting a large number of sessions. We present a simple implementation of the algorithm for a general packet switch. In addition, we prove that the algorithm is fair in the sense that sessions are not penalized for excess bandwidth they received while other sessions were idle. Frame-based fair queueing belongs to a general class of scheduling algorithms, which we call *Rate-Proportional Servers.* This class of algorithms provides the same end-to-end delay and burstiness bounds as PGPS, but allows more flexibility in the design and implementation of the algorithm. We provide a systematic analysis of this class of schedulers and obtain bounds on their fairness.

I. Introduction

Providing QoS guarantees in a packet network requires the use of traffic scheduling algorithms in the switches (or routers). The function of a scheduling algorithm is to select, for each outgoing link of the switch, the packet to be transmitted in the next cycle from the available packets belonging to the flows sharing the output link. Implementation of the algorithm may be in hardware or software. In ATM networks, where information is transmitted in terms of small fixed-size cells, the scheduling algorithm is usually implemented in hardware. In a packet network with larger packet-sizes, the algorithm may be implemented in software.

Several scheduling algorithms are known in the literature for bandwidth allocation and transmission scheduling in output-buffered switches. These include the packet-by-packet version of Generalized Processor Sharing (PGPS) [1] (also known as Weighted Fair Queueing [2]), VirtualClock [3], Self-Clocked Fair Queueing (SCFQ) [4], Delay-Earliest-Due-Date (Delay-EDD) [5], Weighted Round Robin [6], and Deficit Round Robin [7]. Many of these algorithms are also capable of providing deterministic upper bounds on the end-to-end delay seen by a session when the burstiness of the session traffic is bounded (for example, shaped by a leaky bucket).

Based on their internal structure, traffic schedulers can be classified into two main types: *sorted-priority* and *frame-based* [8]. In a sorted-priority scheduler, there is a global variable — usually referred to as the *virtual time* — associated with each outgoing link of the switch. Each time a packet arrives or gets serviced, this variable is updated. A timestamp, computed as a function of this variable, is associated with each packet in the system. Packets are sorted based on their timestamps, and are transmitted in that order. VirtualClock [3], Weighted Fair Queueing [2], and Delay-EDD [5] follow this architecture. A frame-based scheduler, on the other hand, does not require a priority-queue of packets to be maintained. Instead, bandwidth guarantees are provided by splitting time into frames of fixed or variable length, and limiting the amount of traffic a session is allowed to transmit during a frame period. Examples for frame-based schedulers include Hierarchical Round Robin [9], Stop-and-Go Queueing [10], Weighted Round Robin [6] and Deficit Round Robin [7].

A traffic scheduling algorithm must possess several desirable features to be useful in practice:

1. Isolation of flows: The algorithm must isolate an end-to-end session from the undesirable effects of other (possibly misbehaving) sessions. Note that isolation is necessary even when policing mechanisms are used to shape the flows at the entry point of the network, as the flows may accumulate burstiness within the network.
2. Low end-to-end delays: Real-time applications require from the network low end-to-end delay guarantees.
3. Utilization: The algorithm must utilize the link bandwidth efficiently.
4. Fairness: The available link bandwidth must be divided among the connections sharing the link in a fair manner. An unfair scheduling algorithm may offer widely different service rates to two connections with the same reserved rate over short intervals.
5. Simplicity of implementation: The scheduling algorithm must have a simple implementation. In an ATM network, the available time for completing a scheduling decision is very short and the algorithm must be implemented in hardware. In packet networks with larger packet sizes and/or lower speeds, a software implemen-

This research is supported by the NSF Young Investigator Award No. MIP-9257103.

SIGMETRICS '96 5/96 PA, USA
© 1996 ACM 0-89791-793-6/96/0005...$3.50

tation may be adequate, but scheduling decisions must still be made at a rate close to the arrival rate of packets.

6. Scalability: The algorithm must perform well in switches with a large number of connections, as well as over a wide range of link speeds.

Weighted Fair Queueing (WFQ), also known as Packet-level Generalized Processor Sharing (PGPS), is an ideal scheduling algorithm in terms of its fairness and delay bounds, but is complex to implement because of the need to simulate an equivalent fluid-model system in the background. Timestamp computations in PGPS have a time-complexity of $O(V)$, where V is the number of sessions sharing the outgoing link. Self-Clocked Fair Queueing (SCFQ) [4] enables timestamp computations to be performed in $O(1)$ time and has fairness comparable to that of PGPS, but results in increased end-to-end delay bounds [11], [12]. The VirtualClock scheduling algorithm [3] provides the same end-to-end delay bound as that of PGPS with a simple timestamp computation algorithm, but the price paid is in terms of fairness. A backlogged session in the VirtualClock server can be starved for an arbitrary period of time as a result of excess bandwidth it received from the server when other sessions were idle [1].

Frame-based fair queueing (FFQ) is a sorted-priority algorithm, and therefore uses timestamps to order packet transmissions. However, it requires only $O(1)$ time for the timestamp calculation independent of the number of sessions sharing the server. At the same time, the end-to-end delay guarantees of FFQ are identical to those obtained from a corresponding PGPS server. In addition, the server is fair in the sense that connections are always served proportionally to their reservations when they are backlogged, and are not penalized for an arbitrary amount of time for bandwidth they received while the system was empty. The algorithm uses a framing approach similar to that used in frame-based schedulers to update the state of the system; the transmission of packets, however, is still based on timestamps.

II. Preliminaries

We assume a packet switch where a set of V connections share a common output link. The terms *connection*, *flow*, and *session* will be used synonymously. We denote with ρ_i the rate allocated to connection i.

We assume that the servers are non-cut-through devices. Let $A_i(\tau, t)$ denote the arrivals from session i during the interval $(\tau, t]$ and $W_i(\tau, t)$ the amount of service received by session i during the same interval. In a system based on the fluid model, both $A_i(\tau, t)$ and $W_i(\tau, t)$ are continuous functions of t. However, in the packet-by-packet model, we assume that $A_i(\tau, t)$ increases only when the last bit of a packet is received by the server; likewise, $W_i(\tau, t)$ is increased only when the last bit of the packet in service leaves the server. Thus, the fluid model may be viewed as a special case of the packet-by-packet model with infinitesimally small packets.

Definition 1: A *system busy period* is a maximal interval of time during which the server is never idle.

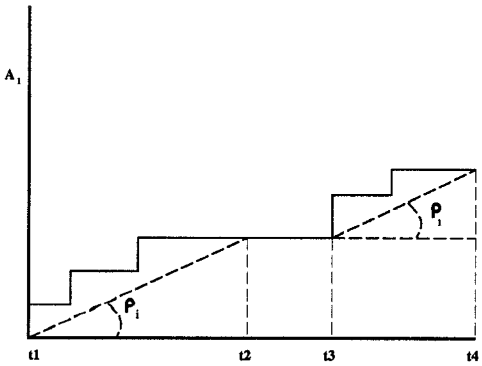

Fig. 1. Intervals $(t_1, t_2]$ and $(t_3, t_4]$ are two different busy periods.

During a system busy period the server is always transmitting packets.

