Rate-Proportional Servers: A Design Methodology for Fair Queueing Algorithms

Dimitrios Stiliadis
Anujan Varma

UCSC-CRL-95-58
December 1995

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

ABSTRACT

Weighted Fair Queueing is considered as the ideal traffic scheduling algorithm in terms of its delay and fairness properties. Timestamp computations in a Weighted Fair Queueing scheduler serving N sessions have a time complexity of $O(N)$ per packet-transmission time, making its implementation difficult. Efforts in the past to simplify the implementation of Weighted Fair Queueing, such as Self-Clocked Fair Queueing, have resulted in degrading its isolation properties, thus affecting the delay bound. In this paper we present a class of scheduling algorithms — called *Rate-Proportional Servers* (RPS) — with bounds on end-to-end delays, buffer requirements and internal traffic burstiness equal to those of Weighted Fair Queueing. This class of algorithms is based on the notion of the *potential* associated with each connection sharing the same outgoing link, as well as, the *system potential* that tracks the progress of work in the system. We show that, depending on the implementation, different algorithms in the RPS class may have significantly different fairness properties. Network designers can use this methodology to implement efficient fair-queueing algorithms, balancing their fairness with implementation complexity. This work is completed in the sequel of this paper, where we present detailed implementations of two novel traffic scheduling algorithms with $O(1)$ timestamp computations, that exhibit the same delay and fairness properties as those of Weighted Fair Queueing.

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

This research is supported by the NSF Young Investigator Award No. MIP-9257103. A part of this paper will appear in the Proceedings of ACM SIGMETRICS '96.

Contents

List of Figures

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. QoS guarantees can be provided both in conventional packet networks and in broadband ATM (Asynchronous Transfer Mode) networks by the use of proper packet scheduling algorithms in the switches (or routers).

The function of a scheduling algorithm is to select, for each outgoing link of the switch, the packet to be transmitted in the next cycle from the available packets belonging to the flows sharing the output link. Implementation of the algorithm may be in hardware or software. In ATM networks, where information is transmitted in terms of small fixed-size cells, the scheduling algorithm must usually be implemented in hardware within 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. Examples of work-conserving schedulers include Generalized Processor Sharing (GPS) [1], Weighted Fair Queueing [2], VirtualClock [3], Delay-Earliest-Due-Date (Delay-EDD) [4], Weighted Round Robin [5], and Deficit Round Robin [6]. On the other hand, Hierarchical-Round-Robin (HRR) [7], Stop-and-Go queueing [8], and Jitter-Earliest-Due-Date [9] 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 $O(\log V)$ 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 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, subject to 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 bound of a session depends only on the parameters of the session, such as its bandwidth reservation, and is independent

of the behavior of other sessions. A higher end-to-end delay bound usually implies a higher level of burstiness at the output of the scheduler, and consequently requires larger buffers in the switches to avoid packet loss. Therefore, the delay bound affects not only the end-to-end behavior of the session, but also the amount of buffering needed in the switches.

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 delay and fairness properties, Generalized-Processor-Sharing (GPS) is an ideal scheduling discipline [1]. 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 [2], 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 Weighted Fair Queueing (WFQ). 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 WFQ with a simple timestamp computation algorithm, but the price paid is in terms of fairness. A backlogged session in the VirtualClock server can be starved for an arbitrary period of time as a result of excess bandwidth it received from the server when other sessions were idle [1].

A scheduling algorithm that combines the delay and burstiness behavior of Weighted Fair Queueing, simple timestamp computations, and bounded unfairness, has so far remained elusive. The objective of our work is to develop an analytical framework for the design of such algorithms, systematically analyze its properties, and present scheduling algorithms based on this framework that have simple and efficient implementations.

Thus, in this paper we present a broad class of schedulers, that we call *Rate-Proportional Servers* (RPS). Schedulers in the RPS class offer the same end-to-end delay and burstiness bounds as WFQ. Since the class of rate-proportional servers is based on a general definition, multiple algorithms with the same properties but with different implementation complexities may be designed. Depending on their design, schedulers in the RPS class may have substantially different fairness properties. It is shown that both GPS, an algorithm with ideal fairness, and a fluid-model equivalent of VirtualClock, an unfair algorithm, are members of the RPS class.

