Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks

Dimitrios Stiliadis

Anujan Varma

UCSC-CRL-95-39
July 18, 1995

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

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. Implementations of the algorithm are described for both general packet switches supporting variable packet sizes, and ATM switches supporting fixed-size cells. 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.

Keywords: Packet scheduling, ATM switch scheduling, fair queueing, delay bounds, fairness.

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

1 Introduction

Many future applications of computer networks such as distance education, remote collaboration, and teleconferencing will rely on the ability of the network to provide Quality-of-Service (QoS) guarantees. These guarantees are usually in the form of bounds on end-to-end delay, bandwidth, delay jitter (variation in delay), packet loss rate, or a combination of these parameters. Broadband packet networks based on ATM (Asynchronous Transfer Mode) are currently enabling the integration of traffic with a wide range of characteristics within a single communication network. QoS guarantees can also be provided in conventional packet networks by the use of proper packet scheduling algorithms in the packet switches.

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. Because of the small size of the ATM cell, the scheduling algorithm must usually be implemented in hardware in an ATM switch. In a packet network with larger packet-sizes, such as the current Internet, the algorithm can be implemented in software.

Several service disciplines are known in the literature for bandwidth allocation and transmission scheduling in output-buffered switches. In general, schedulers can be characterized as *work-conserving* or *non-work-conserving*. A scheduler is work-conserving if the server is never idle when a packet is buffered in the system. A non-work-conserving server may remain idle even if there are available packets to transmit. A server may, for example, postpone the transmission of a packet when it expects a higher-priority packet to arrive soon, even though it is currently idle. When the transmission time of a packet is short, as is typically the case in an ATM network, however, such a policy is seldom justified. Non-work-conserving algorithms are also used to control delay jitter by delaying packets that arrive early [1]. Work-conserving servers always have lower average delays than non-work-conserving servers. Examples of work-conserving schedulers include Generalized Processor Sharing (GPS) [2], Weighted Fair Queueing [3], VirtualClock [4], Delay-Earliest-Due-Date (Delay-EDD) [5], Weighted Round Robin [6], and Deficit Round Robin [7]. On the other hand, Hierarchical-Round-Robin (HRR) [8], Stop-and-Go queueing [9], and Jitter-Earliest-Due-Date [1] are non-work-conserving schedulers.

Another classification of schedulers is based on their internal structure [10]. According to this classification there are two main architectures: *sorted-priority* and *frame-based*. 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, Weighted Fair Queueing, and Delay-EDD follow this architecture. Two factors determine the implementation complexity of all sorted-priority algorithms: First, the complexity of updating the priority list and selecting the packet with the highest priority is at least $O(\log V)$ where V is the number of connections sharing the outgoing link. The second is the complexity of calculating the timestamp associated with each packet; this factor depends heavily on the algorithm. For example, maintaining the virtual time in Weighted Fair Queueing requires

the processing of a maximum of V events during the transmission of a single packet, whereas timestamps in VirtualClock can be calculated in $O(1)$ time.

In a frame-based scheduler, time is split into frames of fixed or variable length. Reservations of sessions are made in terms of the maximum amount of traffic the session is allowed to transmit during a frame period. Hierarchical Round Robin and Stop-and-Go Queueing are frame-based schedulers that use a constant frame size. As a result, the server may remain idle if sessions transmit less traffic than their reservations over the duration of a frame. In contrast, Weighted Round Robin and Deficit Round Robin schedulers allow the frame size to vary within a maximum. Thus, if the traffic from a session is less than its reservation, a new frame can be started early. Therefore, both of these schedulers are work-conserving.

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. That is, the algorithm must be able to maintain the QoS guarantees for a session even in the presence of other misbehaving flows. 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: The algorithm must provide end-to-end delay guarantees for individual sessions. In particular, it is desirable that the end-to-end delay of a session depends only on its bandwidth reservation, and is independent of the behavior of other sessions.
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. Two algorithms with the same maximum delay guarantee may have significantly different fairness characteristics. 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. At SONET OC-3 speeds the transmission time of an cell is less than 3 μs. For higher speeds the available time is even less. This forces a hardware implementation. In packet networks with larger packet sizes and/or lower speeds, a software implementation 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.

Based only on the end-to-end delay bounds and fairness properties, Generalized-Processor-Sharing (GPS) is an ideal scheduling discipline [2]. GPS multiplexing is defined with respect to a fluid-model, where packets are considered to be infinitely divisible. The share of bandwidth reserved by session i is represented by a real number ϕ_i. Let $B(\tau, t)$ be the set of connections that are backlogged in the interval $(\tau, t]$. If r is the rate of the server, the service $W_i(\tau, t)$ offered to a connection i that belongs in $B(\tau, t)$ is proportional to ϕ_i. That is,

$$W_i(\tau, t) \geq \frac{\phi_i}{\sum_{j \in B(\tau,t)} \phi_j} r(t - \tau).$$

The minimum service that a connection can receive in any interval of time is

$$\frac{\phi_i}{\sum_{j=1}^{V} \phi_j} r(t-\tau),$$

where V is the maximum number of connections that can be backlogged in the server at the same time. Thus, GPS serves each backlogged session with a minimum rate equal to its reserved rate at each instant; in addition, the excess bandwidth available from sessions not using their reservations is distributed among all the backlogged connections at each instant in proportion to their individual reservations. This results in perfect isolation, ideal fairness, and low end-to-end session delays.

A packet-by-packet version of the algorithm, known as PGPS or Weighted Fair Queueing, was defined in terms of a virtual clock that is increased with rate equal to

$$\frac{1}{\sum_{i \in B(\tau,t)} \phi_i}.$$

A GPS system is simulated in parallel with the packet-by-packet system in order to identify the set of connections that are backlogged at each time. The virtual time $v(t)$ is a piecewise linear function of the real time t, and its slope changes depending on the number of busy sessions and their service rates. At the arrival of a new packet, the virtual time must be calculated first. Then, the time-stamp TS_i associated with the kth packet of virtual channel i is calculated as:

$$TS_i^k \leftarrow \max(TS_i^{k-1}, v(t)) + \frac{L}{\phi_i},$$

where L is the size of the kth packet.

A maximum of V events may be triggered in the GPS simulator during the transmission of one packet. Thus, the process overhead for completing a scheduling decision is $O(V)$. In order to reduce this complexity, an approximate implementation of GPS multiplexing was proposed in [11] and was later analyzed in [12] under the name *Self-Clocked Fair Queueing* (SCFQ). In this implementation, the timestamp of an arriving packet is computed based on the packet currently in service. Thus, if TS_{cur} denotes the timestamp of the packet in service, and if the new packet is the kth packet of session i, the timestamp of the new packet is calculated as

$$TS_i^k \leftarrow \max(TS_{cur}, TS_i^{k-1}) + \frac{L}{\phi_i}.$$

This approach reduces the complexity of the algorithm greatly. However, the price paid is the reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of sessions that share the outgoing link [13]. Thus, the worst-case delay of a session can no longer be controlled just by controlling its reservation, as is possible in PGPS. The higher end-to-end delay also affects the burstiness of sessions within the network, increasing the buffer requirements.

The VirtualClock scheduling algorithm provides the same end-to-end delay and burstiness bounds as 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 [2]. A scheduling algorithm that combines the delay and burstiness behavior of PGPS, simple timestamp computations, and bounded unfairness, has so far remained elusive. Our objective in this paper is to develop an analytical framework for the design of such algorithms and, in particular, present a novel scheduling algorithm based on this framework, called *Frame-based Fair Queueing*.

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.

The rest of this paper is organized as follows: In Section 2, we present some definitions and a brief summary of the concept of *Latency-Rate Servers* (or $\mathcal{LR}$-servers) [13]. $\mathcal{LR}$-servers provide a general framework for modeling the worst-case behavior of scheduling algorithms. All work-conserving servers known to us can be modeled using this framework. In Section 3, we define a class of scheduling algorithms, called *Rate-Proportional Servers* which provide the same worst-case delay behavior as GPS, but allow the design of algorithms with a wide range of fairness characteristics. We derive bounds on the end-to-end delay and burstiness in a network of rate-proportional servers and analyze their fairness behavior. In Section 4, we define the frame-based fair queueing algorithm and show that it is a rate-proportional server. The basic algorithm is defined using the fluid model, but its extension to the packet-level version is straightforward. In Section 5 we present two implementation of the frame-based fair queueing algorithm, the first for a general packet network with variable-size packets and the second for ATM networks with fixed-size cells. The latter takes into account the constraints of a hardware implementation. Finally, some concluding remarks are presented in Section 6. Appendix A contains the proofs of many lemmas and theorems, while Appendix B provides some results from simulations of the FFQ algorithm.

2 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.