Definition 2: A *backlogged period for session i* is any period of time during which packets belonging to that session are continuously queued in the system.

Let $Q_i(t)$ represent the amount of session i traffic queued in the server at time t, that is,

$$Q_i(t) = A_i(0, t) - W_i(0, t).$$

A connection is backlogged at time t if $Q_i(t) > 0$.

Definition 3: A *session i busy period* is a maximal interval of time $(\tau_1, \tau_2]$ such that for any time $t \in (\tau_1, \tau_2]$, packets of connection i arrive with rate greater than or equal to ρ_i, or,

$$A_i(\tau_1, t) \geq \rho_i(t - \tau_1).$$

A session busy period is the maximal interval of time during which if the session was serviced with exactly the guaranteed rate, it would remain continuously backlogged (Figure 1). Multiple session-i busy periods may appear during a system busy period.

The session busy period is defined only in terms of the arrival function and the allocated rate. It is important to realize the basic distinction between a session backlogged period and a session busy period. The latter is defined with respect to a hypothetical system where a backlogged connection i is serviced at a constant rate ρ_i, while the former is based on the actual system where the instantaneous service rate varies according to the number of active connections and their service rates. Thus, a busy period may contain intervals during which the actual backlog of session i traffic in the system is zero; this occurs when the session receives an instantaneous service rate of more than ρ_i during the busy period.

In [11], we introduced a general model for traffic scheduling algorithms, called *Latency-Rate* ($\mathcal{LR}$) servers. Any server in this class is characterized by two parameters: *latency* Θ_i and *minimum allocated rate* ρ_i. Let us assume that the jth busy period of connection i starts at time τ. We denote by $W_{i,j}^{S}(\tau, t)$ the total service provided to the packets of the connection that arrived after time τ and until time t by server $\mathcal{S}$. Notice that the total service offered to connection i in this interval, $W_i^{S}(\tau, t)$, may actually be more than $W_{i,j}^{S}(\tau, t)$ since some packets from a previous busy period, that are still queued in the system, may be serviced as well.

Definition 4: A server $\mathcal{S}$ belongs in the class $\mathcal{LR}$ if and only if for all times t after time τ that the j-th busy period started and until the packets that arrived during this period are serviced,

$$W_{i,j}^{S}(\tau,t) \geq \max(0, \rho_i(t-\tau-\Theta_i^S)).$$

Θ_i^S is the minimum non-negative number that can satisfy the above inequality.

The right-hand side of the above equation defines an envelope to bound the minimum service offered to session i during a busy period. It is easy to observe that the latency Θ_i^S represents the worst-case delay seen by a session-i packet arriving into an empty queue. The maximum delay through a network of $\mathcal{LR}$-servers can be computed from the knowledge of the latencies of the individual servers and the traffic model. Thus, the theory of $\mathcal{LR}$-servers allows us to determine tight upper-bounds on end-to-end delays in a network of servers where the servers on a path may not all use the same scheduling algorithm.

The function $W_{i,j}^{S}(\tau,t)$ may be a step function in a packet-by-packet scheduler. As in the case of $W_i(\tau,t)$, we update $W_{i,j}^{S}(\tau,t)$ only when the last bit of a packet has been serviced. Only in the case of a fluid-server, packets can be arbitrarily small and thus $W_{i,j}^{S}(\tau,t)$ may be continuous.

To determine end-to-end delay bounds, we assume that traffic from session i at the source is leaky-bucket shaped [13]. That is,

$$A_i(\tau,t) \leq \sigma_i + \rho_i(t-\tau)$$

during any time interval $(\tau,t]$. Also, we assume that session i is allocated a minimum rate of ρ_i in the network. We state without proof the following key result from [11].

Theorem 1: The maximum delay D_i^K and the maximum backlog Q_i^K of session i after the Kth node in an arbitrary network of $\mathcal{LR}$-servers are bounded as

$$D_i^K \leq \frac{\sigma_i}{\rho_i} + \sum_{j=1}^{K} \Theta_i^{(S_j)};$$

$$Q_i^K \leq \sigma_i + \rho_i \sum_{j=1}^{K} \Theta_i^{(S_j)};$$

where $\Theta_i^{(S_j)}$ is the latency of the jth server on the path of the session.

In Table I we summarize the latencies of many well-known work-conserving schedulers, along with bounds on their fairness and implementation complexity. The fairness parameter in the table is the maximum difference in normalized service offered by the scheduler to two connections over any interval during which both connections are continuously backlogged. The implementation complexity is at least $O(\log_2 V)$ for all sorted-priority schedulers.

The packet-by-packet approximation of GPS (PGPS) has the lowest latency among all the packet-by-packet servers; thus, from Theorem 1, PGPS has the lowest bounds on end-to-end delay and buffer requirements. However, PGPS also has the highest implementation complexity. VirtualClock has the same latency as PGPS, but is not a fair algorithm [3], [1]. Notice, however, that none of the other algorithms suffers from such a high level of unfairness. In SCFQ as well as the round-robin schedulers, the latency is a function of the number of connections that share the output link. In a broadband network, the resulting end-to-end delay bounds may be prohibitively large.

The GPS scheduler provides ideal fairness by offering the same normalized service to all backlogged connections at every instant of time. Thus, if we represent the total amount of service received by each session by a function, then these functions can be seen to grow at the same rate for each backlogged session. Golestani [4] introduced such a function and called it *virtual time*. The virtual time of a backlogged session is a function whose rate of growth at each instant is exactly the rate of normalized service provided to it by the scheduler at that instant. Similarly, we can define a global virtual-time function that increases at the rate of the total service performed by the scheduler at each instant during a server-busy period. In a GPS scheduler, the virtual times of all backlogged connections are identical at every instant, and are equal to the global virtual time. This is achieved by setting the virtual time of a connection to the global virtual time when it becomes backlogged and then increasing the former at the rate of the instantaneous normalized service received by the connection during the backlogged period. This allows an idle connection to receive service immediately once it becomes backlogged, resulting in zero latency.

We introduce such a function to represent the state of each connection in a scheduler and call it *potential*. The potential of a connection is a non-decreasing function of time during a system-busy period. When connection i is backlogged, its potential increases exactly by the normalized service it received. That is, if $P_i(t)$ denotes the potential of connection i at time t, then, during any interval $(\tau,t]$ within a backlogged period for session i,

$$P_i(t) - P_i(\tau) = \frac{W_i(\tau,t)}{\rho_i}.$$

Note that the potentials of all connections can be initialized to zero at the beginning of a system-busy period, since all state information can be reset when the system becomes idle.