This work is completed in the sequel to this paper [14], where two novel traffic scheduling algorithms in the RPS class, called *Frame-based fair queueing* (FFQ) and *Starting Potential-based Fair Queueing* (SPFQ) are defined and analyzed [14]. Both algorithms require only $O(1)$ time for the timestamp calculation, independent of the number of sessions sharing the server, and provide bounded unfairness.

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], which provides us the necessary tools for analysis of the RPS framework. In Section 3, we define the class of rate-proportional servers and derive bounds on the end-to-end delay and burstiness in a network of rate-proportional servers. In Section 4 we analyze their fairness and derive bounds on the unfairness of both the fluid-model and the packet-by-packet versions of a Rate-Proportional Server. Finally, we conclude the paper in Section 5 with a discussion of how the RPS framework is useful in the design of practical fair-queueing algorithms.

2 Preliminaries

2.1 Definitions and Notations

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

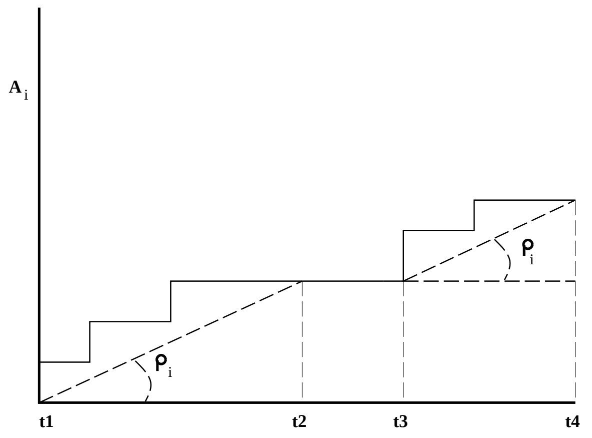

Figure 2.1: Intervals $(t_1, t_2]$ and $(t_3, t_4]$ are two different busy periods.

functions of t. However, in the packet-by-packet model, we assume that $A_i(\tau, t)$ increases only when the last bit of a packet is received by the server; likewise, $W_i(\tau, t)$ is increased only when the last bit of the packet in service leaves the server. Thus, the fluid model may be viewed as a special case of the packet-by-packet model with infinitesimally small packets.

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

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 remain continuously backlogged (Figure 2.1). Multiple session-i busy periods may appear during a system busy period. It is important to realize the basic distinction between a session backlogged period and a session busy period. The latter is defined only in terms of the arrival function and the allocated rate. Thus, the busy period serves as an invariant for evaluating the worst-case behavior of different scheduling algorithms under the same arrival pattern. For a more detailed explanation of the busy period, the reader is referred to [13].

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 $W^{\mathcal{S}}_{i,j}(\tau, t)$ the total service provided to the packets of the connection that arrived after time τ and until time t by server $\mathcal{S}$.

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,*

$$W^{\mathcal{S}}_{i,j}(\tau, t) \geq \max(0, \rho_i(t - \tau - \Theta^{\mathcal{S}}_i)).$$

$\Theta^{\mathcal{S}}_i$ *is the minimum non-negative number that can satisfy the above inequality.*

Server	Latency	Fairness	Complexity
GPS	0	0	-
Weighted Fair Queueing	$\frac{L_i}{\rho_i} + \frac{L_{max}}{r}$	$\max(\max(C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}, C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i})$, where $C_i = \min((V-1)\frac{L_{max}}{\rho_i}, \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}))$.	$O(V)$
Self-Clocked Fair Queueing	$\frac{L_i}{\rho_i} + \frac{L_{max}}{r}(V-1)$	$\frac{L_i}{\rho_i} + \frac{L_j}{\rho_j}$	$O(\log V)$
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 WFQ 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.

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 at the beginning of a session busy period. 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.

The following upper bounds on the behavior of a LR-server were shown in [13] when the arrivals or session i are shaped by a leaky bucket with parameters (σ_i, ρ_i).

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.

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, Weighted Fair Queueing, Weighted Round Robin, Self-Clocked Fair Queueing, VirtualClock and Deficit-Round-Robin — have been shown to be $\mathcal{LR}$-servers [13]. In Table 2.1 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 (WFQ) has the lowest latency among all the packet-by-packet servers; thus, from Theorem 1, WFQ has the lowest bounds on end-to-end delay and buffer requirements. However, WFQ also has the highest implementation complexity. VirtualClock has the same latency as WFQ, but is not a fair algorithm [3, 1]. Notice, however, that none of the other algorithms suffers from such a high level of unfairness. In Self-Clocked Fair Queueing 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.