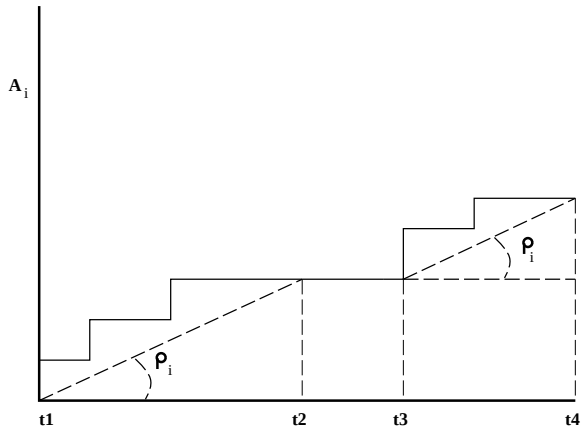

Figure 2.1: Intervals $(t_1, t_2]$ and $(t_3, t_4]$ are two different busy periods.

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

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 were serviced with exactly the guaranteed rate, it would be remain continuously backlogged (Figure 2.1). 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.

For a given session-i backlogged period, the corresponding busy period can be longer. In addition, if $(s_1, f_1]$ is a busy period for session i, multiple backlogged periods may occur in the actual system during the interval $(s_1, f_1]$. The beginning of a busy period is always caused by the

arrival of a packet into the system. Since we are interested in a worst-case analysis of the system, the session busy period provides us a convenient means to bound the queueing delays within the system.

In [13], 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 $W_{i,j}^{\mathcal{S}}(\tau,t)$ 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, $W_i^{\mathcal{S}}(\tau,t)$, may actually be more than $W_{i,j}^{\mathcal{S}}(\tau,t)$ 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,*

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

$\Theta_i^{\mathcal{S}}$ 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 $\Theta_i^{\mathcal{S}}$ represents the worst-case delay seen by a session-i packet arriving into an empty queue. For a fluid-model server, this is the worst-case delay until the first bit of the packet is transmitted; for a packet-by-packet server, $\Theta_i^{\mathcal{S}}$ denotes the maximum delay before the last bit of the packet is serviced. 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 $W_{i,j}^{\mathcal{S}}(\tau,t)$ may be a step function in a packet-by-packet scheduler. As in the case of $W_i(\tau,t)$, we update $W_{i,j}^{\mathcal{S}}(\tau,t)$ 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 $W_{i,j}^{\mathcal{S}}(\tau,t)$ may be continuous.

To determine end-to-end delay bounds, we assume that traffic from session i at the source is leaky-bucket shaped [14]. 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 [13].

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.

Server	Latency	Fairness	Complexity
GPS	0	0	-
PGPS	$\frac{L_i}{\rho_i} + \frac{L_{max}}{r}$	$\max(\frac{L_i}{\rho_i} + C_i, \frac{L_j}{\rho_j} + C_j,$ $\frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}, \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j})$, 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 2.1: 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.

This theorem allows us to calculate bounds on end-to-end delays and buffer requirements for an arbitrary topology network where the only constraint is that individual switches use scheduling algorithms belonging to the class $\mathcal{LR}$. Furthermore, all known work-conserving schedulers — such as GPS, PGPS, Weighted Round Robin, Self-Clocked Fair Queueing, VirtualClock and Deficit-Round-Robin — have been shown to be $\mathcal{LR}$-servers [13]. In Table 2 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 [4, 2]. 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, 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 [12] introduced such a function and called it *virtual time*. Virtual time of a backlogged session is a function whose rate of growth at each instant is exactly the rate of 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 is 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, denoted by $S_i(t_1, t_2)$. If the potential of an idle connection is increased by the service it missed, it is easy to see that, when the connection becomes 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.

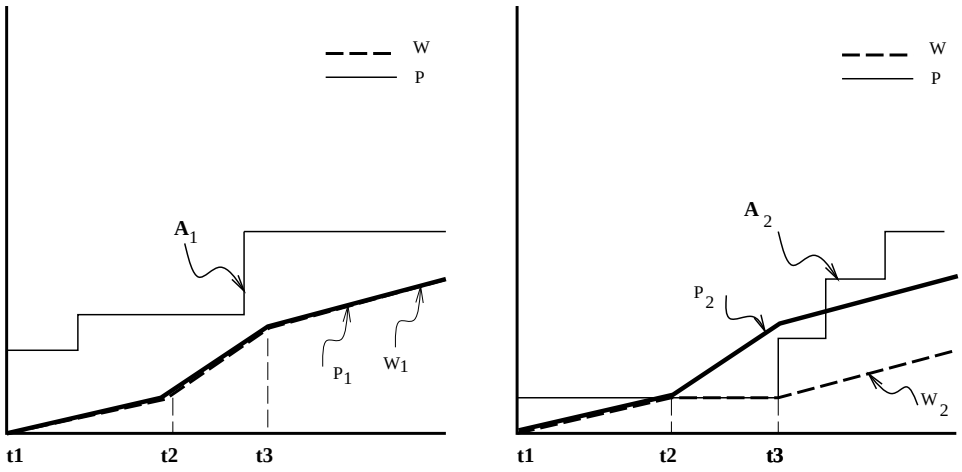

Figure 2.2: Evolution of the potential and offered service for two connections in the GPS multiplexer.

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

For example, the GPS server initializes the potential of 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

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

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

The concept of potential is illustrated with respect to a GPS scheduler in Figure 2.2. Let us assume that only two connections with rates $\rho_1 = \rho_2$ are continuously serviced for the interval of time $(t_1, t_2]$. By the definition of the GPS multiplexer they are serviced with rates proportional to the reserved, and therefore their potentials increase by exactly the same amount. During the interval $(t_2, t_3]$, no traffic arrives for connection 2 and it is thus receiving no service. Connection 1 is exclusively serviced during this interval, and its potential is increasing by the normalized service it receives. Traffic from connection 2 arrives at the server again at time t_3 and the two connections are again serviced proportional to their reservations. Since connection 2 was absent from the system during the interval $(t_2, t_3]$ it lost some service compared to the other connection that was busy. The service it lost is equal to the service that the other connection received during the same interval. Connection 2 will never receive this service. We therefore see that during the interval $(t_2, t_3]$ the potential of connection 2 should increase by exactly the same amount as that of connection 1 although it is not in the system. Thus, when connection 2 becomes backlogged again, the potential of the two connections will be equal, and they will be serviced proportional to their requests. If we

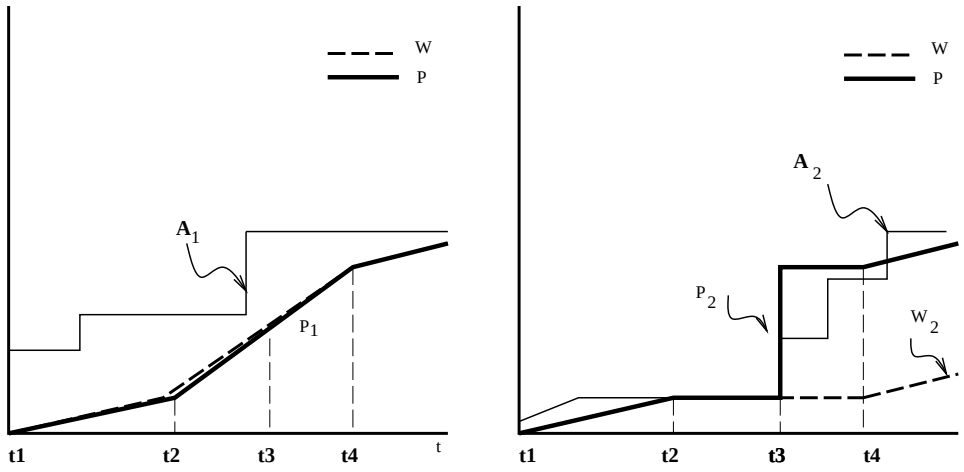

Figure 2.3: Evolution of the potential and offered service for two connections in the SCFQ multiplexer.

take into account this definition of the potential, the scheduling algorithm can be defined as the process that tries to equalize the potential of all backlogged connections and adjusts the potential of the connections when they are not in the system.

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.

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 potential of the connection that is currently being serviced. Packets are transmitted in increasing order of their finishing potential. Consider again the example we used earlier to present the evolution of the potential function in a GPS server. Assume a fluid-model server based on the SCFQ algorithm. The evolution of the potentials of the two connections is shown in Figure 2.3. The objective of the algorithm is to service the backlogged connections in such a way that their potentials will be equalized. Therefore, if a connection has a lower potential than others it will be exclusively serviced until its potential catches up with the potentials of others. In Figure 2.3, connection 2 becomes backlogged again at time t_3, and is assigned a potential equal to the finishing potential of the packet being serviced, $P_1(t_4)$. Thus, connection 2 will receive no service until time t_4, and will be be serviced at a rate proportional to its reservation after t_4. This behavior is different from that in GPS, where an idle connection starts to receive service immediately when it becomes backlogged.