From the above definition of potentials, it is clear that a fair algorithm must attempt to increase the potentials of all backlogged connections at the same rate, the rate of increase of the system potential. Thus, the basic objective is to *equalize the potential* of each connection. Sorted-priority schedulers such as GPS, PGPS, SCFQ, and VirtualClock all attempt to achieve this objective. However, in our definition of potential, we did not specify how the potential of a connection is updated when it is idle, except that the potential is non-decreasing. Scheduling algorithms differ in the way they update the potentials of idle connections. Ideally, during every time interval that a connection i is not backlogged, its potential must increase by the normalized service that the connection could receive if it were backlogged. We will call this service the *missed service* of connection i. If the potential of an idle connection is increased by the service it missed, it is easy to see that, when the connection becomes

Server	Latency	Fairness	Complexity
GPS	0	0	-
PGPS	$\frac{L_i}{\rho_i} + \frac{L_{max}}{r}$	$\max(\max(C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}, C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i})$, where $C_i = \min((V-1)\frac{L_{max}}{\rho_i}, \max_{1 \le n \le V}(\frac{L_n}{\rho_n}))$.	$O(V)$
SCFQ	$\frac{L_i}{\rho_i} + \frac{L_{max}}{r}(V-1)$	$\frac{L_i}{\rho_i} + \frac{L_j}{\rho_j}$	$O(\log V)$
VirtualClock	$\frac{L_i}{\rho_i} + \frac{L_{max}}{r}$	∞	$O(\log V)$
Deficit Round Robin	$\frac{(3F-\phi_i)}{r}$	$\frac{3F}{r}$	$O(1)$
Weighted Round Robin	$\frac{(F-\phi_i+L_c)}{r}$	$\frac{F}{r}$	$O(1)$

TABLE I. Latency, fairness and implementation complexity of several work-conserving servers. L_i is the maximum packet size of session i and L_{max} the maximum packet size among all the sessions. C_i is the maximum normalized service that a session may receive in a PGPS server in excess of that in the GPS server. In weighted round-robin and deficit round-robin, F is the frame size and ϕ_i is the amount of traffic in the frame allocated to session i. L_c is the size of the fixed packet (cell) in weighted round-robin.

busy again, its potential will be identical to that of other backlogged connections in the system, allowing it to receive service immediately.

One way to update the potential of a connection when it becomes backlogged is to define a *system potential* that keeps track of the progress of the total work done by the scheduler. The system potential $P(t)$ is a non-decreasing function of time. When an idle session i becomes backlogged at time t, its potential $P_i(t)$ can be set to $P(t)$ to account for the service it missed. Schedulers use different functions to maintain the system potential, giving rise to widely different delay- and fairness-behaviors. In general, the system potential at time t can be defined as a non-decreasing function of the potentials of the individual connections before time t, and the real time t. Let $t-$ denote the instant just before time t. Then,

$$P(t) = \mathcal{F}(P_1(t-), P_2(t-), \ldots, P_V(t-), t). \tag{2.1}$$

For example, the GPS server initializes the potential of a newly backlogged connection to that of a connection currently backlogged in the system. That is,

$$P(t) = P_i(t), \quad \text{for any } i \in B(t);$$

where $B(t)$ is the set of backlogged connections at time t. The VirtualClock scheduler, on the other hand, initializes the potential of a connection to the real time when it becomes backlogged, so that

$$P(t_2) - P(t_1) = t_2 - t_1.$$

We will later show how the choice of the function $P(t)$ influences the delay and fairness behavior of the scheduler.

The utility of the system potential function $P(t)$ is in estimating the amount of service missed by a connection while it was idle. In an ideal server like GPS, the system potential is always equal to the potential of the connections that are currently backlogged and are thus receiving service. However, this approach requires that all connections can receive service at the same time. In a packet-by-packet scheduler we need to relax this constraint since only one connection can be serviced at a time. In the next section we will formulate the necessary conditions that the system potential function must satisfy in order for the server to have zero latency.

If the potential of a newly backlogged connection is estimated higher than the potential of the connections currently being serviced, the former may have to wait all the other connections before it can be serviced. The self-clocked fair queueing (SCFQ) algorithm is a self-contained approach to estimate the system potential function. The potential of the system is estimated by the finish potential of the packet that is currently being serviced. This approximation, may assign to the system potential function a value greater than the potential of some backlogged connections. Thus, a connection may not receive service immediately when a busy period starts. This behavior is different from that in GPS, where an idle connection starts to receive service immediately when it becomes backlogged.

III. Rate-Proportional Servers

We now use the concept of potential introduced in the last section to define a general class of schedulers, which we call *Rate-Proportional Servers* (RPS). We will first define these schedulers based on the fluid model and later extend the definition to the packet-by-packet version. We denote the set of backlogged connections at time t by $B(t)$.

Definition 5: A *rate proportional server* has the following properties:

1. Rate ρ_i is allocated to connection i and

$$\sum_{i=1}^{V} \rho_i \leq r$$

where r is the total service rate of the server.

2. A potential function $P_i(t)$ is associated with each connection i in the system, describing the state of the connection at time t. This function must satisfy the following properties:
   (a) When a connection is not backlogged, its potential remains constant.
   (b) If a connection becomes backlogged at time τ, then

$$P_i(\tau) = \max(P_i(\tau-), P(\tau-)) \tag{3.2}$$

   (c) For every time $t > \tau$, that the connection remains backlogged, the potential function of the connection is increased by the normalized serviced offered to that connection during the interval $(\tau, t]$. That is,

$$P_i(t) = P_i(\tau) + \frac{W_i(\tau, t)}{\rho_i} \tag{3.3}$$

3. The system potential function $P(t)$ describes the state of the system at time t. Two main conditions must be satisfied for the function $P(t)$:
   (a) For any interval $(t_1, t_2]$ during a system busy period,

$$P(t_2) - P(t_1) \geq (t_2 - t_1).$$

   (b) The system potential is always less than or equal to the potential of all backlogged connections at time t. That is,

$$P(t) \leq \min_{j \in B(t)} (P_j(t)). \tag{3.4}$$

4. Connections are serviced at each instant t according to their instantaneous potentials as per the following rules:
   (a) Among the set of backlogged connections, only the set of connections with the minimum potential at time t is serviced.
   (b) Each connection in this set is serviced with an instantaneous rate proportional to its reservation, so as to increase the potentials of the connections in this set at the same rate.

The above definition specifies the properties of the system potential function for constructing a zero-latency server, but does not define it precisely. In practice, the system potential function must be chosen such that the scheduler can be implemented efficiently. When we introduce the frame-based fair queueing algorithm in the next section, it will become clear how this definition can be used to design a practical scheduling algorithm.

GPS multiplexing is a rate-proportional server where the system potential is always equal to the potential of the backlogged connections. Since the service rate offered to the connections is proportional to their reservations at every instant, the normalized service they receive during an interval $(t_1, t_2]$ is never less than $(t_2 - t_1)$. Thus, the amount of service received by a connection i, backlogged during the interval (t_1, t_2), is given by

$$W_i(t_1, t_2) \geq \rho_i(t_2 - t_1),$$

and therefore,

$$\begin{aligned} P(t_2) - P(t_1) = P_i(t_2) - P_i(t_1) &= \frac{W_i(t_1, t_2)}{\rho_i} \\ &\geq t_2 - t_1. \end{aligned}$$

VirtualClock is a rate-proportional server as well. Consider a server where the system potential function is defined as

$$P(t) = t.$$

It is easy to verify that such a server satisfies all the properties of a rate-proportional server. Consider a packet-by-packet server that transmits packets in increasing order of their finishing potentials. Such a server is equivalent to the packet-by-packet VirtualClock server.