2.2 Potential Functions

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 normalized service provided to it by the scheduler at that instant. Similarly, we can define a global virtual-time function that increases at the rate of the total service performed by the scheduler at each instant during a server-busy period. In a GPS scheduler, the virtual times of all backlogged connections are identical at every instant, and 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, WFQ, 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. If the potential of an idle connection is increased by the normalized 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.

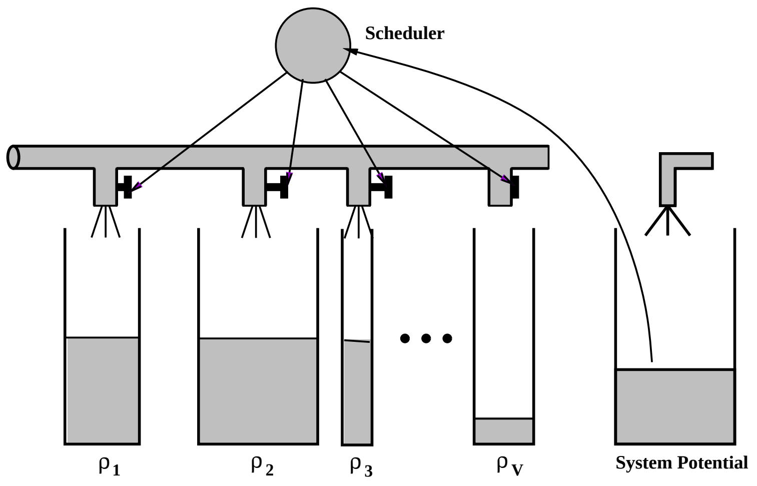

Figure 2.2: Illustration of the concept of potentials.

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.

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

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

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

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

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

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

The concept of potentials can be illustrated by the following analogy in Figure 2.2. Each connection is represented by a jar in the figure, with the level of fluid in the jar representing the total normalized service received by the connection during the current system-busy period. Thus, our definition of connection potential corresponds to the level of fluid in the jar representing the connection. The objective of a fair scheduler is then to add fluid to the jars corresponding to backlogged connections such that their fluid levels stay close to each other. A GPS scheduler meets this objective perfectly by enabling the fluid levels to rise exactly at the same rate.

From Figure 2.2, it is easy to explain the intuition behind the definition of the system potential function. The system potential can be likened to the level of fluid in a separate jar that keeps track of the global state of the system. This level is used as the reference to set the level of fluid in the jar of a connection when it becomes backlogged after an idle period. The actual system-potential function used determines how this reference level is determined. For example, in the case of the VirtualClock server, the jar representing the global state is being filled at a constant rate during a system-busy period, regardless of the distribution of packets transmitted by each connection.

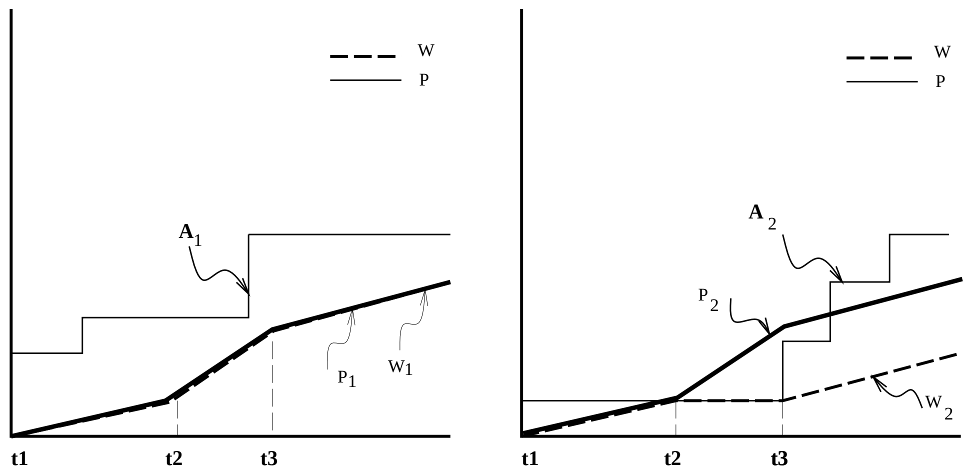

Figure 2.3: Evolution of the potential and offered service for two connections in the GPS multiplexer.