The above example illustrates that, 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 for one packet to be transmitted from each of the other connections before it can be

serviced. This results in a latency that is proportional to the number of active connections. Thus, since the potential of a newly backlogged connection is set to the system potential, the system potential should not be allowed to exceed the potential of backlogged connections to achieve zero latency in a fluid server. Although we have used a fluid server to illustrate this point, the concept applies to a packet-by-packet server as well. In the next section we formalize these intuitive results, and define a class of schedulers to achieve low latency.

3 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 schedules based on the fluid model and later extend the definition to the packet-by-packet version. These schedulers are characterized by their service discipline which adjusts the instantaneous service rate to individual backlogged connections so as to equalize their potentials. In addition, the definition also requires that the system potential $P(t)$ be maintained at or below the potential of any connection backlogged at time t, at every instant of time the server is busy. This ensures that a newly backlogged connection acquires a starting potential not higher that of any other connection currently backlogged in the system, enabling it to receive service immediately. Thus, a rate-proportional server is a zero-latency server. However, beyond this constraint, we do not define exactly how the system potential function $P(t)$ is synthesized, giving rise to a range of possible scheduling algorithms in this class. For example, GPS and VirtualClock are rate-proportional servers, but their system-potential functions are quite different. Self-clocked fair queueing, on the other hand, is not a rate-proportional server since it does not meet the constraint on the system-potential function.

We can now define the RPS class of scheduler formally. 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*

$$P_i(\tau) = \max(P_i(\tau-), P(\tau-)) \tag{3.1}$$

   (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,*

$$P_i(t) = P_i(\tau) + \frac{W_i(\tau, t)}{\rho_i} \tag{3.2}$$

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 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,*

$$P(t) \leq \min_{j \in B(t)} (P_j(t)). \tag{3.3}$$

4. *Connections are serviced at each instant t according to their instantaneous potentials as per the following rules:*
   (a) *Among the 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 always greater 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.

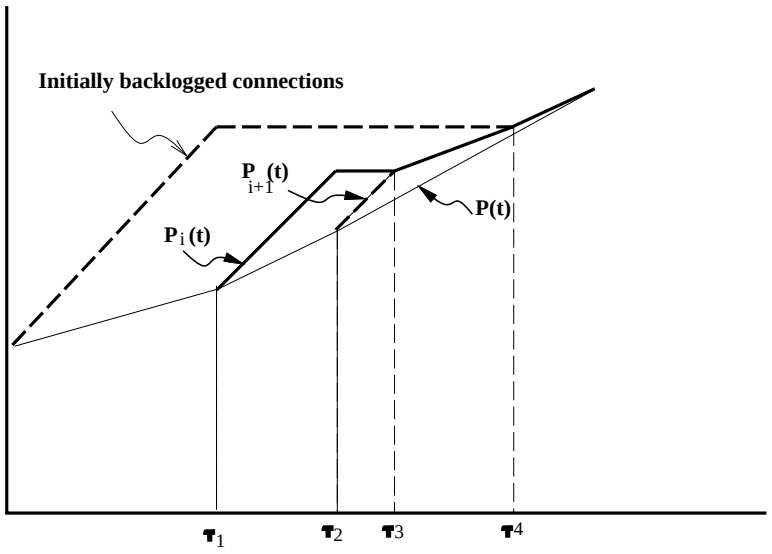

Figure 3.1: An example illustrating the evolution of potential functions in a rate-proportional server.

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.

We can view the evolution of the potential function as in Figure 3.1. Assume that the system potential is always maintained below the potential of every backlogged connection. At time τ_1, connection i i becomes active and receives all the bandwidth, trying to achieve the same potential as the rest of the connections. At time τ_2 a second connection $i+1$ becomes active, and the service of i is temporarily suspended. The potentials of the two new connections become equal at τ_4; during the interval $(\tau_3, \tau_4]$, each of them receives service proportional to its reservation so that their potentials remain equal. That is,

$$\frac{W_i(\tau_3, \tau_4)}{\rho_i} = \frac{W_{i+1}(\tau_3, \tau_4)}{\rho_{i+1}}$$

At τ_4, the potentials of i and $i+1$ become equal to that of other connections already backlogged in the system; therefore, from τ_4, all backlogged connections in the system receive service proportional to their allocated rates. If another new connection becomes active after time τ_4, service to all the connections will be suspended until the new connection reaches the same potential. In addition, if a connection finishes service, the instantaneous service rates of other backlogged connections will increase because of the work-conserving nature of the scheduler. However, a connection may be temporarily suspended if it has received more than its allocated bandwidth earlier during the same active period.

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 lemma is given in Appendix A. 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 Appendix A. 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 is provided in Appendix A.

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.

3.1 Packet-by-Packet Rate-Proportional Servers

In the previous section we defined the rate proportional servers using a fluid-model, where packets from different connections can be served at the same time with different rates. However, in a real system only one connection can be serviced at each time and in addition packets can not be split in smaller units. 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 [2], 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,*

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

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, backlogged in the fluid server at time t, 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 Appendix A. 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 Appendix A, but we provide some intuition here on these bounds. Consider any session i, backlogged in both servers. By Lemma 3, any backlogged session in the packet-by-packet server may lag in service by as much as L_{max} from the fluid server. Thus, in an extreme case, every backlogged session excluding i may be lagging in service by L_{max} in the packet-by-packet server. Since the server is work-conserving, session i can therefore be ahead in the packet-by-packet server by as much as $(V-1)L_{max}$, where V is the number of sessions sharing the outgoing link.

The bound of $(V-1)L_{max}$ may be too loose in many cases. The second bound in the lemma provides a much tighter bound in those cases. To illustrate this bound, let us assume that

$$\frac{L_i}{\rho_i} = \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}).$$

Assume two packets arrive at the server simultaneously, one from session i and the other from a second session j. Assume that the packets are assigned the same finishing potential. If the packets start service in the fluid server at time t, they also finish service simultaneously at time $t + L_i/\rho_i$. If the packet-by-packet transmits the session-j packet first, the service received by session j in the packet-by-packet server can be ahead by $\left(\frac{L_i}{\rho_i}\right)\rho_j$. This reasoning gives rise to the second upper-bound of Lemma 5.

3.2 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

$$\begin{aligned} W_{i,j}^P(\tau, t) &\geq W_{i,j}^F(\tau, t - \frac{L_{max}}{r}) - L_i \\ &\geq \max(0, \rho_i(t - \tau - \frac{L_{max}}{r}) - L_i), \quad \text{by Theorem 2} \\ &\geq \max(0, \rho_i(t - \tau - \frac{L_{max}}{r} - \frac{L_i}{\rho_i})) \end{aligned} \tag{3.4}$$

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.

3.3 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. This behavior can be illustrated with respect to the VirtualClock algorithm. The system potential in VirtualClock grows at the rate of real time, regardless of the potentials of the sessions in the system. Thus, the potential of a connection receiving more service than its reserved rate will continue to diverge from the system potential. Should an idle connection become active later, the connection that received the excess service will be penalized severely. This shows that, to avoid short-term unfairness, the system potential should be maintained close to that of backlogged connections receiving service. We will formalize this idea and show that if the difference between the system potential and those of individual backlogged sessions is bounded, the unfairness is also bounded.

In VirtualClock, the difference between the system potential and the potential of individual backlogged connections cannot be bounded. Thus, the unfairness is also not bounded. This can be seen as a result of the scheduler performing an averaging process on the rate of service provided to individual sessions. In VirtualClock, the averaging interval can be arbitrarily long. The GPS scheduler, on the other hand, occupies the opposite extreme where no memory of past bandwidth usage of connections is maintained. Every backlogged connection has the same potential at all times in a GPS server, giving rise to its ideal fairness behavior. In practice, the scheduling algorithm must trade off short-term unfairness with other desirable properties such as low latency and ease of implementation.

The ideal fairness behavior of GPS is compromised in self-clocked fair queueing, but the difference in potentials is still bounded. Therefore, SCFQ can be considered as a fair scheduling algorithm. However, SCFQ is not a rate-proportional server as it allows the system potential to exceed that of a backlogged connection, resulting in worse delay behavior.

There is no common accepted method for estimating the fairness of a scheduling algorithm. In general, we would like the system to always serve connections proportional to their reservations and never penalize connections for bandwidth they received earlier, The measure of fairness that we will use is an extension of the definition presented for SCFQ [12]. 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,

$$\left| \frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j} \right| \leq \mathcal{FR}, \tag{3.5}$$

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 Appendix A. 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,*

$$\left| \frac{\hat{W}_i(\tau,t)}{\rho_i} - \frac{\hat{W}_j(\tau,t)}{\rho_j} \right| \leq \max(\Delta P + \frac{L_i}{\rho_i} + C_i, \Delta P + \frac{L_j}{\rho_j} + C_j, \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}, \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}). \tag{3.6}$$

A proof of this theorem can be found in Appendix A. 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. (3.6).