We now proceed to show that every rate-proportional server is a zero-latency server. This will establish that this class of servers provide the same upper-bounds on end-to-end delay as GPS. To prove this result, we first introduce the following definitions:

Definition 6: A session-i *active* period is a maximal interval of time during a system busy period, over which the potential of the session is not less than the potential of the system. Any other period will be considered as an *inactive* period for session i.

The concept of active period is useful in analyzing the behavior of a rate-proportional scheduler. When a connection is in an inactive period, it can not be backlogged and therefore can not be receiving any service. On the other hand, an active period need not be the same as a backlogged period for the connection. Since, in a rate-proportional server, the potential of a connection can be below the system potential only when the connection is idle, a transition from inactive to active state can occur only by the arrival of a packet of a connection that is currently idle, whose potential is below that of the system. A connection in an active period may not receive service throughout the active period since a rate-proportional server services only connections with the minimum potential at each instant. However, it always receives service at the beginning of the active period, since its potential is set equal to the system potential at that time.

Since $\mathcal{LR}$-servers are defined in terms of busy periods, it is necessary to establish the correspondence between busy periods and active periods in a rate-proportional server. We will now show that the beginning of a busy period is the beginning of an active period as well.

Lemma 1: If τ is the beginning of a session-i busy period in a rate-proportional server, then τ is also the beginning of an active period for session i.

A proof of this and subsequent lemmas and theorems can be found in [14]. When connection i becomes active, its potential is the minimum among all backlogged connections, enabling it to receive service immediately. However, if a subsequent connection j becomes active during the busy period of connection i, then the service of i may be temporarily suspended until the potentials of i and j become equal. In the following lemma, we derive a lower bound on the amount of service received by connection i during an active period.

Lemma 2: Let τ be the time at which a connection i becomes active in a rate-proportional server. Then, at any time $t > \tau$ that belongs in the same active period, the service offered to connection i is

$$W_i(\tau, t) \geq \rho_i(t - \tau).$$

This lemma is proved in [14]. Intuitively, this result asserts that the service of a backlogged connection is suspended only if it has received more service than its allocated rate earlier during the active period.

A session busy period may actually consist of multiple session active periods. In order to prove that a rate proportional server is an $\mathcal{LR}$ server with zero latency, we need to prove that for every time t after the beginning of the j-th busy period at time τ,

$$W_{i,j}(\tau, t) \geq \rho_i(t - \tau).$$

The above lemmas lead us to one of our key results:

Theorem 2: A rate-proportional server belongs to the class $\mathcal{LR}$ and has zero latency.

The main argument for proving this theorem is that during inactive periods the connection is not backlogged and is thus receiving no service. By Lemma 2, the connection can receive less than its allocated bandwidth only during an inactive period. However, since no packets are waiting to be serviced in an inactive period, the connection busy period must have ended by then. The formal proof can be found in [14].

Thus, the definition of rate-proportional servers provides us a tool to design scheduling algorithms with zero latency. Since both GPS and VirtualClock can be considered as rate-proportional servers, by Theorem 2, they have the same worst-case delay behavior.

A. Packet-by-Packet Rate-Proportional Servers

A packet-by-packet rate proportional server can be defined in terms of the fluid-model as one that transmits packets in increasing order of their finishing potential. Let us assume that when a packet from connection i finishes service in the fluid server, the potential of connection i is TS_i. We can use this finishing potential to timestamp packets and schedule them in increasing order of their time-stamps. We call such a server a *packet-by-packet rate-proportional server* (PRPS).

In the following, we denote the maximum packet size of session i as L_i and the maximum packet size among all the sessions as L_{max}.

In order to analyze the performance of a packet-by-packet rate-proportional server we will bound the difference of service offered between the packet-by-packet server and the fluid-server when the same pattern of arrivals is applied to both the servers. Let us assume that the service offered to session i during the interval $(\tau, t]$ by the fluid server is $W_i^F(\tau, t)$ and by the packet-by-packet server is $W_i^P(\tau, t)$. Let us assume that the kth packet leaves the system under the PRPS service discipline at time t_k^P. The same packet leaves the RPS server at time t_k^F. Using a similar approach as the one used for GPS servers [1], we can prove the following lemma:

Lemma 3: For all packets in a packet-by-packet rate-proportional server,

$$t_k^P \leq t_k^F + \frac{L_{max}}{r}.$$

If we include the partial service received by packets in transmission, the maximum lag in service for a session i in the packet-by-packet server occurs at the instant when a packet starts service. Let us denote with $\hat{W}_i(t)$ the service offered to connection i at time t if this partial service is included. At the instant when the kth packet starts service in PRPS,

$$\begin{aligned} \hat{W}_i^F(0, t_k) &\leq \hat{W}_i^F(0, t_k - \frac{L_{max}}{r}) + L_{max} \\ &\leq \hat{W}_i^P(0, t_k) + L_{max}. \end{aligned}$$

Thus, we can state the following corollary:

Corollary 1: At any time t,

$$\hat{W}_i^F(0, t) - \hat{W}_i^P(0, t) \leq L_{max}.$$

In order to be complete we also have to bound the amount by which the service of a session in the packet-by-packet server can be *ahead* of that in the fluid-server. Packets are serviced in PRPS in increasing order of their finishing potentials. If packets from multiple connections have the same finishing potential, then one of them will be selected for transmission first by the packet-by-packet server, causing the session to receive more service temporarily than in the fluid server. In order to bound this additional service, we need to determine the service that the connection receives in the fluid-server. The latter, in turn, requires knowledge of the potentials the other connections sharing the same outgoing link. We will use the following lemma to derive such an upper bound.

Lemma 4: Let $(0, t]$ be a server-busy period in the fluid server. Let i be a session backlogged in the fluid server at time t such that i received more service in the packet-by-packet server in the interval $(0, t]$. Then there is another session j, with $P_j(t) \leq P_i(t)$, that received more service in the fluid server than in the packet-by-packet server during the interval $(0, t]$.

A proof of this lemma can be found in [14]. We will now use the above lemma and a method similar to the one presented in [15] for the PGPS server to find an upper bound for the amount of service a session may receive in PRPS as compared to that in the fluid server.

Lemma 5: At any time t,

$$\hat{W}_i^P(0, t) - \hat{W}_i^F(0, t) \leq \min((V - 1)L_{max}, \rho_i \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}))$$

Lemma 5 establishes two distinct upper bounds for the excess service received by a session in the packet-by-packet server. A formal proof of the lemma is given in [14].

B. Delay Analysis

Based on the bounds on the discrepancy between the service offered by the packet-by-packet server and that by the fluid server at any time during a session busy period, we can bound the performance of the PRPS system using the worst-case performance of the fluid-system. Thus, we will now prove that a packet-by-packet rate proportional server is an $\mathcal{LR}$-server and estimate its latency.