The concept of potentials is further illustrated with respect to a GPS scheduler in Figure 2.3. 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 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 2.4. 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 2.4, 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

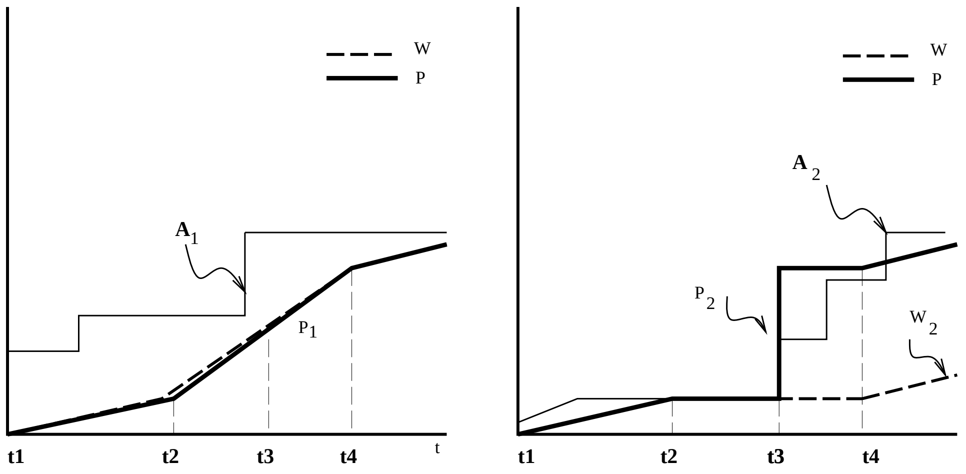

Figure 2.4: Evolution of the potential and offered service for two connections in the SCFQ multiplexer.

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

Having described the concept of potential, we now use it 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 backlogged connection, at every instant the server is busy. This ensures that a newly backlogged connection acquires a starting potential not higher than 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* connection potential *$P_i(t)$ 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 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. In a following paper [14], we will demonstrate two specific system-potential functions that lead to practical scheduling algorithms.

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,

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

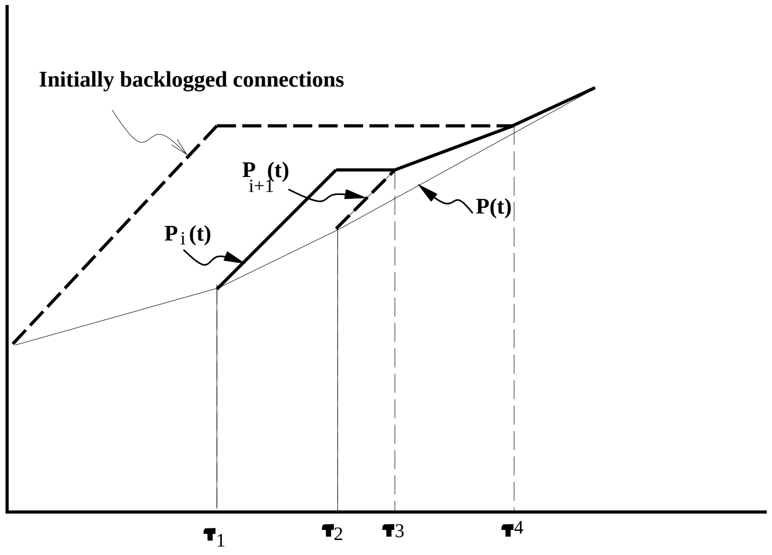

Figure 3.1: An example illustrating the evolution of potential functions in a rate-proportional server.

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

$$P(t) = t.$$

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

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

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

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

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 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.*

Proof: 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

$$P_i(t) \geq P(t). \tag{3.4}$$

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 $(\tau_0, \tau]$,

$$P_i(t) - P_i(\tau_0) = \frac{W_i(\tau_0, t)}{\rho_i}. \tag{3.5}$$

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

$$P_i(\tau_0) = P(\tau_0). \tag{3.6}$$

From equations (3.4) and (3.6),

$$\begin{aligned} P_i(t) - P_i(\tau_0) &\geq P(t) - P(\tau_0) \\ &\geq (t - \tau_0). \end{aligned} \tag{3.7}$$

Therefore, from equations (3.5) and (3.7),

$$W_i(\tau_0, t) \geq \rho_i(\tau - \tau_0). \tag{3.8}$$

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.