Corollary 3: *For a PGPS scheduler,*

$$\left|\frac{\hat{W}_i(\tau,t)}{\rho_i} - \frac{\hat{W}_j(\tau,t)}{\rho_j}\right| \leq \max(\frac{L_i}{\rho_i} + C_i, \frac{L_j}{\rho_j} + C_j, \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}, \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}).$$

It can be shown that the above bound is tight. For example, consider the case where connection i is already backlogged in the system, and assume that connection j becomes backlogged at time τ. Then, connection i may have received an additional amount of service equal to C_i in the PGPS server. If $P_i(\tau)$ is the potential of connection i at time τ, the finishing potential of the next packet of connection i is $F_i \leq P_i(\tau) + C_i + \frac{L_i}{\rho_i}$. Connection j will see a starting potential of $P_j(\tau) = P_i(\tau)$, and is therefore exclusively serviced until the timestamps of its packets become greater than F_i. The total normalized service that connection j may receive while connection i is waiting is bounded by $C_i + \frac{L_i}{\rho_i}$.

4 Frame-based Fair Queueing

Our analysis of rate-proportional servers in the previous section was based only on the properties of the potential functions. We established that every rate-proportional server belongs to the class of $\mathcal{LR}$-servers with zero latency and showed that their unfairness can be bounded by bounding the difference between the system potential and the potentials of backlogged connections. In this section, we introduce a novel scheduling algorithm called frame-based fair queueing (FFQ) and prove that it is a rate-proportional server. The algorithm is defined in this section in terms of the fluid model. Later, in the next section, we will present a self-contained approach for implementing a packet-by-packet version of the algorithm without requiring the parallel simulation of the fluid-model.

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

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

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

$$L_i \leq \phi_i. \tag{4.1}$$

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,

$$P_i(t) \geq f(t) + T, \quad \forall i \in B(t), \tag{4.2}$$

   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

$$f(\tau) = f(\tau-) + T, \tag{4.3}$$

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

$$P(\tau) = \max(P(\tau-), f(\tau)). \tag{4.4}$$

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

$$P(t) = P(\tau) + (t - \tau), \tag{4.5}$$

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.

4.1 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 prove a sequence of two lemmas to classify frame-based fair queueing as a rate-proportional server.

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,*

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

Proof: Assume that the busy period under observation started at time 0. If no frame updates occurred during the interval $(t_1, t_2]$, then the lemma is true by Eq. (4.5). Now consider the case when one or more frame updates occurred during the interval $(t_1, t_2]$. Let $\tau_1, \tau_2, \ldots, \tau_k$ be the instants in this interval just after a frame update, and $\tau_1-, \tau_2-, \ldots, \tau_k-$ the corresponding instants just before the update. Then, by equations (4.4) and (4.5),

$$\begin{aligned} P(t_2) &= P(\tau_k) + (t_2 - \tau_k) \\ &\geq P(\tau_k-) + (t_2 - \tau_k) \\ &\geq P(\tau_{k-1}) + (t_2 - \tau_{k-1}). \end{aligned}$$

Proceeding similarly,

$$\begin{aligned} P(t_2) &= P(\tau_1) + (t_2 - \tau_1) \\ &\geq P(\tau_1-) + (t_2 - \tau_1) \\ &\geq P(t_1) + (t_2 - t_1). \end{aligned}$$

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

$$P(t) \leq P_i(t), \quad \forall i \in B(t). \tag{4.6}$$

Proof: We will prove that, if $P(t) \leq P_i(t)$ at any instant t during a system busy period, then $P(t+\Delta t) \leq P_i(t+\Delta t)$. Since $P(0) = P_i(0)$, this will provide an inductive proof for all instants of time at which the system is busy.

We need to consider two cases:

Case 1: No frame updates occurred during the interval $(t, t+\Delta t)$. Assume, if possible, that $P_i(t+\Delta t) < P(t+\Delta t)$, for some connection i. We can choose Δt as the minimum interval of time such that $P_i(t) \geq P(t)$ and $P_i(t+\Delta t) < P(t+\Delta t)$. Then i is a session with minimum potential during the interval $(t, t+\Delta t]$. Therefore, it is serviced with rate at least ρ_i during this interval. Thus, the potential of session i at $(t+\Delta t)$ must be at least

$$P_i(t) + \frac{\rho_i \Delta t}{\rho_i} \geq P(t) + \Delta t \geq P(t+\Delta t).$$

Thus, the result is true by contradiction.

Case 2: A frame update occurred at time t. Let $t-$ denote the instant just before the update and t the instant just after the update. Then, by Eq. (4.2),

$$P_i(t-) \geq f(t-) + T. \tag{4.7}$$

The new system potential after the update is given by

$$P(t) = \max(P(t-), f(t-) + T). \tag{4.8}$$

Using equations (4.7) and (4.8), as well as the fact that $P_i(t-) \geq P(t-)$,

$$P_i(t) \geq P(t). \tag{4.9}$$

□

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

Proof: It is sufficient to prove that frame-based fair queueing has all the properties of Definition 5. Lemmas 6 and 7 prove that the system potential function satisfies the two main conditions imposed by the definition. The rest of the conditions are satisfied by the definition of the algorithm. Therefore, 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.

4.2 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. We can state the following Lemma:

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

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

Proof: While a connection is backlogged in the FFQ server, its potential is increasing by the normalized service offered to it. The system potential, on the other hand, is increased in two cases. While the frame is not changing it is increased by the real time, and when the frame changes it becomes at least equal to the starting potential of the current frame. Let us assume that the current time is t. The next frame-update will occur after the time when all backlogged connections have crossed the current frame. Since $\phi_i > L_i$, this will occur before the potential of any connection becomes greater than $f(t)+2T$. The largest difference between the system potential and a connection potential will appear just before the frame update. At this time

$$P_i(t) \leq f(t) + 2T, \tag{4.10}$$

and

$$P(t) \geq f(t) + \frac{\phi_i}{r}. \tag{4.11}$$

Subtracting Eq. (4.11) from (4.10),

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

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 $2T - \frac{\phi_i}{r}$. 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,*

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

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.

5 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 will first describe a general algorithm for use with variable-size packets and then present an even more simplified version for ATM networks where all packets are of the same size.

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 $f(t) + T$.

Proof: We will prove the lemma by contradiction. Let us denote with i the connection with the minimum potential in the fluid server and let us assume that the potential of connection i is less than $f(t) + T$. Connection i has received until time t more service in the packet-by-packet server than in the fluid-server. By Lemma 4, there is another connection k with potential $P_k(t) \leq P_i(t)$ that has received less service in the packet-by-packet server than in the fluid-server. Let s_k be the starting potential of the packet that is being serviced in the fluid-server at time t from connection k. Then $s_k \leq P_k(t)$ and thus $s_k \leq f(t) + T$. This is a contradiction. □

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. This can be seen better by an example. Let us assume that the frame size F is set to 100 cells. Assume that connections $1, 2, \ldots, 50$ reserved $1/100$ of the output link bandwidth and connection 0 reserved half of the link bandwidth. At time 0, a packet of size 1 arrives for each of the 50 connections, and 50 packets of size 1 arrive for connection 0. The timestamps of the packets of connection 0 will be $2, 4, \ldots, 100$; and the timestamp of the packet of

each if the other connections will all be 100. This means that all packets from connection 0 may be serviced first in the packet-by-packet system. At this time the potential of all connections in the fluid-system will be equal to 50. Only at the time that all packets with finish time equal to 100 finish service under the packet-by-packet system, will the potential of all connections in the fluid-system be 100 as well. At this time, all backlogged connections will have a packet with a starting potential of greater or equal to 100, and the frame can be updated.

5.1 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. On the arrival of a packet the following algorithm is executed:

$temp = P +$ (Time since last update of P)$/F$
$start \leftarrow \max(TS_i(k-1), temp)$
$TS_i(k) \leftarrow start + \frac{L_i(k)}{\phi_i}$
if $(\lfloor start \rfloor < \lfloor TS_i(k) \rfloor)$ **then** (if finishg potential is in next frame)
    B$[\lfloor start \rfloor] \leftarrow$ B$[\lfloor start \rfloor] + 1$
    mark packet
endif

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 this 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

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

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.

On the departure of a packet, the following algorithm is executed to update the state of the system. We assume that the current packet being transmitted is the kth packet of connection i.

```
P ← P + L_i(k)/F (update system potential)
if (frame-index < ⌊TS_i(k)⌋) then
    if (B[frame-index] > 0) then
        if (current packet is marked ) then
            B[frame-index] ← B[frame-index] − 1
        end if
    end if
    if (B[frame-index] = 0) then (update frame)
        frame-index ← ⌊TS_i(k)⌋
        P ← max(frame-index, P)
    end if
end if
```

The system potential is first increased by the transmission time of the current packet. The variable *frame-index* keeps track of the index of the frame currently in progress. If the packet currently being transmitted has a finishing potential within the next frame and 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, and both the frame number and the system potential can be updated.