Let us assume that a packet from connection i leaves the PRPS system at time t^P and the fluid-system at time t^F. Then, by Lemma 3,

$$t_k^P \leq t_k^F + \frac{L_{max}}{r}.$$

For the analysis of a network of $\mathcal{LR}$ servers it is required that the service is bounded for any time after the beginning of a busy period. In addition, we can only consider that a packet left the packet-by-packet server if all of its bits have left the server. These requirements are necessary in order to provide accurate bounds for the traffic burstiness inside the network. Therefore, just before time $t^P + \frac{L_{max}}{r}$ the whole packet has not yet departed the packet-by-packet server. Let L_i be the maximum packet size of connection i. The service offered to connection i in the packet-by-packet server will be equal to the service offered to the same connection in the fluid server until time t^P, minus this last packet. Therefore, the service received by session i during the jth busy period in the packet-by-packet server is given by

$$\begin{aligned} W_{i,j}^P(\tau,t) &\geq W_{i,j}^F(\tau, t - \frac{L_{max}}{r}) - L_i \\ &\geq \max(0, \rho_i(t - \tau - \frac{L_{max}}{r}) - L_i) \\ &\geq \max(0, \rho_i(t - \tau - \frac{L_{max}}{r} - \frac{L_i}{\rho_i})) \quad (3.5) \end{aligned}$$

Hence, we can state the following corollary:

Corollary 2: A packet-by-packet rate proportional server is an $\mathcal{LR}$ server and its latency is

$$\frac{L_{max}}{r} + \frac{L_i}{\rho_i}.$$

Note that this latency is the same as that of PGPS. Thus, any packet-by-packet rate-proportional server has the same upper bound on end-to-end delay and buffer requirements as those of PGPS when the traffic in the session under observation is shaped by a leaky bucket.

Although all servers in the RPS class have zero latency, their fairness characteristics can be widely different. Therefore, we take up the topic of fairness in the next section and derive bounds on the fairness of rate-proportional servers.

C. Fairness of Rate-Proportional Servers

In our definition of rate-proportional servers, we specified only the conditions the system potential function must satisfy to obtain zero latency, but did not explain how the choice of the actual function affects the behavior of the scheduler. The choice of the system-potential function has a significant influence on the fairness of service provided to the sessions. In the last section, we showed that a backlogged session in a rate-proportional server receives an average service over an active period at least equal to its reservation. However, significant discrepancies may exist in the service provided to a session over the short term among scheduling algorithms belonging to the RPS class. The scheduler may penalize sessions for service received in excess of their reservations at an earlier time. Thus, a backlogged session may be starved until others receive an equivalent amount of normalized service, leading to short-term unfairness.

Since, in a fluid-model rate-proportional server, backlogged connections are serviced at the same normalized rate in steady state, unfairness in service can occur only when an idle connection becomes backlogged. If the estimated system potential at that time is far below that of the backlogged connections, the new connection may receive exclusive service for a long time until its potential rises to that of other backlogged connections.

Let us assume that at time τ two connections i, j become greedy, requesting an infinite amount of bandwidth. Thus, the two connections will be continuously backlogged in the system after time τ. A scheduler is considered as fair if the difference in normalized service offered to the two connections i, j during any interval of time $(t_1, t_2]$ after time τ is bounded. That is,

$$\left| \frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j} \right| \leq \mathcal{FR}, \quad (3.6)$$

where $\mathcal{FR} < \infty$ is a measure of the fairness of the algorithm. Note that the requirement of an infinite supply of packets from sessions i and j arises because we require the two sessions to be backlogged *at every instant* after τ in each of the schedulers we study. Since, for the same arrival pattern, the backlogged periods of individual sessions can vary across schedulers, a comparison of fairness of different scheduling algorithms can yield misleading results without this condition. When the connections have an infinite supply of packets after time τ, they will be continuously backlogged in the interval $(t_1, t_2]$ irrespective of the scheduling algorithm used. Thus, to compare the fairness of different schedulers, we can analyze each of the schedulers with the same arrival pattern and determine the difference in normalized service offered to the two connections in a specified interval of time.

Let us denote with ΔP, the maximum difference between the system potential and the potential of the connections being serviced in a rate-proportional server. The following theorem formalizes our basic result on the fairness properties of rate-proportional servers.

Theorem 3: If the system potential function in a rate-proportional server never lags behind more than a finite amount ΔP from the potential of the connections that are serviced in the system, the difference in normalized service offered to any two connections during any interval of time that they are continuously backlogged is also bounded by

ΔP. That is, if $\Delta P < \infty$, then for all $i, j \in B(t_1, t_2)$ during the interval $(t_1, t_2]$,

$$\left| \frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j} \right| \leq \Delta P.$$

A proof of this theorem is given in [14]. The theorem applies to the fluid system. A real system can only use a packet-by-packet rate-proportional server. We will now expand the above theorem to prove that a similar relationship holds for the packet-by-packet version of the algorithm. Let us define C_i as

$$C_i = \min((V-1)\frac{L_{max}}{\rho_i}, \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}))$$

That is, C_i is the maximum normalized service that a connection can receive over any interval in the packet-by-packet server in excess of that offered by the fluid-server.

Theorem 4: In a packet-by-packet rate-proportional server, for every time interval $(t_1, t_2]$ after time τ that both connections became greedy,

$$|\frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i}| \leq \max(\Delta P + C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}, \Delta P + C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}) \quad (3.7)$$

A proof of this theorem can be found in [14]. Since PGPS is a packet-by-packet rate proportional server with $\Delta P = 0$, we obtain the following result on the fairness of a PGPS scheduler by setting $\Delta P = 0$ in Eq. (3.7).

Corollary 3: For a PGPS scheduler,

$$|\frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i}| \leq \max(C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}, C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}) \quad (3.8)$$

It can be shown that the above bound is tight. For example, consider the case where connections i, j are already backlogged in the system, at time τ. Then, connection j may have received an additional amount of service equal to C_i in the PGPS server and connection i may have received less service equal to L_{max} in the PGPS server compared to the GPS server. The finishing potential of the last packet serviced from connection i before a packet is serviced from connection j may be $F_i \leq P_j(\tau)+C_i+\frac{L_i}{\rho_i}$. The total normalized service that connection i may receive while connection j is waiting is bounded by $C_j + \frac{L_j}{\rho_j} + \frac{L_{max}}{\rho_i}$.

IV. Frame-based Fair Queueing

The basic difficulty in the design of a scheduling algorithm in the RPS class is in maintaining the system potential function. We use a potential function that can be calculated in a simple way, and use a framing mechanism to bound its difference ΔP from the potential of a backlogged connection.