□

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*

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

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

$$P_i(t) \geq P(t), \tag{3.9}$$

and

$$P_i(\tau) = P(\tau). \tag{3.10}$$

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

$$P(t) - P(\tau) \geq (t - \tau). \tag{3.11}$$

From equations (3.9), (3.10), and (3.11) we can easily conclude that

$$P_i(t) - P_i(\tau) \geq (t - \tau). \tag{3.12}$$

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

$$\begin{aligned} P_i(t) - P_i(\tau) &= \frac{W_i(\tau, t)}{\rho_i} \\ &\geq t - \tau. \end{aligned} \tag{3.13}$$

From equations (3.12) and (3.13),

$$W_i(\tau, t) \geq \rho_i(t - \tau) \tag{3.14}$$

□

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.

Proof: 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

$$W_i(\tau, t^*) < \rho_i(t^* - \tau). \tag{3.15}$$

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 $t^* > t_a$,

$$W_i(\tau, t_a) \geq \rho_i(t_a - \tau). \tag{3.16}$$

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

$$W_i(t_a, t^*) \geq \rho_i(t^* - t_a) \tag{3.17}$$

From equations (3.16) and (3.17) we can conclude that

$$W_i(\tau, t^*) \geq \rho_i(t^* - \tau). \tag{3.18}$$

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

$$W_i(\tau, t^* - \Delta t) \geq \rho_i(t^* - \Delta t - \tau). \tag{3.19}$$

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

$$W_i(\tau, t^* - \Delta t) = A_i(\tau, t^* - \Delta t). \tag{3.20}$$

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

$$W_i(\tau, t^* - \Delta t) = W_i(\tau, t^*). \tag{3.21}$$

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,

$$A_i(\tau, t^* - \Delta t) = A_i(\tau, t^*). \tag{3.22}$$

From equations (3.15), (3.20), (3.21), and (3.22) we can conclude that

$$A_i(\tau, t^*) < \rho_i(t^* - \tau). \tag{3.23}$$

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

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 [1], we can prove the following lemma:

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

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

Proof: The proof is very similar to that of a GPS server [1] and is given in Appendix A.

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 $\hat{W}_i(t_1, t_2)$ the service offered to connection i during the interval (t_1, t_2) when this partial service is included. At the instant when the kth packet starts service in PRPS,

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

Thus, we can state the following corollary:

Corollary 1: *At any time t,*

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

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

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

Proof: 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 in the fluid server at time t. In this case $P_j(t) \leq P_i(t)$ for every 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. 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. Thus we can write

$$\hat{W}_k^F(\tau, t) \leq \hat{W}_k^P(\tau, t) \quad \forall k \in S. \tag{3.24}$$

Furthermore, the current session i was not backlogged in the fluid-server just before time τ; otherwise, a connection from the set S would not have been serviced just before τ. Therefore,

$$\hat{W}_i^F(0, \tau) \geq \hat{W}_i^P(0, \tau) \tag{3.25}$$

But we also know, that at time t connection i has received more service in the packet-by-packet server than in the fluid server. Thus,

$$\hat{W}_i^F(0, t) < \hat{W}_i^P(0, t) \tag{3.26}$$

Subtracting eq.(3.25) from eq.(3.26),

$$\hat{W}_i^F(\tau, t) < \hat{W}_i^P(\tau, t) \tag{3.27}$$

That is, during the interval $(\tau, t]$ connection i received more service in the packet-by-packet server than in the fluid server. Similarly, during the interval $(\tau, t]$ all connections in set S received more or equal service in the packet-by-packet server than in the fluid server. Since both servers are work conserving, there must exist at least one connection j that during the same interval $(\tau, t]$ received more service in the fluid server than in the packet-by-packet server. That is,

$$\hat{W}_j^F(\tau, t) > \hat{W}_j^P(\tau, t) \tag{3.28}$$

Notice also that this connection does not belong in the set S and therefore it can only have potential $P_j(t) \leq P_i(t)$. Notice also, that connection j became backlogged in the fluid server at or after time τ. Therefore,

