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

Dimitrios Stiliadis, Member, IEEE, and Anujan Varma, Member, IEEE

Abstract—Generalized processor sharing (GPS) has been considered as an ideal scheduling discipline based on its end-to-end delay bounds and fairness properties. Until recently, emulation of GPS in a packet server has been regarded as the ideal means of designing a packet-level scheduling algorithm to obtain low delay bounds and bounded unfairness. Strict emulation of GPS, as required in the weighted fair queueing (WFQ) scheduler, however, incurs a time-complexity of \( O(N) \) where \( N \) is the number of sessions sharing the link. Efforts in the past to simplify the implementation of WFQ, such as self-clocked fair queueing (SCFQ), have resulted in degrading its isolation properties, thus affecting the delay bound. In this paper we present a methodology for the design of scheduling algorithms that provide the same end-to-end delay bound as that of WFQ and bounded unfairness without the complexity of GPS emulation. The resulting class of algorithms, called rate-proportional servers (RPS's), are based on isolating scheduler properties that give rise to ideal delay and fairness behaviors. Network designers can use this methodology to construct efficient fair-queueing algorithms, balancing their fairness with implementation complexity. This work is completed in a sequel to this paper, where we present the detailed design and implementation of two novel scheduling algorithms based on the RPS framework.

Index Terms—Fair queueing algorithms, performance bounds, switch scheduling, traffic scheduling.

I. 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 asynchronous transfer mode (ATM) networks by the use of proper packet scheduling algorithms in the switches (or routers).

Several service disciplines are known in the literature for bandwidth allocation and transmission scheduling in output-buffered switches [1]–[9] (for a survey, see [10]). 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. 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 of the backlogged sessions 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 (WFQ) [2], is defined by simulating the fluid system and serving packets based on their transmission times under the fluid system. A maximum of \( N \) events may be triggered in the GPS simulator during the transmission of one packet. Thus, the time required for completing a scheduling decision is \( O(N) \). 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. A similar implementation was also proposed in [13]. This approach reduces the complexity of the algorithm greatly. However, the price paid is in terms of the delay bounds that grow linearly with the number of sessions that share the outgoing link [14]–[16]. Thus, the worst-case delay of a session can no longer be controlled just by controlling its reservation, as is possible in WFQ. The higher end-to-end delay also affects the burstiness of sessions within the network, increasing the buffer requirements. The VirtualClock scheduling algorithm [3] 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], [17].

A scheduling algorithm that combines the delay and burstiness behavior of WFQ, 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

1Since this work was first published, a number of algorithms with these properties have appeared in the literature, for example [18] and [19].
rate-proportional servers (RPS's). Schedulers in the RPS class offer the same end-to-end delay and burstiness bounds as WFQ under the rate-proportional service discipline [1]. Since the class of RPS's is based on a general definition, multiple algorithms may be designed with the same delay behavior but with different implementation complexities. 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 both members of the RPS class.

This work is completed in a sequel to this paper [20], 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. Both algorithms require only $O(1)$ time for the timestamp calculation, independent of the number of sessions sharing the server, and provide bounded unfairness.

II. PRELIMINARIES

A. Definitions and Notations

We assume a packet switch where a set of $N$ sessions share a common output link. We denote with $p_i$ the rate allocated to session $i$. We assume that the servers are non-cut-through devices. Let $A_i(t, t)$ denote the arrivals from session $i$ during the interval $(t, t]$ and $W_i(t, t)$ the amount of service received by session $i$ during the same interval. We will call the fraction $W_i(t, t)/p_i$ as the normalized service offered to session $i$. In a packet-by-packet model, we assume that $A_i(t, t)$ increases only when the last bit of a packet is received by the server; likewise, $W_i(t, t)$ is increased only when the last bit of the packet in service leaves the server.

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

In [16] we introduced a general model for analysis of traffic scheduling algorithms, called latency-rate (LR) servers. We will use the theory of LR servers as a tool in the delay analysis of RPS's in the next section. A brief overview of LR servers follows. For a more detailed treatment, the reader is referred to [16].