The fluid version of the frame-based fair queueing (FFQ) algorithm is defined as follows: FFQ is a rate-proportional server and therefore follows all the conditions in Definition 5. That is, at each instant, it services only the set of backlogged connections with the minimum potential and connections in this set are serviced at rates proportional to their reservations. We assume that a rate ρ_i is allocated to connection i. Let $r_i = \rho_i / r$ denote the fraction of the link rate allocated to connection i. We split the time in frames and assume that F bits may transmitted during a frame period. Furthermore, let us define T as the frame period. Then,

$$T = \frac{F}{r}.$$

We define ϕ_i as

$$\phi_i = r_i \times F = \rho_i \times T.$$

Thus, ϕ_i denotes the amount of session i traffic that can be serviced during one frame. If a connection is backlogged its potential is increasing by the normalized service offered to it. Thus, when ϕ_i bits are serviced from connection i, its potential will increase by

$$\frac{\phi_i}{\rho_i} = T.$$

We impose one more restriction on the value of ϕ_i, that if L_i is the maximum packet size for connection i, then

$$L_i \leq \phi_i. \quad (4.9)$$

That is, the largest packet of a connection can be transmitted during a frame period. We denote the current frame in progress at time t by $f(t)$. The function $f(t)$ is a step function. When the system is empty, $f(t)$ is reset to 0 and the potentials of all connections are also reset to 0. When the potentials of all backlogged connections become equal to T, the value of $f(t)$ is increased by T. Similarly, when the potentials of all connections reach the value $k \cdot T$ for any integer k, the function $f(t)$ is stepped up to $k \cdot T$.

In a fluid server, all backlogged connections reach the value $k \cdot T$ at the same time. However, in a packet-by-packet server this is not the case. In order to allow frame updates to occur only when a packet finishes or starts service in the packet-by-packet server, we relax the above update rule for $f(t)$ in the packet-by-packet version of the algorithm as follows: $f(t)$ can be updated to the value $k \cdot T$ at any time after all backlogged connections reach the potential $k \cdot T$ and before their potentials become higher than $(k+1) \cdot T$. Thus, we update the frame at time t when both of the following conditions hold:

1. The potentials of all backlogged connections belong in the next frame. That is,

$$P_i(t) \geq f(t) + T, \quad \forall i \in B(t), \quad (4.10)$$

   where $B(t)$ is the set of connections currently backlogged.
2. $P_i(t) < f(t) + 2T, \quad i = 1, 2, \ldots, V.$

Note that the above conditions may hold at different instants of time. Updating the system potential function at any time during these intervals will result in a valid algorithm. Let us assume that we decide to update the frame at time τ. Then, at time τ we set

$$f(\tau) = f(\tau-) + T, \tag{4.11}$$

where $\tau-$ is the instant just before the update occurred.

The system potential is estimated in terms of the function $f(t)$ which keeps track of the progress of the total work performed by the server. When $f(t)$ is updated, say at time τ, the system potential is set to

$$P(\tau) = \max(P(\tau-), f(\tau)). \tag{4.12}$$

At any other time t, the system potential is computed as

$$P(t) = P(\tau) + (t - \tau), \tag{4.13}$$

where τ is the last instant of time when an update occurred.

If $P(t)$ is used to initialize the potential of a connection when it becomes backlogged, it is easy to see that the potential of every backlogged connection cannot be less than $P(t)$ at any time t. The potential of a connection i can drift away from $P(t)$ within a frame, but the discrepancy is corrected when the next frame-update occurs. Thus, the frame update mechanism is the means by which FFQ bounds the difference between the system potential and the potentials of individual connections. The maximum interval between successive updates acts as bound for the difference in potentials ΔP. A tight bound for ΔP will be derived later.

A. Performance Bounds for Frame-based Fair Queueing

In this section we show that FFQ belongs to the class of rate-proportional servers. Note that, in our description of FFQ, we did not define exactly the time when the frame is updated, but instead defined an interval during which the update must occur. We will show that this is sufficient to prove that frame-based fair-queueing is a rate-proportional server. This flexibility in updating the frame will allow us to provide a simple implementation for the packet-by-packet version of the algorithm.

We now state sequence of two lemmas to classify frame-based fair queueing as a rate-proportional server. Formal proofs can be found in [14].

Lemma 6: If the system potential function is updated as described by the frame-based fair queueing algorithm, then for any interval $(t_1, t_2]$ during a system busy period,

$$P(t_2) - P(t_1) \geq (t_2 - t_1).$$

Lemma 7: If the system potential function is updated as described by the frame-based fair queueing algorithm, then

$$P(t) \leq P_i(t), \quad \forall i \in B(t).$$

Based on the above two lemmas, we can now state the following theorem:

Theorem 5: Frame-based fair queueing is a rate-proportional server.

Since we proved that every rate-proportional server is an $\mathcal{LR}$-server with zero latency, frame-based fair queueing is also an $\mathcal{LR}$-server with zero latency. Therefore, it provides the same upper bound on end-to-end delay as GPS in a network of servers.

B. Fairness of Frame-based Fair Queueing

Since frame-based fair queueing is a rate-proportional server, in order to analyze its fairness it is sufficient to prove that the difference between the system potential and the potential of any backlogged connection is always bounded. In [14], we prove the following Lemma:

Lemma 8: For every connection i that is backlogged at time t,

$$P_i(t) - P(t) \leq 2T - \frac{\phi_i}{r}.$$

Note that the fastest way for the potential of a connection to reach the value $P_i(t)$ from the time that the frame was last updated is through its normalized service. However, by the time the next frame update occurs, the system potential function would have increased by at least the time to service ϕ_i bits of connection i. This bounds the difference in potentials to $2T - \frac{\phi_i}{r}$.

The above bound applies to the fluid server. The packet-by-packet version of frame-based fair queueing, described in the next section, guarantees that a frame update will always occur before the finishing potential of any packet becomes greater than $f(t)+2T$. Thus, when we estimate the fairness of the packet-by-packet algorithm (PFFQ), the maximum difference between the starting potential of a connection and the finishing potential of any other connection is still bounded by $2T$. Using the general proof of Theorem 4 for rate-proportional servers and the fact that $T = F/r$, it is easy to show that

Corollary 4: For any two connections i, j that are continuously backlogged in the interval $(t_1, t_2]$ in the PFFQ server,

$$\left| \frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j} \right| \leq \frac{2F}{r} + \max\left(\frac{L_i}{\rho_i}, \frac{L_j}{\rho_j}\right).$$

The fairness of the algorithm depends on the selection of the frame size. The latter, in turn, depends on the maximum packet size of each connection and its minimum bandwidth allocation. Thus, the algorithm is especially suited to application in ATM networks where the packets are transmitted in terms of small cells and the frame size can be kept small. Notice, however, that the frame size does not affect the latency of the server as is the case in frame-based schedulers such as weighted-round-robin and deficit-round-robin. In addition, some short-term unfairness is unavoidable in any packet-level scheduler. We have already seen in Theorem 4 that the difference in normalized service received by two connections can be proportional to the number of backlogged connections even in a PGPS server. Most applications can tolerate a small amount of short-term unfairness as long as the unfairness is bounded.

V. Implementation of Frame-Based Fair Queueing

In the last section we described frame-based fair queueing and showed that it is a rate-proportional server. A straightforward implementation of the algorithm will be similar to that of any other rate-proportional server. That is, the fluid server is simulated in parallel and packets are transmitted in increasing order of their finishing potentials. This approach, however, is very expensive since up to V events may be triggered in the simulator of the fluid model during the service time of one packet. This results in an algorithm with $O(V)$ time complexity that is not acceptable in a high-speed network.