$$\hat{W}_j^F(0, \tau) \geq \hat{W}_j^P(0, \tau) \tag{3.29}$$

By adding eq.(3.28) and eq.(3.29),

$$\hat{W}_j^F(0, t) > \hat{W}_j^P(0, t) \tag{3.30}$$

□

We will now use the above lemma and a method similar to the one presented in [16] for the WFQ 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. We will provide some intuition on these bounds and refer the reader to Appendix A for a formal proof. 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. The complete proof can be found in Appendix A.

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

$$\begin{aligned} W_{i,j}^P(\tau, t) &\geq W_{i,j}^F(\tau, t - \frac{L_{max}}{r}) - L_i \\ &\geq \max(0, \rho_i(t - \tau - \frac{L_{max}}{r}) - L_i), \quad \text{by Theorem 2} \\ &\geq \max(0, \rho_i(t - \tau - \frac{L_{max}}{r} - \frac{L_i}{\rho_i})) \end{aligned} \tag{3.31}$$

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 WFQ. Thus, any packet-by-packet rate-proportional server has the same upper bound on end-to-end delay and buffer requirements as those of WFQ 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.

4 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 normalized 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,

$$\left| \frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j} \right| \leq \mathcal{FR}, \tag{4.1}$$

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: *Let i and j be two connections that became greedy at time τ in a packet-by-packet rate-proportional server. The following bound holds for every time interval $(t_1, t_2]$ after time τ.*

$$\left|\frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i}\right| \leq \max(\Delta P + C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}, \Delta P + C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}) \tag{4.2}$$

Appendix A contains a proof of this theorem. Since Weighted Fair Queueing is a packet-by-packet rate proportional server with $\Delta P = 0$, we obtain the following result on the fairness of a WFQ scheduler by setting $\Delta P = 0$ in Eq. (4.2).

Corollary 3: *For a WFQ scheduler,*

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

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

5 Conclusions

In this paper we developed the framework of rate-proportional servers (RPS) for designing schedulers with low latency and bounded unfairness. Fundamental to the definition of rate-proportional servers is the system-potential function that maintains the global state of the system by tracking the service offered by the system to all connections sharing the outgoing link. Similarly, the state of each connection is represented by a connection potential function. In a packet-by-packet server, the system potential and connection potentials provide the basis for computing a timestamp for each arriving packet. Packets are then transmitted in increasing order of their timestamps. We defined the necessary properties the system potential function must satisfy for the server to provide a delay bound, burstiness and buffer requirements identical to that of Weighted Fair Queueing.

Besides providing valuable insight into the behavior of scheduling algorithms, the RPS model is useful in the design of practical scheduling algorithms. This fact is illustrated in the sequel to this paper [14], where we present two practical algorithms belonging to the RPS class, with application in both general packet networks and in ATM networks. Note that the fundamental difficulty in designing a practical rate-proportional server is the need to maintain the system potential function. Tracking the global state of the system precisely requires simulating the corresponding fluid-model RPS in parallel with the packet-by-packet system. The algorithms in [14], however, avoid this need by maintaining the system potential only as an approximation of the actual global state in the fluid model, and re-calibrating the system potential periodically to correct any discrepancies. In the Frame-based Fair Queueing algorithm this re-calibration is done at frame boundaries, while in Starting Potential-based Fair Queueing (SPFQ) the re-calibration occurs at packet boundaries. This gives rise to two algorithms with the same delay bound, but with slightly different fairness properties. Both algorithms, however, provided bounded unfairness and $O(1)$ timestamp computations.

It is hoped that the RPS framework will lead to the development of other algorithms in the future. Further work will include the analysis of rate-proportional servers under probabilistic input traffic models, such as the *exponentially-bounded-burstiness model* [17].

References

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

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

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

[4] 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.

[5] 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.

[6] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. ACM SIGCOMM '95*, pp. 231–242, September 1995.

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

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

[9] D. Verma, D. Ferrari, and H. Zhang, "Guaranteeing delay jitter bounds in packet switching networks," in *Proc. Tricomm 91*, pp. 35–43, April 1991.