5.2 Implementation for ATM Networks

Although the algorithm described above can be applied to an ATM network, in this section we present a simplified version suitable for hardware implementation. In ATM networks the available time for completing a scheduling decision is very short. At SONET OC-3 speeds the transmission time of a cell is less than 2.6 μs. For higher speeds the available time is even less. Therefore, it is desirable that the scheduling algorithm be implementable without floating-point operations.

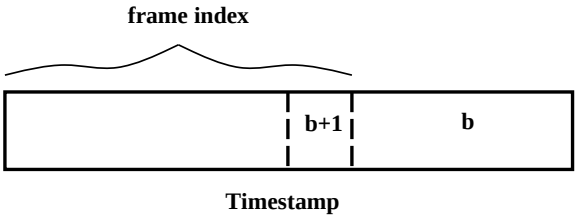

Figure 5.1: Timestamp is formed by the index of the current frame plus an offset. A frame-crossing point of the connection is detected when the $(b+1)$th bit of the timestamp flips during the timestamp computation.

The time axis is again split in frames, with a maximum frame size of F. 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. Therefore, F is the time required to transmit F ATM cells on the output link. As before, let us assume that a fraction r_i of the output link bandwidth is allocated to connection i. Then, during a frame there are ϕ_i credits allocated to connection i, where

$$\phi_i = \lceil F \times r_i \rceil.$$

This means that no more than ϕ_i cells may be transmitted from connection i during one frame. If all connections were busy during a frame period, then by the end of the frame each connection would have sent exactly ϕ_i cells, thus satisfying its bandwidth reservation, and the size of the frame would be F. However, if some connections are absent during a frame, then the real time at which all of the active connections complete sending their cells belonging to the frame will occur earlier.

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. When the server is idle the system potential is reset to zero.

The variable P that denotes the system potential function in the algorithm is composed of two fields, as shown in Figure 5.1. The b least significant bits indicate the time that passed since the beginning of the frame and the remaining bits indicate the index of the current frame. A timestamp $TS_i(k)$ is associated with the kth cell of connection i. The format of the timestamp is the same as that of the system potential function P. That is, the b least significant bits indicate the time within the frame, and the most significant bits indicate the frame during which this cell must depart the system. If a cell has a timestamp whose frame index field is larger than that of the previous cell, it marks the point at which the connection crosses a frame. When such as cell is serviced, subsequent cells from the connection will be treated as belonging to the next frame. The frame crossing point is detected when, during the calculation of the timestamp, the $(b+1)$th bit of the timestamp flips.

In an ATM switch, cell arrivals can be processed at the end of transmission of each cell. On the arrival of the kth cell in the server, the timestamp $TS_i(k)$ associated with the cell is calculated according to the following algorithm, The operation $x >> y$ indicates shift of operand x by y bits. The constant b is equal to $\log_2 F$.

$start \leftarrow \max(TS_i(k-1), P)$ (get starting potential of cell)
$TS_i(k) \leftarrow start + \frac{1}{\rho_i}$ (compute timestamp)
if $((start >> b) < (TS_i(k) >> b))$ **then** ($(b+1)$th bit of timestamp flipped)
    B$[start >> b] \leftarrow$ B$[start >> b] + 1$
    mark cell
endif

Cells are serviced in increasing order of their timestamps. The cells that represent frame-crossing points for the connections are marked. As in the general version of the algorithm, a counter is decremented during the transmission of each such cell, and the frame is updated when the counter becomes zero. The following algorithm summarizes the actions performed to update the system potential when a cell departs the server.

$P \leftarrow P + 1$
if $(frame\text{-}index < (TS_i(k) >> b))$ **then**
    if $(B[frame\text{-}index] > 0)$ **then**
        if (cell is marked) **then**
            $B[frame\text{-}index] \leftarrow B[frame\text{-}index] - 1$
        end if
    end if
    if $(Bframe\text{-}index] = 0)$ **then**
        $frame\text{-}index \leftarrow TS_i(k) >> b$
        $P \leftarrow \max(\text{frame-index} << b, P)$
    end if
end if

The algorithm is essentially the same as the one presented in the previous section. However, we take advantage of the fact that only integers are used, and replace the floating point operations in the original algorithm by shifts. In addition, the priority list implementation can be simplified as well by using the techniques described in [16].

6 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.

The design of a traffic scheduling algorithm involves an inevitable tradeoff among its delay, complexity of implementation, and fairness. Among the three, the delay and implementation complexity are clearly the most important criteria for the use of an algorithm in a real system.

In addition to minimizing the end-to-end delay in a network of servers, the delay behavior of an ideal algorithm must include (i) insensitivity to traffic patterns of other sessions (isolation), (ii) delay bounds that are independent of the number of sessions sharing the outgoing link, and (iii) ability to control the delay bound of a session by controlling only its bandwidth reservation. SCFQ simplifies the timestamp computation of PGPS considerably, but at the expense of seriously degrading its delay behavior. VirtualClock, on the other hand, has both simple implementation and delay bounds identical to that of PGPS, but its fairness behavior makes it unacceptable for many applications. Frame-based fair queueing maintains the same delay behavior of PGPS and has a simple implementation, still providing fairness to individual sessions.

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.

References

[1] D. Verma, D. Ferrari, and H. Zhang, "Guaranteeing delay jitter bounds in packet switching networks," in *Tricomm 91*, April 1991.

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

[3] 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.

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

[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*, September 1995.

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

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

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

[11] J. Davin and A. Heybey, "A simulation study of fair queueing and policy enforcement," *Computer Communication Review*, vol. 20, pp. 23–29, October 1990.

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

[13] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," Tech. Rep. UCSC-CRL-95-38, U.C. Santa Cruz, Dept. of Computer Engineering, July 1995.

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

[15] J. Rexford, A. Greenberg, and F. Bononi, "A fair leaky-bucket shaper for atm networks." AT&T unpulbished report.

[16] J. L. Rexford, A. Greenberg, and F. Bonomi, "Hardware efficient fair queueing architectures for high-speed networks," in *Proceedings of INFOCOM 96*, IEEE, 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] A. Varma and D. Stiliadis, "FAST: an FPGA-based simulation testbed for ATM switching systems," in *Interop'95 Engineer Conference*, April 1995.

Appendix A: Proofs of Main Results

Proof of Lemma 1: We will prove the lemma by contradiction. Assume, if possible, that time τ is not the beginning of an active period. We have two cases:

Case 1: Time τ belongs in inactive period. Since connection i was not busy before time τ and it becomes busy at time τ, a packet must have arrived. But then, the potential of the connection would have become equal to the system potential and thus τ is the beginning of an active period.

Case 2: An active period started at time $\tau_0 < \tau$ and is currently in progress. Then, for every time $t \in (\tau_o, \tau]$, we must have

$$P_i(t) \geq P(t). \tag{A.1}$$

During the interval $(\tau_0, \tau]$, the potential of connection i has only increased by the normalized service offered to connection i. Therefore, at any time t during the interval $(\tau_0, \tau]$,

$$P_i(t) - P_i(\tau_0) = \frac{W_i(\tau_0, t)}{\rho_i}. \tag{A.2}$$

But, since τ_0 is the beginning of an active period,

$$P_i(\tau_0) = P(\tau_0). \tag{A.3}$$

From equations (A.1) and (A.3),

$$\begin{aligned} P_i(t) - P_i(\tau_0) &\geq P(t) - P(\tau_0) \\ &\geq (t - \tau_0). \end{aligned} \tag{A.4}$$

Therefore, from equations (A.2) and (A.4),

$$W_i(\tau_0, t) \geq \rho_i(\tau - \tau_0). \tag{A.5}$$

Before time τ_0, the system was not backlogged and therefore we can write:

$$A_i(\tau_0, t) \geq W_i(\tau_0, t) \geq \rho_i(t - \tau_0).$$

Thus, time τ_0 belongs in the same busy period as any time t in the interval $(\tau_0, \tau]$. Therefore, time τ cannot be the beginning of a busy period.