Definition 3: A session $i$ busy period is a maximal interval of time $(t_1, t_2]$ such that at any time $t \in (t_1, t_2]$, the accumulated arrivals of session $i$ since the beginning of the interval do not fall below the total service received during the interval at a rate exactly $p_i$. That is,

$$A_i(t, t) \geq p_i(t - t_1).$$

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 (Fig. 1).

A server in the LR class is characterized by two parameters: latency $\Theta_i$ and minimum allocated rate $p_i$. Let us assume that the $j$th busy period of session $i$ starts at time $\tau$. We denote by $W_i^S_j(\tau, t)$ the total service provided to the packets of the session that arrived after time $\tau$ and until time $t$ by server $S$.

Definition 4: Let $\tau$ be the starting time of the $j$th busy period of session $i$ in server $S$, and $\tau^*$ the time at which the last packet that arrived during the $j$th busy period leaves the server. Then, server $S$ is an LR server if and only if, at every instant $t$ in the interval $(\tau, \tau^*)$

$$W_i^S_j(\tau, t) \geq \max[0, p_i(t - \tau - \Theta_i)].$$

$\Theta_i$ is the minimum nonnegative number that satisfies the above inequality.

The right-hand side of the above equation defines a linear envelope to bound the minimum service offered to session $i$ during a busy period. The following upper bounds on the behavior of an LR server were shown in [16] when the arrivals or session $i$ are shaped by a leaky bucket with parameters $(\sigma_i, p_i)$.

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

$$D_i^K \leq \frac{Q_i}{p_i} + \sum_{j=1}^K \Theta_i(S_j)$$

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

where $\Theta_i(S_j)$ is the latency of the $j$th server on the path of the session.

In [16], we calculated the latencies of many well-known work-conserving schedulers, along with bounds on their fairness and implementation complexity. The fairness parameter chosen was the maximum difference in normalized service offered by the scheduler to two sessions over any interval during which both sessions are continuously backlogged. The implementation complexity is at least $O(\log N)$ for all sorted-priority schedulers. The packet-by-packet approximation of GPS, namely WFQ, has the lowest latency among all of the 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 [1], [3]. In SCFQ as well as all round-robin schedulers, latency is a function of the number of sessions that share the outgoing link. Therefore, the resulting end-to-end delay bounds may be much larger compared to WFQ. Our objective in this paper is to develop a methodology for the design of schedulers with the same delay bound as that of WFQ and bounded unfairness without the need for strict GPS emulation.

B. Potential Functions

The GPS scheduler provides ideal fairness by offering the same normalized service to all backlogged sessions 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. Following the approach of Golestani [12], we introduce such a function that represents the state of each session in a scheduler and call it potential. The potential of a session is a nondecreasing function of time during a system busy period. When session \( i \) is backlogged, its potential increases exactly by the normalized service it received. That is, if \( P_i(t) \) denotes the potential of session \( i \) at time \( t \), then, during any interval \( (\tau, t] \) within a backlogged period for session \( i \),

\[
P_i(t) - P_i(\tau) = \frac{W_i(\tau, t)}{p_i}.
\]

Note that the potentials of all sessions 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. At any time, the scheduler serves only the set of sessions with the minimum potential values. If multiple sessions have the same potential, then they are served at rates proportional to their reservations.

A fair algorithm is one that provides the same normalized service to all backlogged sessions within a small interval of time. Since the session potential increases at the same rate when they are equal, the basic objective of a fair scheduling algorithm is to equalize the potential of each backlogged session. Sorted-priority schedulers such as GPS, WFO, SCFO, and VirtualClock all attempt to achieve this objective. However, in our definition of potential, we did not specify how the potential of a session is updated when it becomes idle. Except that the potential is nondecreasing. Scheduling algorithms differ in the way they update the potentials of idle sessions. Ideally, during any interval in which session \( i \) is idle, its potential must increase by the normalized service it could have received if it were backlogged. Thus, if the potential of an idle session is increased by the normalized service it missed, it is easy to see that when the session becomes busy again its potential will be identical to that of other backlogged sessions in the system, allowing it to receive service immediately.