[10] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. 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 *Proc. IEEE INFOCOM '94*, pp. 636–646, April 1994.

[13] D. Stiliadis and A. Varma, "Latency-Rate servers: A general model for analysis of traffic scheduling algorithms," in *Proc. IEEE INFOCOM '96*, pp. 111–119, March 1996.

[14] D. Stiliadis and A. Varma, "Efficient fair-queueing algorithms for ATM and packet networks," submitted to *IEEE/ACM Transactions on Networking*, April 1996.

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

[16] J. Rexford, A. Greenberg, and F. Bonomi, "A fair leaky-bucket shaper for ATM networks." unpublished report, AT&T Bell Laboratories, 1995.

[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.

Appendix A

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

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

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

$$t_k^P = \frac{1}{r} \sum_{j=1}^{k} L_j. \tag{A.2}$$

From (A.1) and (A.2),

$$t_k^P \leq t_k^F. \tag{A.3}$$

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 $m+1, m+2, \ldots, k$ 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

$$\begin{aligned} t_k^F - (t_m^P - \frac{L_m}{r}) &\geq \frac{1}{r} \sum_{j=m+1}^{k} L_j, \\ \text{or,} \qquad t_k^F &\geq t_m^P + \frac{1}{r} \sum_{j=m+1}^{k} L_j - \frac{L_m}{r}. \end{aligned} \tag{A.4}$$

But,

$$t_k^P = t_m^P + \frac{1}{r} \sum_{m+1}^{k} L_j. \tag{A.5}$$

From (A.4) and (A.5), and noting that $L_m \leq L_{max}$,

$$t_k^P \leq t_k^F + \frac{L_{max}}{r}. \tag{A.6}$$

Thus, combining (A.3) and (A.6), we get the upper bound as

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

□

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

$$\begin{aligned} \hat{W}_i^P(0,t_i^k) &= \hat{W}_i^P(0,\tau_i^k) + L_i^k \\ &\leq \hat{W}_i^F(0,\tau_i^k) + L_i^k \\ &\leq \hat{W}_i^F(0,t_i^k) + L_i^k \\ &\leq \hat{W}_i^F(0,t_i^k) + \rho_i \max_{1\leq n\leq V}(\frac{L_n}{\rho_n}). \end{aligned} \tag{A.7}$$

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. Then, at t_i^k, session i will have to receive an additional amount of service at maximum $\rho_i(F_i^k - P(t_i^k))$ in the fluid server to catch up with the potential in the packet-by-packet server. Thus,

$$\begin{aligned} \hat{W}_i^P(0,t_i^k) - \hat{W}_i^F(0,t_i^k) &\leq \rho_i(F_i^k - P_i(t_i^k)) \\ &\leq \rho_i(F_i^k - P_i(\tau_i^k)), \quad \text{since } P_i(t_i^k) \geq P_i(\tau_i^k). \end{aligned} \tag{A.8}$$

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, with potential $P_j(\tau_i^k) \leq P_i(\tau_i^k)$ that has received more service in the fluid server. Let the last packet serviced from 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

$$P_i(\tau_i^k) \geq P_j(\tau_i^k) \geq S_j^m, \tag{A.9}$$

and

$$F_i^k \leq F_j^m. \tag{A.10}$$

Substituting for F_i^k and $P_i(t_i^k)$ in Eq. (A.8) from equations (A.10) and (A.9), respectively,

$$\begin{aligned} \hat{W}_i^P(0,t_i^k) - \hat{W}_i^F(0,t_i^k) &\leq \rho_i(F_j^m - S_j^m) \\ &\leq \rho_i\frac{L_j}{\rho_j} \\ &\leq \rho_i \max_{1\leq n\leq V}(\frac{L_n}{\rho_n}). \end{aligned} \tag{A.11}$$