We will now describe how we can extract the information needed by the scheduler from the packet-by-packet system itself, without requiring the parallel simulation of the fluid server. The resulting algorithm has $O(1)$ implementation complexity.

We first prove the following lemma that enables us to find a suitable time to update the frame in FFQ by using only information extracted from the packet-by-packet system. Let us first define the starting potential s_i^j of a packet j of connection i as the potential of the connection when packet j starts being serviced in the corresponding fluid server. Let $\hat{B}(t)$ denote the set of backlogged sessions at time t in the packet-by-packet server.

Lemma 9: Assume that at time t, for each backlogged session in the packet-by-packet system, the starting potential of its first packet belongs in the next frame. That is,

$$s_i^j \geq f(t) + T, \quad \forall i \in \hat{B}(t).$$

Then, the potential of each backlogged session in the fluid server is also greater than $f(t) + T$.

The intuition behind the lemma is that we can determine a valid update time for the frame by using information extracted by the packet-by-packet server. The scheduler can keep track of all the connections that are backlogged and have packets with starting potential in the next frame. When the starting potentials of the packets at the head of the queue of all backlogged sessions have crossed the frame, we know that the potentials of the connections in the fluid-system have also crossed the frame. Therefore, the crossing time of the last connection is a valid time to update the frame and the system potential function.

A. Implementation for General Packet-Switched Networks

Without loss of generality we can assume that the service rate of the server is 1. Thus the time to transmit F bits is also equal to F. A fraction r_i of the output link bandwidth is allocated to connection i and therefore $\phi_i = F \times r_i$ bits can be sent from connection i during a frame. An additional requirement is that the maximum packet size must be less than ϕ_i, so that a single packet can be transmitted within one frame. On the arrival of a packet, it is stamped with a timestamp equal to the potential of the connection when the packet finishes service. Packets are then serviced in increasing order of their timestamps. When a packet finishes service, a frame update mechanism is used to calculate the new system potential.

```
Calculate current value of system potential.
Let t be the current time and t_s the time when the
packet currently in transmission started its service
temp ← P + (t − t_s)/F

Calculate the starting potential of the new packet
SP(i,k) ← max(TS(i,k − 1), temp)

Calculate timestamp of packet
TS(i,k) ← SP(i,k) + length(i,k)/ρ_i

Check if packet crosses a frame boundary
n1 ← int(SP(i,k)); n2 ← int(TS(i,k))
if (n1 < n2) then
    B[n1] ← B[n1] + 1 (increment counter);
    mark packet
endif
```

Fig. 2. Algorithm executed on the arrival of a packet.

On the arrival of a packet the algorithm in Figure 2 is executed. An explanation of the algorithm follows: The variable P keeps track of the system potential. P is a floating-point number with two parts — the integer part representing the current frame number and the fractional part representing the elapsed real time since the last frame update. On arrival of a packet, the current potential is estimated in the variable *temp*, scaled to the frame size. The starting potential of the packet is then computed as the maximum of the finishing potential of its previous packet and the system potential. The packet is then stamped with its finishing potential, computed from knowledge of its length and the reserved rate. If the starting and finishing potentials of the packet belong to different frames, the current packet is one that crosses over to the next frame. Therefore the packet is marked to indicate that it is the first packet of the session to cross over to the next frame. In addition, a counter is incremented to keep track of the number of connections that have crossed over into the new frame. The algorithm maintains one counter per frame to keep track of the number of sessions whose packets cross into the frame. Later, when the marked packet is scheduled for transmission, the corresponding counter is decremented; when the counter reaches zero, the potentials of all the backlogged connections have crossed over to the next frame, and a frame update can be performed.

The array of counters B is used to count the number of connections that have packets with a starting potential in each frame. Although an infinite number of frames may need to be serviced, in practice the number of distinct frames in which the potentials of queued packets can fall into is limited by the buffer size allocated to the connections. Thus, if b_i denotes the buffer space allocated to connection i, the size of the array B can be limited to

$$M = \max_{1 \leq i \leq V} \lceil \frac{b_i}{\phi_i} \rceil.$$

```
    Increase system potential by the transmission time
    of the packet just completed
    P ← P + length(j)/F
    Find the next packet for transmission.
    TS_min ← min_{i∈B}(TS(i))
    Determine frame of the packet with
    the minimum timestamp.
    F_min ← int(TS_min)
    Perform frame update operation if required
    if (current packet is marked) then
        B[Frame] ← B[Frame] − 1
    end if
    if (B[Frame] = 0 and F_min > Frame) then
        Frame ← Frame + 1
        P ← max(Frame, P)
    end if
    Store starting time of transmission
    of next packet in t_s
    t_s ← current time
    Transmit packet from head of queue.
```

Fig. 3. Algorithm executed on the departure of a packet.

If M is rounded up to the nearest power of 2, then the array can be addressed with the $\lceil \log_2 M \rceil$ least significant bits of the current frame number. Obviously, instead of the array, a linked-list implementation of the counters can be used as well. When a packet finishes transmission, the algorithm described in Figure 3 is executed to update the state of the system. We assume that the current packet being transmitted is the kth packet of connection i.

The system potential is first increased by the transmission time of the current packet. The packet with the minimum timestamp is then selected. Packets can be organized in a priority list structure, like a heap, and the time complexity of the operation is $O(\log V)$. The variable *Frame* keeps track of the index of the frame currently in progress. If the packet is marked (first packet from the connection to cross the frame), the counter corresponding to the current frame is decremented. If the counter becomes zero, the session being serviced is the last to cross the current frame. However, it is possible that during the transmission of that packet, a packet arrived from another connection with a potential in the current frame. Thus, only if all connections have crossed the frame and no packet has a finish potential in the current frame can we update the frame number. The system potential is then updated, based on the new frame number.

Although the algorithm described above can be applied to an ATM network as stated above, the implementation may be simplified even further by replacing the necessary multiplication by a shift operation. We require that F is a power of two. The unit of time is the time required to transmit one cell at the link rate. The average cell inter-arrival time $1/\rho_i$ is selected as an integer. Then, we define the system potential $P(t)$ as the current frame number multiplied by the maximum frame size plus the units of time that passed since the last update of the frame. Forcing the frame size to be a power of two enables the multiplication step to be implemented as a shift operation over $b = \log_2 F$ bits. If this approach is used, the priority list implementation can be simplified as well by using the techniques described in [16]. The interested reader is referred to [14] for a correctness proof of the algorithm, and more details on its implementation.

VI. Simulation Results

The analysis in the previous sections showed that Frame-based Fair Queueing provides the same end-to-end delay bound as PGPS and bounded unfairness. In this section we study the behavior of the FFQ algorithm by simulation in some realistic network configurations. For comparison, we provide simulation results for PGPS and Self-Clocked Fair Queueing (SCFQ). The performance metrics we use for this comparison are the average and maximum delays. We present results from simulating the algorithms as applied to a single output port of an ATM switch.