One way to update the potential of a session when it becomes backlogged is to define a system potential function that keeps track of the progress of the total work done by the scheduler. The system potential \( P(t) \) is a nondecreasing 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 nondecreasing function of the potentials of the individual sessions before time \( t \), and the real time \( t \).

\[
P(t) = \mathcal{F}(P_1(t-), P_2(t-), \ldots, P_N(t-), t).
\]

For example, the GPS server initializes the potential of a newly backlogged session to that of a session 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 sessions at time \( t \). The VirtualClock scheduler, on the other hand, initializes the potential of a session 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 behaviors of the scheduler.

The utility of the system potential function \( P(t) \) is in estimating the amount of service missed by a session while it was idle. In an ideal server like GPS the system potential is always equal to the potential of the sessions that are currently backlogged and are thus receiving service. However, this approach requires that all sessions can receive service at the same time. In a packet scheduler we need to relax this constraint since only one session can be serviced at a time. In addition, the algorithm will have different properties depending on the potential value that a session receives when it becomes backlogged after an idle period. If the potential of a newly backlogged session is estimated higher than the potential of the sessions currently being serviced, the former may have to wait for one or more packets to be transmitted from each of the other sessions before it can be serviced. This results in a latency that is proportional to the number of backlogged sessions. Since the potential of a newly backlogged session is set to the system potential, achieving zero latency in a fluid server requires that the system potential not be allowed to exceed the potential of backlogged sessions. This observation is the basis of the RPS framework introduced in the next section.

III. RATE-PROPORTIONAL SERVERS

A. Fluid RPS's

Having described the concept of potentials, we now use it to define a general class of schedulers, which we call RPS's. We will first define these schedules based on the fluid model and later extend the definition to the packet version. We denote the set of backlogged sessions at time \( t \) by \( B(t) \).
**Definition 5:** An RPS is a work-conserving server with the following properties.

1) Rate $\rho_i$ is allocated to session $i$ and

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

where $r$ is the total service rate of the server.

2) A session potential $P_i(t)$ is associated with each session $i$ in the system, describing the state of the session at time $t$. This function must satisfy the following properties.

   a) When a session is not backlogged, its potential remains constant.

   b) If a session becomes backlogged at time $\tau$, then

   $$P_i(\tau) = \max[P_i(\tau^+), P_i(\tau^-)].$$

   c) At each instant $t > \tau$ that the session remains backlogged, the potential function of the session is increased by the normalized serviced offered to that session during the interval $(\tau, t]$. That is

   $$P_i(t) = P_i(\tau) + \frac{W_i(\tau, t)}{\rho_i}.$$  \hfill (3.1)

3) The system potential function $P(t)$ describes the state of the system at time $t$. Two main conditions must be satisfied for the function $P(t)$.

   a) For any interval $(t_1, t_2]$ during a system busy period

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

   b) The system potential is always less than or equal to the potential of all backlogged sessions at any time $t$. That is

   $$P(t) \leq \min_{j \in \mathcal{S}(t)} \{P_j(t)\}.  \hfill (3.3)$$

4) Sessions are serviced at each instant $t$ according to their instantaneous potentials as per the following rules.

   a) Among the backlogged sessions, only the set of sessions with the minimum potential at time $t$ is serviced.

   b) Each session in this set is serviced with an instantaneous rate proportional to its reservation, so as to increase the potentials of the sessions 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 follow-on paper [20], we demonstrate two specific system potential functions that lead to practical scheduling algorithms.

**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 system potential. Any other period will be considered an inactive period for session $i$.

The concept of active period is useful in analyzing the behavior of an RPS. When a session is in an inactive period, it can not be backlogged and, therefore, can not be receiving service. On the other hand, an active period need not be the same as a backlogged period for the session. Since in an RPS the potential of a session can be below the system potential only when the session is idle, a transition from inactive to active state can occur only by the arrival of a packet of a session that is currently idle, whose potential is below the system potential. A session in an active period may not receive service throughout the active period since an RPS services only sessions 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.