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

$$P_i(t_1) \geq P(t_1). \tag{A.12}$$

We also know that

$$P_j(t_1) \leq P(t_1) + \Delta P. \tag{A.13}$$

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,

$$P_i(t_2) - P_i(t_1) = \frac{W_i(t_1, t_2)}{\rho_i}, \tag{A.14}$$

and

$$P_j(t_2) - P_j(t_1) = \frac{W_j(t_1, t_2)}{\rho_j}. \tag{A.15}$$

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,

$$x = P_j(t_2) - P_i(t_2) \geq 0. \tag{A.16}$$

From equations (A.12),(A.14), and (A.16),

$$\frac{W_i(t_1, t_2)}{\rho_i} \leq P_j(t_2) - P(t_1) - x. \tag{A.17}$$

Similarly, from equations (A.13) and (A.15),

$$\frac{W_j(t_1, t_2)}{\rho_j} \geq P_j(t_2) - P(t_1) - \Delta P. \tag{A.18}$$

From equations (A.17) and (A.18) we can easily conclude that

$$\frac{W_i(t_1, t_2)}{\rho_i} - \frac{W_j(t_1, t_2)}{\rho_j} \leq \Delta P - x \leq \Delta P. \tag{A.19}$$

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

$$\frac{W_j(t_1, t_2)}{\rho_j} - \frac{W_i(t_1, t_2)}{\rho_i} \leq \Delta P. \tag{A.20}$$

Therefore,

$$\left|\frac{W_i(t_1, t_2)}{\rho_i} - \frac{W_j(t_1, t_2)}{\rho_j}\right| \leq \Delta P. \tag{A.21}$$

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 that after time τ both connections i, j have an infinite supply of packets. Without loss of generality, let

$$\frac{\hat{W}_i(t_1, t_2)}{\rho_i} \geq \frac{\hat{W}_j(t_1, t_2)}{\rho_j}, \tag{A.22}$$

for a time interval $(t_1, t_2]$ with $\tau \leq t_1 < t_2$.

In order to provide tight bounds on the fairness of the algorithm we need to define the potential function of the packet-by-packet server. Thus, let us denote with $a_i(t)$ the potential of connection i at time t on the packet-by-packet server. Note that the function $a_i(t)$ is defined in terms of the potential of the connection in the fluid server. That is, a connection misses the same amount of service in both the fluid server and the packet server while it is absent. The only difference is that, when the connection becomes backlogged, its potential in the packet is increased only when a packet of that connection is transmitted. Let $P_i(t)$ represent the potential of connection i in the fluid server as usual.

We know that, after time τ both connections have an infinite supply of packets. Thus, the potential of both connections in the fluid server is only increased by the normalized service offered to them. For the service offered to connection i in the interval $(t_1, t_2]$ we can write:

$$\frac{\hat{W}_i(t_1,t_2)}{\rho_i} \leq \max(a_i(t_2) - P_i(t_1)) + \frac{L_{max}}{\rho_i} \tag{A.23}$$

If the potential of connection i at time t_2 is greater than $P_i(t_1)$, then the normalized service offered to connection i during the interval $(t_1, t_2]$ is equal to the increase in potential after time t_1 plus the amount of service that connection i received more in the fluid-server compared to the packet-by-packet server until time t_1. If on the other hand, the potential of connection i in the packet-by-packet server is less than $P_i(t_1)$, then the packets that were serviced after time t_1 in the fluid-server have not yet been serviced in the packet-by-packet server. Thus, the only service offered from the packet-by-packet server to connection i, has already been offered to the fluid-server before time t_1. This service is always bounded by L_{max}. Similarly for connection j we can write:

$$\frac{\hat{W}_j(t_1,t_2)}{\rho_j} \geq \max(a_j(t_2) - P_i(t_1)) - C_j \tag{A.24}$$

Notice that connection j may have received more service in the packet-by-packet server than the fluid-server until time t_1. But we also know, that the difference in potentials between connections i and j is bounded by ΔP at time t_1. That is,

$$|P_i(t_1) - P_j(t_1)| \leq \Delta P \tag{A.25}$$

Finally, since we assumed that connection i has received more normalized service, and from the definition of the packet-by-packet rate-proportional servers

$$a_i(t_2) \leq a_j(t_2) + \frac{L_j}{\rho_j} \tag{A.26}$$

Combining Equations (A.23),(A.24),(A.25), and (A.26),

$$\frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j} \leq \Delta P + C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j} \tag{A.27}$$

Similarly, if connection j received more normalized service in the interval $(t_1, t_2]$ we can write:

$$\frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i} \leq \Delta P + C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i} \tag{A.28}$$

From Equations (A.27),(A.28) we can conclude that

$$|\frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i}| \leq \max(\Delta P + C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}, \Delta P + C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}) \tag{A.29}$$

□