□

Proof of Lemma 2: Let us consider any time t during the connection active period. By the definition of active period,

$$P_i(t) \geq P(t), \tag{A.6}$$

and

$$P_i(\tau) = P(\tau). \tag{A.7}$$

From the definition of rate-proportional servers we also know that,

$$P(t) - P(\tau) \geq (t - \tau). \tag{A.8}$$

From equations (A.6), (A.7), and (A.8) we can easily conclude that

$$P_i(t) - P_i(\tau) \geq (t - \tau). \tag{A.9}$$

During an active period, the potential of a connection is only increased by the normalized service offered to it. Therefore,

$$\begin{aligned} P_i(t) - P_i(\tau) &= \frac{W_i(\tau, t)}{\rho_i} \\ &\geq t - \tau. \end{aligned} \tag{A.10}$$

From equations (A.9) and (A.10),

$$W_i(\tau, t) \geq \rho_i(t - \tau) \tag{A.11}$$

□

Proof of Theorem 2: Let us again trace the evolution of the potential function of connection i. We can split the busy period in intervals during which the connection is in active or inactive states. During an inactive period, the connection is not receiving any service and no packets from the connection are backlogged in the system. We will prove the theorem by contradiction. Let us denote with t^* the first time such that

$$W_i(\tau, t^*) < \rho_i(t^* - \tau). \tag{A.12}$$

Assume that t^* belongs to a busy period that started at time τ. We distinguish two cases:
Case 1: Time t^* belongs in an active period. Let us denote with t_a, the time that this active period started. We know from Lemma 1, that $t_a \geq \tau$. Then, since $t^* > t_a$,

$$W_i(\tau, t_a) \geq \rho_i(t_a - \tau). \tag{A.13}$$

From Lemma 2, we also know that for time t^* that belongs in the same active period,

$$W_i(t_a, t^*) \geq \rho_i(t^* - t_a) \tag{A.14}$$

From equations (A.13) and (A.14) we can conclude that

$$W_i(\tau, t^*) \geq \rho_i(t^* - \tau). \tag{A.15}$$

Case 2: Time t^* is part of an inactive period. Consider time $t^* - \Delta t$. At this time, we know that

$$W_i(\tau, t^* - \Delta t) \geq \rho_i(t^* - \Delta t - \tau). \tag{A.16}$$

Since the connection is in an inactive period, there are no packets backlogged from that connection, and therefore,

$$W_i(\tau, t^* - \Delta t) = A_i(\tau, t^* - \Delta t). \tag{A.17}$$

In addition, no packets were serviced from the connection in the interval $(t^* - \Delta t, t^*]$, or,

$$W_i(\tau, t^* - \Delta t) = W_i(\tau, t^*). \tag{A.18}$$

It is clear that no arrivals of session-i packets occurred during the interval $(t^* - \Delta t, t^*)$; if there was an arrival during this interval, the connection would have entered an active period. Thus,

$$A_i(\tau, t^* - \Delta t) = A_i(\tau, t^*). \tag{A.19}$$

From equations (A.12), (A.17), (A.18), and (A.19) we can conclude that

$$A_i(\tau, t^*) < \rho_i(t^* - \tau). \tag{A.20}$$

This means that time t^* does not belong in the same busy period as $t^* - \Delta t$.

□

Proof of Lemma 3: Assume that a system-busy period starts in both servers at time 0. Let the packets transmitted by the PRPS system during the system-busy period be numbered $1, 2, \ldots, k, \ldots$ in their order of transmission. Since both servers are work-conserving, the system-busy period must end in both at the same time. However, the order in which the last bit of packets leave the system in the fluid server can be different from that in the packet-by-packet server because multiple packets can be in service at the same time in the former. Therefore, we need to consider two cases:

Case 1: The last bits of packets $1, 2, \ldots, k-1$ left the fluid server before the last bit of the kth packet. Then, the time of departure of the last bit of the kth packet in the fluid server, denoted by t_k^F, must satisfy

$$t_k^F \geq \frac{1}{r} \sum_{j=1}^{k} L_j, \tag{A.21}$$

where L_j is the size of the jth packet and r the service rate on the outgoing link.

The corresponding departure time of the last bit of the kth packet in the packet-by-packet server is given by

$$t_k^P = \frac{1}{r} \sum_{j=1}^{k} L_j. \tag{A.22}$$

From (A.21) and (A.22),

$$t_k^P \leq t_k^F. \tag{A.23}$$

Case 2: Now consider the case when one or more of the packets $1, 2, \ldots, k-1$ were still in service in the fluid server when the last bit of the kth packet left the server. Among this set of packets, let the packet with the largest index be the mth packet, $m < k$, with a length of L_m. This packet left the packet-by-packet server at time t_m^P and started transmission at $L_m^P - \frac{L_m}{r}$. At this point, packets $m+1, m+2, \ldots, k$ had not arrived in the system; if they had, they would have received timestamps lower than that of packet m, and therefore would have been serviced earlier than m in the packet-by-packet server. Also, the packets $m+1, m+2, \ldots, m+k$ were serviced completely in the fluid server before packet k left the system.

Since packets $m+1, m+2, \ldots, k$ were serviced completely during the interval $(t_m^P - \frac{L_m}{r}, t_k^F]$, we must have

$$t_k^F - (t_m^P - \frac{L_m}{r}) \geq \frac{1}{r} \sum_{j=m+1}^{k} L_j,$$

$$\text{or,} \qquad t_k^F \geq t_m^P + \frac{1}{r} \sum_{j=m+1}^{k} L_j - \frac{L_m}{r}. \tag{A.24}$$

But,

$$t_k^P = t_m^P + \frac{1}{r}\sum_{m+1}^{k} L_j. \tag{A.25}$$

From (A.24) and (A.25), and noting that $L_m \leq L_{max}$,

$$t_k^P \leq t_k^F + \frac{L_{max}}{r}. \tag{A.26}$$

Thus, combining (A.23) and (A.26), we get the upper bound as

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

□

Proof of Lemma 4:

Since both servers are work-conserving, it is clear that if session i receives more service in the packet-by-packet server, then there must be another backlogged session j that has received less service in the interval $(0, t]$. We only need to prove that $P_j(t) \leq P_i(t)$ for one such session j.

We will distinguish two cases:

Case 1: Session i has the maximum potential among the backlogged sessions in the fluid server at time t. In this case $P_j(t) \leq P_i(t)$ for every backlogged session j.

Case 2: There are other sessions at time t with potentials higher than that of i. Let S be the set of sessions with potentials higher than $P_i(t)$ at time t in the fluid server. Then, by the definition of rate-proportional servers, these connections are not receiving service at time t in the fluid server. Let τ be the most recent time when a session from the set S was in service in the fluid server. Then, during the interval $(\tau, t]$, none of the connections in the set S were serviced by the fluid server. Furthermore, the current backlogged period of session i started at or after time τ; otherwise, a connection from the set S would not have been serviced just before τ.

Thus, during the interval $(\tau, t]$, session i received less service in the fluid server than in the packet-by-packet server, and sessions in the set S received no service in the former. Hence, there must be another session j that received more service in the fluid server during the interval $(\tau, t]$. Since $j \notin S$, it follows that $P_j(t) \leq P_i(t)$.

□

Proof of Lemma 5: Both RPS and PRPS are work-conserving servers. Let us assume that a connection i has received more service in the packet-by-packet server than in the fluid-server. In the worst case, every other backlogged session may have received less service in the former. By Lemma 3, any backlogged session in the packet-by-packet server may lag in service by as much as L_{max} from the fluid server. Thus, in an extreme case, every backlogged session excluding i may be lagging in service by L_{max} in the packet-by-packet server. Since the servers are work-conserving, session i must be ahead in service in the packet-by-packet server by an amount equal to the total lag of all other sessions. That is,

$$\hat{W}_i^P(0,t) - \hat{W}_i^F(0,t) \leq (V-1)L_{max}.$$

A tighter bound may be obtained in some cases. Let us denote with t_i^k the time a packet k from session i finishes service in the PRPS system. The maximum difference in service seen by session i between the two servers will occur at time t_i^k. Let L_i^k denote the size of packet k. This packet started service in the PRPS system at time $\tau_i^k = t_i^k - \frac{L_i^k}{r}$. We will distinguish two cases for the time τ_i^k.