The evolution of potential functions in an RPS is illustrated in Fig. 2 with an example. Assume that the system potential is always maintained below the potential of every backlogged session. At time $\tau_1$ session $i$ becomes active and receives exclusive service, trying to achieve the same potential as the rest of the sessions. At time $\tau_2$ a second session $i+1$ becomes active and the service of $i$ is temporarily suspended. The
potentials of the two new sessions become equal at \( \tau_3 \); during the interval \((\tau_3, \tau_4]\) each of them receives service proportional to its reservation so that their potentials remain equal. That is

\[
W_i(\tau_3, \tau_4) = \frac{W_{i+1}(\tau_3, \tau_4)}{\rho_{i+1}}.
\]

At \( \tau_1 \) the potentials of \( i \) and \( i+1 \) become equal to that of other sessions already backlogged in the system; therefore, from \( \tau_1 \) all backlogged sessions in the system receive service proportional to their allocated rates. If another new session becomes active after time \( \tau_4 \), service to all of the sessions will be suspended until the new session reaches the same potential. In addition, if a session finishes service, the instantaneous service rates of other backlogged sessions will increase because of the work-conserving nature of the scheduler. However, a session may be temporarily suspended if it has received more than its allocated bandwidth earlier during the same active period.

Lemma 1: Let \( \tau \) be the time at which a session \( i \) becomes active in an RPS. Then, at any time \( t > \tau \) that belongs in the same active period, the service offered to session \( i \) is

\[
W_i(\tau, t) \geq \rho_i(t - \tau).
\]

Proof: Intuitively, this result asserts that the service of a backlogged session 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 session active period. By the definition of active period

\[
P_i(t) \geq P(t)
\]

and

\[
P_\tau(\tau) = P(\tau).
\]

From (3.4), (3.5), and the definition of RPS we can easily conclude that

\[
P_i(t) - P_\tau(\tau) \geq (t - \tau).
\]

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

\[
P_i(t) - P_\tau(\tau) = \frac{W_i(\tau, t)}{\rho_i} \geq t - \tau.
\]

From (3.6) and (3.7), \( W_i(\tau, t) \geq \rho_i(t - \tau) \).

Since \( LR \) servers are defined in terms of busy periods, it is necessary to establish the correspondence between busy periods and active periods in an RPS. We will now show that the beginning of a busy period is the beginning of an active period as well.

Lemma 2: If \( \tau \) is the beginning of a session \( i \) busy period in an RPS, then \( \tau \) is also the beginning of an active period for session \( i \).

Proof: We will prove the lemma by contradiction. Assume, if possible, that time \( \tau \) is not the beginning of an active period. We have two cases.

Case 1: Time \( \tau \) belongs within an inactive period. Since session \( i \) was not busy before time \( \tau \) and becomes busy at \( \tau \), a packet must have arrived. Then the potential of the session would have become equal to the system potential and, thus, \( \tau \) is the beginning of an active period.

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

\[
P_i(t) \geq P(t).
\]

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

\[
P_i(t) - P_i(\tau_0) = \frac{W_i(\tau_0, t)}{\rho_i}.
\]

Since \( \tau_0 \) is the beginning of an active period, from Lemma 2

\[
W_i(\tau_0, t) \geq \rho_i(t - \tau_0).
\]

Before time \( \tau_0 \) the system was not backlogged. Therefore, we can write

\[
A_i(\tau_0, t) = W_i(\tau_0, t) \geq \rho_i(t - \tau_0).
\]

Thus, time \( \tau_0 \) belongs within the same busy period that covers the interval \((\tau_0, \tau]\). Therefore, \( \tau \) cannot be the beginning of a busy period.

A session busy period may actually consist of multiple session active periods. In order to prove that an RPS is an \( LR \) server with zero latency, we need to prove that at any time \( t \) during the \( j \)th busy period

\[
W_i, j(\tau, t) \geq \rho_i(t - \tau)
\]

where \( \tau \) marks the beginning of the \( j \)th busy period.

Using Lemmas 1 and 2 we can prove the following key result on the latency of an RPS.

Theorem 2: An RPS is an \( LR \) server with zero latency.

Proof: Let us again trace the evolution of the potential function associated with session \( i \). Consider the \( j \)th busy period of session \( i \) that started at time \( \tau \). We can split the busy period into intervals during which the session is in active or inactive states. During an inactive period the session is not receiving any service and no packets from the session are backlogged in the system.

We will prove the theorem by contradiction. Let us denote with \( t^* \) the first instant after \( \tau \) such that

\[
W_i(\tau, t^*) < \rho_i(t^* - \tau).
\]

We will show that \( t^* \) cannot be within the \( j \)th busy period of session \( i \).