In our model, we considered eight sessions sharing the same outgoing link. The reservations of the sessions are shown in Table 1. An ON-OFF traffic model was used to generate traffic within each session. Both the ON and OFF periods of the traffic model were drawn from a Poisson distribution; the mean duration of the ON period of session i was set to $100 \times \rho_i$ cell times, and the mean OFF time to $100 \times (1-\rho_i)$ cell times, where ρ_i is the reservation of session i. The traffic was then shaped through a leaky bucket.

Since our interest is in evaluating the delay in the scheduler rather than the effect of input burstiness, we selected a σ of 2 for each connection. We also assumed that one session (session 1) is misbehaving, attempting to transmit more than its reservation. We assumed an infinite number of buffers, causing session 1 to remain backlogged throughout the simulation. For the simulation of Frame-based Fair Queueing we set the frame size as 1000 cell times. With this model, we measured the delays and bandwidth allocations received by all the sessions. A summary of our results is presented in Table II. Delays are shown in the table in terms of cell-transmission times.

Both the average and maximum delays of session 0, which has reserved 50% of the link bandwidth, are substantially lower in the FFQ server as compared to the SCFQ server. FFQ provides a maximum end-to-end delay of 2 for this session. By using Theorem 1, the maximum end-to-end delay for virtual channel 0 can be computed as $\frac{2}{0.5} + 1 = 5$. Note, that the maximum delay observed under SCFQ is even higher than this upper bound. A large value of the maximum delay may lead to increased burstiness and buffer requirements within the network, if the session is going through multiple hops. This is consistent with [11], [12] where it was shown that the maximum delay for SCFQ increases with the number of connections sharing the outgoing link.

The delays of sessions 2–7 are also lower in the FFQ server. The higher delays experienced in the SCFQ server are a result of its different fairness properties. The SCFQ

Session	Reserved Bandwidth	Arrival Rate	FFQ		PGPS		SCFQ	
			Avg. Delay	Max Delay	Avg. Delay	Max Delay	Avg. Delay	Max Delay
0	0.500000	0.498	1.995	2.00	1.995	3.00	4.2353	8.00
1	0.062500	0.100	N/A	N/A	N/A	N/A	N/A	N/A
2	0.062500	0.062	6.424	21.0	7.259	25.0	15.101	29.0
3	0.062500	0.061	8.391	21.0	9.166	23.0	15.828	25.0
4	0.078125	0.076	3.685	12.0	4.152	13.0	10.284	14.0
5	0.078125	0.076	5.264	13.0	5.759	17.0	11.060	18.0
6	0.078125	0.076	6.283	14.0	6.795	16.0	11.618	18.0
7	0.078125	0.076	7.023	14.0	7.550	16.0	11.943	20.0

TABLE II. Comparison of *average* delays from a simulation of FFQ, PGPS and SCFQ algorithms. The eight sessions shown share the same output link. Delays are measured in terms of cell-transmission times. Session 1 is misbehaving while others are transmitting within their reservations.

algorithm provided to session 1 more service than its reservation, sacrificing the delays of the other connections. In contrast, the FFQ server may slightly penalize the misbehaving connection for bandwidth it received while the other connections were absent. Remember that the system potential may lag behind the connection potentials by an amount determined by the frame size. However, it is important to note that FFQ behaves very similar to PGPS for all connections. At the same time, the average delays experienced by the connections under SCFQ differ significantly from those in PGPS. These results, clearly illustrate that FFQ can be considered as a very good approximation of PGPS.

VII. Conclusions

In this paper, we introduced and analyzed frame-based fair queueing, a novel traffic scheduling algorithm with application in both ATM and general packet networks. The algorithm provides the same end-to-end delay guarantees as a PGPS server if the input traffic is leaky-bucket shaped. We also analyzed the fairness properties of the algorithm, and showed that the difference in normalized service offered to any two connections that are continuously backlogged is always bounded. The main advantage of the algorithm compared to PGPS is that it does not require simulation of the fluid server, enabling it to be implemented in a simple and efficient manner. All the information needed for the algorithm can be extracted from the packet-by-packet server itself.

Apart from the scheduling algorithm itself, a major contribution of this paper is in developing a framework for designing schedulers with low latency and bounded fairness. This framework of rate-proportional servers (RPS) provides valuable insight into the behavior of scheduling algorithms. It is hoped that this framework will lead to the development of other algorithms in the future.

Further work will include the analysis of frame-based fair queueing under probabilistic input traffic models, such as the *exponentially-bounded-burstiness model* [17]. We also plan to implement the algorithm in hardware using our FPGA-based ATM Simulation Testbed [18]. A network of switches will be simulated in order to evaluate the performance of the algorithm in conjunction with variable-bit-rate traffic.

Acknowledgments

The authors would like to acknowledge the anonymous reviewers for their helpful comments.

References

[1] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control - the single node case," in *Proc. of INFOCOM '92*, vol. 2, pp. 915–924, May 1992.

[2] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Internetworking: Research and Experience*, vol. 1, no. 1, pp. 3–26, 1990.

[3] L. Zhang, "VirtualClock: a new traffic control algorithm for packet switching networks," *ACM Transactions on Computer Systems*, vol. 9, pp. 101–124, May 1991.

[4] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. of INFOCOM '94*, pp. 636–646, IEEE, April 1994.

[5] D. Ferrari and D. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 368–379, April 1990.

[6] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1265–79, October 1991.

[7] M. Shreedhar and G. Varghese, "Efficient Fair Queueing using Deficit Round Robin," in *Proc. SIGCOMM'95*, pp.231–242, September 1995.

[8] H. Zhang and S. Keshav, "Comparison of rate based service disciplines," in *Proc. of ACM SIGCOMM '91*, pp. 113–122, 1991.

[9] C. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks," in *IEEE Global Telecommunications Conference*, pp. 300.3.1–300.3.9, December 1990.

[10] S. Golestani, "A framing strategy for congestion management," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1064–1077, September 1991.

[11] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," in *Proc. of IEEE INFOCOM '96*, March 1996, also (http://www.cse.ucsc.edu/research/hsnlab/publications/).

[12] S. Golestani, "Network delay analysis of a class of fair queueing algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1057–70, August 1995.

[13] J. Turner, "New directions in communications (or which way to the information age?)," *IEEE Communications Magazine*, vol. 24, pp. 8–15, October 1986.

[14] D. Stiliadis and A. Varma, "Frame-based fair queueing:a new traffic scheduling algorithm for packet-switched networks," Tech. Rep. UCSC-CRL-95-39, U.C. Santa Cruz, http://www.cse.ucsc.edu/research/hsnlab/publications/.

[15] J. Rexford, A. Greenberg, and F. Bonomi, "A fair leaky-bucket shaper for ATM networks." AT&T unpublished report.

[16] J. L. Rexford, A. Greenberg, and F. Bonomi, "Hardware efficient fair queueing architectures for high-speed networks," in *Proc. of IEEE INFOCOM 96*, March 1996.

[17] O. Yaron and M. Sidi, "Performance and stability of communication networks via robust exponential bounds," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 372–385, June 1993.

[18] D. Stiliadis and A. Varma, "FAST: an FPGA-based simulation testbed for ATM Networks," in *Proc. ICC '96*, June 1996.