Case 1: $\hat{W}_i^P(0,\tau_i^k) \leq \hat{W}_i^F(0,\tau_i^k)$. Then we can write

$$\begin{aligned}\hat{W}_i^P(0,t_i^k) &= \hat{W}_i^P(0,\tau_i^k) + L_i^k \\ &\leq \hat{W}_i^F(0,\tau_i^k) + L_i^k \\ &\leq \hat{W}_i^F(0,t_i^k) + L_i^k \\ &\leq \hat{W}_i^F(0,t_i^k) + \rho_i \max_{1\leq n\leq V}(\frac{L_n}{\rho_n}). \end{aligned} \tag{A.27}$$

The last inequality follows from the fact that

$$\frac{L_i^k}{\rho_i} \leq \max_{1\leq n\leq V}(\frac{L_n}{\rho_n}).$$

Case 2: $\hat{W}_i^P(0,\tau_i^k) > \hat{W}_i^F(0,\tau_i^k)$. Let F_i^k be the finishing potential of packet k of session i in the fluid server. $P_i(t_i^k)$ is the potential of session i in the fluid server at time t_i^k, and F_i^k the corresponding potential in the packet-by-packet server. Then, at t_i^k, session i will have to receive an additional amount of service equal to $\rho_i(F_i^k - P(t_i^k))$ in the fluid server to catch up with the potential in the packet-by-packet server. Thus,

$$\begin{aligned}\hat{W}_i^P(0,t_i^k) - \hat{W}_i^F(0,t_i^k) &= \rho_i(F_i^k - P_i(t_i^k)) \\ &\leq \rho_i(F_i^k - P_i(\tau_i^k)), \quad \text{since } P_i(t_i^k) \geq P_i(\tau_i^k). \end{aligned} \tag{A.28}$$

At time τ_i^k, session i has received less service in the fluid server as compared to the packet-by-packet server. Therefore, by Lemma 4, there is another session j, backlogged in the fluid server at time τ_i^k, with potential $P_j(\tau_i^k) \leq P_i(\tau_i^k)$ that has received more service in the fluid server. Let the packet in service of this session at time τ_i^k in the fluid server be the mth packet. Let S_j^m and F_j^m denote the potentials of session j in the fluid server when this packet begins and ends service, respectively. Then, $S_j^m \leq P_j(\tau_i^k)$. Since the packet has not completed service in the packet-by-packet server, $F_j^m \geq F_i^k$. Thus, we have

$$P_i(\tau_i^k) \geq P_j(\tau_i^k) \geq S_j^m, \tag{A.29}$$

and

$$F_i^k \leq F_j^m. \tag{A.30}$$

Substituting for F_i^k and $P_i(t_i^k)$ in Eq. (A.28) from equations (A.30) and (A.29), respectively,

$$\begin{aligned}\hat{W}_i^P(0,t_i^k) - \hat{W}_i^F(0,t_i^k) &= \rho_i(F_j^m - S_j^m) \\ &\leq \rho_i \frac{L_j}{\rho_j} \\ &\leq \rho_i \max_{1\leq n\leq V}(\frac{L_n}{\rho_n}). \end{aligned} \tag{A.31}$$

This completes the proof of Lemma 5.

□

Proof of Theorem 3: Consider time t_1. Without loss of generality, let us assume that at time t_1, $P_j(t_1) > P_i(t_1)$. Since connection i is backlogged

$$P_i(t_1) \geq P(t_1). \tag{A.32}$$

We also know that

$$P_j(t_1) \leq P(t_1) + \Delta P. \tag{A.33}$$

Since both sessions are backlogged in the interval $(t_1, t_2]$, their potentials in this interval have increased only by the normalized service offered to the two connections. Therefore,

$$P_i(t_2) - P_i(t_1) = \frac{W_i(t_1, t_2)}{\rho_i}, \tag{A.34}$$

and

$$P_j(t_2) - P_j(t_1) = \frac{W_j(t_1, t_2)}{\rho_j}. \tag{A.35}$$

At time t_2, the potential of connection i can not be more than that of connection j. Let us denote their difference with x. Then,

$$x = P_j(t_2) - P_i(t_2) \geq 0. \tag{A.36}$$

From equations (A.32),(A.34), and (A.36),

$$\frac{W_i(t_1, t_2)}{\rho_i} \leq P_j(t_2) - P(t_1) - x. \tag{A.37}$$

Similarly, from equations (A.33) and (A.35),

$$\frac{W_j(t_1, t_2)}{\rho_j} \geq P_j(t_2) - P(t_1) - \Delta P. \tag{A.38}$$

From equations (A.37) and (A.38) we can easily conclude that

$$\frac{W_i(t_1, t_2)}{\rho_i} - \frac{W_j(t_1, t_2)}{\rho_j} \leq \Delta P - x \leq \Delta P. \tag{A.39}$$

If $P_j(t_1) < P_i(t_2)$, we can derive in the same way that

$$\frac{W_j(t_1, t_2)}{\rho_j} - \frac{W_i(t_1, t_2)}{\rho_i} \leq \Delta P. \tag{A.40}$$

Therefore,

$$\left|\frac{W_i(t_1, t_2)}{\rho_i} - \frac{W_j(t_1, t_2)}{\rho_j}\right| \leq \Delta P. \tag{A.41}$$

Thus, if ΔP is finite, the difference in normalized service offered to any two backlogged connections is also bounded. □

Proof of Theorem 4: Let us assume, without loss of generality, that before time τ session i was not backlogged. Note that, since both sessions have an infinite supply of packets from time τ, the timestamp of every packet is calculated from that of the previous packet of the same session. Let us denote with S_i^k and S_j^m, the starting potentials of the first packet in the queue for sessions i and j, respectively. Also, let us denote with F_i^n and F_j^l, the finishing potentials of the last packet serviced from the two sessions before time t_2.

Case 1: $S_i^k \leq S_j^m$. We will separate the problem into two sub-cases: First, let us assume that at least one packet was serviced from session j after time τ. Then, $F_j^{m-1} = S_j^m$ and $F_i^k \geq F_j^{m-1}$. Since the maximum packet size for session i is L_i,

$$S_j^m - \frac{L_i}{\rho_i} \leq S_i^k. \tag{A.42}$$

Now consider the sub-case when no other packet was serviced from j after time τ. Then

$$S_j^m \leq P_j(\tau) + C_j. \tag{A.43}$$

We also know that

$$S_i^k \geq P_i(\tau). \tag{A.44}$$

By subtracting (A.44) from (A.43),

$$\begin{aligned} S_j^m - S_i^k &\leq (P_j(\tau) - P_i(\tau)) + C_j \\ &\leq \Delta P + C_j. \end{aligned} \tag{A.45}$$

Therefore, combining the two subcases,

$$S_j^m - S_i^k \leq \max(\Delta P + C_j, \frac{L_i}{\rho_i}). \tag{A.46}$$

The service offered to session i in the interval $(t_1, t_2]$ is equal to the difference in time-stamps $F_i^n - S_i^k$. Also,

$$F_i^n \leq F_j^l + \frac{L_j}{\rho_j}.$$

Therefore,

$$\begin{aligned} \frac{\hat{W}_i(\tau,t)}{\rho_i} &\leq F_i^n - S_i^k \\ &\leq F_j^l - S_i^k + \frac{L_j}{\rho_j}. \end{aligned} \tag{A.47}$$

Similarly, the normalized service offered to session j is equal to the difference between the time-stamps. That is,

$$\frac{\hat{W}_j(\tau,t)}{\rho_j} = F_j^l - S_j^m. \tag{A.48}$$

Therefore,

$$\frac{\hat{W}_i(\tau,t)}{\rho_i} - \frac{\hat{W}_j(\tau,t)}{\rho_j} \leq S_j^m - S_i^k + \frac{L_j}{\rho_j}$$
$$\leq \max(\Delta P + C_j, \frac{L_i}{\rho_i}) + \frac{L_j}{\rho_j}. \quad \text{(A.49)}$$

Similarly, we can write that

$$S_i^k - S_j^m \leq 0. \quad \text{(A.50)}$$

For session j,

$$F_j^l \leq F_i^n + \frac{L_i}{\rho_i} \quad \text{(A.51)}$$

and

$$\frac{\hat{W}_j(\tau,t)}{\rho_j} \leq F_j^l - S_j^m$$
$$\leq F_i^n - S_j^m + \frac{L_i}{\rho_i}. \quad \text{(A.52)}$$

For session i,

$$\frac{\hat{W}_i(\tau,t)}{\rho_i} = F_i^n - S_i^k. \quad \text{(A.53)}$$

Subtracting (A.53) from (A.51),

$$\frac{\hat{W}_j(\tau,t)}{\rho_j} - \frac{\hat{W}_i(\tau,t)}{\rho_i} \leq (S_i^k - S_j^m) + \frac{L_i}{\rho_i} \quad \text{(A.54)}$$
$$\leq \frac{L_i}{\rho_i}. \quad \text{(A.55)}$$

Case 2: $S_i^k > S_j^m$. In order to estimate a tight bound we will need to separate the problem into several sub-cases.

Subcase (a): The starting potential of session i was estimated by the system potential. Then, this is the first packet that will be serviced from session i after time τ. Since $P_i(\tau) = S_i^k > S_j^m$ and $P_i(\tau) \leq P_j(\tau)$, from the definition of rate-proportional servers, session j received less service in the packet-by-packet server than in the fluid server. Therefore, for the normalized service offered to connection j between time τ and t_2, we can write:

$$\frac{\hat{W}_j(\tau,t_2)}{\rho_j} \leq F_i^n - S_i^k + \frac{L_i}{\rho_i} + \frac{\hat{W}_j^G(0,\tau) - \hat{W}_j^F(0,\tau)}{\rho_j}$$
$$\leq \frac{\hat{W}_i(t_1,t_2)}{\rho_i} + \frac{L_i}{\rho_i} + \frac{L_{max}}{\rho_j}. \quad \text{(A.56)}$$

This means that the maximum normalized service that can be offered to session j is bounded by the maximum increase in the potential of session i plus the excess normalized service that session j received in the packet-by-packet server. Since the service is non-decreasing, we have

$$\hat{W}_j(\tau,t_2) \geq \hat{W}_j(t_1,t_2).$$

Therefore,

$$\frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i} \leq \frac{L_i}{\rho_i} + \frac{L_{max}}{\rho_j}. \tag{A.57}$$

Subcase (b): The starting potential of the kth packet of connection i was computed from the potential of the $(k-1)$th packet. Again, we will have to consider several sub-problems. First of all, let us assume that there was another packet serviced from connection i after time τ. Since $S_i^k = F_i^{k-1}$ and $F_j^m \geq F_i^{k-1}$, we can write

$$S_i^k - \frac{L_j}{\rho_j} \leq S_j^m, \tag{A.58}$$

or equivalently

$$S_i^k - S_j^m \leq \frac{L_j}{\rho_j}. \tag{A.59}$$

Now let us assume that there is no other packet serviced from session i after time τ. Then, the packet-by-packet server has offered more service to session i until time τ than the fluid system. Let us denote with t^* the time at which the $(k-1)$th packet was serviced from session i. If session j was backlogged at time t^*, then

$$F_j^a - \frac{L_j}{\rho_j} \leq F_i^{k-1} \leq F_j^a, \tag{A.60}$$

for some packet a of connection j. But since $F_i^{k-1} = S_i^k$ and $S_j^m \geq F_j^a - \frac{L_j}{\rho_j}$, we can conclude that

$$S_i^k - S_j^m \leq \frac{L_j}{\rho_j}. \tag{A.61}$$

If connection j was not backlogged at time t^*, then when it becomes backlogged at some time $t^* < t \leq \tau$,

$$P_j(t) \geq P_i(t^*) - \Delta P,$$

and

$$S_j^a \geq P_j(t),$$

for some packet a of connection j. The second inequality follows from the fact that the fluid-server can only be behind the packet-by-packet server for connection j at time t^* since the packet-by-packet server is not backlogged. Also, $S_j^m \geq S_j^a$. Therefore,

$$S_j^m \geq P_i(t^*) - \Delta P.$$

We also know from Lemma 5 that

$$F_i^{k-1} = S_i^k \leq P_i(t^*) + C_i.$$

Subtracting,

$$S_i^k - S_j^m \leq \Delta P + C_i \tag{A.62}$$

Therefore, from equations (A.59), (A.61), and (A.62),

$$S_i^k - S_j^m \leq \max(\Delta P + C_i, \frac{L_j}{\rho_j}).$$

For session j, the maximum service offered in the interval $(t_1, t_2]$ will be

$$\begin{aligned}\frac{\hat{W}_j(\tau,t)}{\rho_j} &\leq F_j^l - S_j^m \\ &\leq F_i^n - S_j^m + \frac{L_i}{\rho_i}. \end{aligned} \tag{A.63}$$

For session i,

$$\frac{\hat{W}_i(\tau,t)}{\rho_i} = F_i^n - S_i^k. \tag{A.64}$$

Subtracting,

$$\begin{aligned}\frac{\hat{W}_j(\tau,t)}{\rho_j} - \frac{\hat{W}_i(\tau,t)}{\rho_i} &\leq S_i^k - S_j^m + \frac{L_i}{\rho_i} \\ &\leq \max(\Delta P + C_i, \frac{L_j}{\rho_j}) + \frac{L_i}{\rho_i}. \end{aligned} \tag{A.65}$$

Finally, if session i has received more normalized service in this interval, we can write

$$\begin{aligned}\frac{\hat{W}_i(\tau,t)}{\rho_i} &\leq F_i^n - S_i^k \\ &\leq F_j^l - S_i^k + \frac{L_j}{\rho_j}, \end{aligned} \tag{A.66}$$

and

$$\frac{\hat{W}_j(\tau,t)}{\rho_j} = F_j^l - S_i^m. \tag{A.67}$$

By subtracting and using the assumption that $S_i^k > S_i^m$,

$$\frac{\hat{W}_i(\tau,t)}{\rho_i} - \frac{\hat{W}_j(\tau,t)}{\rho_j} \leq \frac{L_j}{\rho_j}. \tag{A.68}$$

Combining equations (A.49), (A.55), (A.57), (A.65), and (A.68) we can conclude that

$$|\frac{\hat{W}_i(\tau,t)}{\rho_i} - \frac{\hat{W}_j(\tau,t)}{\rho_j}| \leq \max(\Delta P + \frac{L_i}{\rho_i} + C_i, \Delta P + \frac{L_j}{\rho_j} + C_j, \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}). \tag{A.69}$$

A symmetrical argument for the case where session j becomes backlogged after i will complete the proof of the theorem. □

Appendix B: Simulation Results

This appendix provides some simulation results to further illustrate the ability of frame-based fair queueing (FFQ) to provide low session latencies and isolation from other misbehaving flows. We simulated the packet-by-packet version of the algorithm (PFFQ) as applied to a single output port of an ATM switch. For comparison, we also simulated the self-clocked fair-queueing (SCFQ) algorithm since its implementation complexity is comparable to that of FFQ. Our model consists of eight sessions sharing the same output link. The reservation of each of these sessions is shown in Table 2. An ON-OFF traffic model was used to generate traffic within each session. 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. With this model, we measured the delays and bandwidth allocations received by all the sessions. A summary of our results is presented in Table 2. Delays are shown in the table in terms of cell-transmission times. The upper bounds for delay for each session in the two servers, computed using Theorem 1, are as shown in Table 3.

Both the average and maximum delays of session 0, which has reserved 50% of the link bandwidth, are substantially lower in the PFFQ server as compared to the SCFQ server. PFFQ 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 $\frac{2}{0.5} + 1 = 5$. The delays of sessions 2–7 are also lower in the PFFQ server. The higher delays experienced in the SCFQ server are a result of its poor isolation properties. The SCFQ algorithm provided more service to session 1 than its reservation, sacrificing the delays of other channels. In contrast, the PFFQ is able to provide stricter isolation, and does not allow the misbehaving connection to influence the end-to-end delays of the other connections. This example verifies the excellent isolation properties of PFFQ.

An important property of PFFQ is that the maximum latency seen by a session can be controlled by varying its reservation. That is, increasing the reservation results in a corresponding decrease in the latency, affecting both the maximum and average delays in a real system, as is the case in a GPS server. With SCFQ, on the other hand, the delay bound is much less affected by the reservation of the session. The simulation results in Table 4 verify this important property. In this set of simulations, the reservations of the individual sessions are identical to those in Table 2, but all sessions are transmitting within their reservations. Again, both the average and maximum delays seen by session 0 are substantially lower in the PFFQ server as compared to the SCFQ server. The delays of sessions 2–7 are slightly higher in PFFQ owing to their lower reservations, but the differences are small.

Thus, the simulations in this appendix illuminate some of the important properties of the frame-based fair queueing algorithm.

Session	Reserved Bandwidth	Arrival Rate	PFFQ		SCFQ	
			Avg. Delay	Max Delay	Avg. Delay	Max Delay
0	0.500000	0.498	1.003	2.0	3.231	7.0
1	0.062500	0.100	N/A	N/A	N/A	N/A
2	0.062500	0.062	5.343	22.0	13.777	23.0
3	0.062500	0.061	7.394	24.0	14.600	26.0
4	0.078125	0.076	3.726	12.0	9.146	19.0
5	0.078125	0.076	4.265	12.0	9.822	19.0
6	0.078125	0.076	5.305	13.0	10.447	19.0
7	0.078125	0.076	5.985	15.0	10.750	17.0

Table 2.1: Comparison of delays from a simulation of PFFQ 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.

Session	PFFQ	SCFQ
0	5	11
1	33	39
2	33	39
3	33	39
4	27	33
5	27	33
6	27	33
7	27	33

Table 2.2: Analytical delay bounds for the sessions in the simulation.

Session	Reserved Bandwidth	Arrival Rate	PFFQ		SCFQ	
			Avg. Delay	Max Delay	Avg. Delay	Max Delay
0	0.500000	0.498	1.002	2.0	2.480	7.0
1	0.062500	0.062	6.985	14.0	5.598	11.0
2	0.062500	0.062	7.191	20.0	6.391	11.0
3	0.062500	0.061	7.512	24.0	6.952	16.0
4	0.078125	0.076	5.726	16.0	4.146	10.0
5	0.078125	0.076	6.265	16.0	4.822	11.0
6	0.078125	0.076	5.653	16.0	5.255	15.0
7	0.078125	0.076	6.194	16.0	5.603	16.0

Table 2.3: Comparison of delays from a simulation of PFFQ and SCFQ algorithms. The eight sessions shown share the same output link. Delays are measured in terms of cell-transmission times. All sessions are transmitting within their reservations.