Assume, if possible, that \( t^* \) is within the \( j \)th busy period of session \( i \). We distinguish two cases.

Case 1: Time \( t^* \) belongs within an active period. Let us denote with \( t_a \) the time that this active period started. From Lemma 2, \( t_a \geq \tau \). Then, since \( t^* > t_a \)

\[
W_i(\tau, t_a) \geq \rho_i(t_a - \tau).
\]

From Lemma 1, we also know that for time \( t^* \) that belongs in the same active period

\[
W_i(t_a, t^*) \geq \rho_i(t^* - t_a).
\]
From (3.12) and (3.13) we can conclude that
\[ W_i(\tau, t^*) \geq \rho_i(t^* - \tau). \]  
(3.14)

This contradicts with (3.11) and, hence, \( t^* \) cannot be within the \( j \)th busy period of session \( i \).

Case 2: Time \( t^* \) is within an inactive period. Consider time \( t^* - \Delta t \), where \( \Delta t \) is a small interval of time such that \( t^* - \Delta t \) also belongs in the same inactive period. Since, by hypothesis, \( t^* - \Delta t \) belongs to the \( j \)th busy period of session \( i \), we must have
\[ W_i(\tau, t^* - \Delta t) \geq \rho_i(t^* - \Delta t - \tau). \]  
(3.15)

Since the session is in an inactive period at time \( t^* - \Delta t \), there are no packets backlogged from that session. Therefore
\[ W_i(\tau, t^* - \Delta t) = A_i(\tau, t^* - \Delta t). \]  
(3.16)

In addition, no packets were serviced from the session during the interval \((t^* - \Delta t, t^*)\). That is
\[ W_i(\tau, t^* - \Delta t) = W_i(\tau, t^*). \]  
(3.17)

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 session would have entered an active period. Thus
\[ A_i(\tau, t^* - \Delta t) = A_i(\tau, t^*). \]  
(3.18)

From (3.11) and (3.16)-(3.18) we can conclude that
\[ A_i(\tau, t^*) < \rho_i(t^* - \tau). \]  
(3.19)

This means that time \( t^* \) does not belong in the same busy period as \( t^* - \Delta t \), a contradiction.

The above result enables us to apply all of the results on \( L \) servers to the class of RPS’s. Since both GSP and a fluid version of VirtualClock can both be considered as RPS’s, it is not surprising that they share the same delay bound. Thus, the definition of RPS’s provides us a methodology to design scheduling algorithms with zero latency.

B. Packet-by-Packet RPS

In the previous section we defined the RPS’s using a fluid model, where packets from different sessions can be served at the same time with different rates. However, in a real system only one session can be serviced at each time and, in addition, packets can not be split in smaller units. A packet-by-packet RPS can be defined in terms of the corresponding fluid system as follows. Let us assume that when the \( k \)th packet from session \( i \) finishes service in the fluid server, the potential of session \( i \) is \( T_j^i \). We can use this finishing potential to timestamp packets and schedule them in increasing order of their timestamps. We call such a server a packet-by-packet RPS (PRPS).

In the following discussion we denote the maximum packet size of session \( i \) by \( L_i \) and the maximum packet size among all of the sessions by \( L_{\text{max}} \). In order to analyze the performance of a PRPS we will bound the difference of service offered between the packet server and the fluid server when the same pattern of arrivals is applied to both of the servers. If we include the partial service received by packets in transmission, the maximum lag in service for a session \( i \) in the packet server occurs at the instant when a packet starts service. Let us denote with \( \bar{W}_i^F(t_1, t_2) \) the service offered to session \( i \) during the interval \([t_1, t_2]\) by the packet server when this partial service is included. Let \( \bar{W}_i^F(t_1, t_2) \) denote the service offered to session \( i \) by the fluid server during the interval \([t_1, t_2]\). The following lemma shows that the service offered by the packet server to a session can never lag behind that of the fluid server by more than \( L_{\text{max}} \).

Lemma 3: At any time \( t \)
\[ \bar{W}_i^F(0, t) - \bar{W}_i^F(0, t) \leq L_{\text{max}}. \]

Proof: Since both servers are work conserving, their system busy periods are identical. Hence, it is sufficient to prove the lemma for a single system busy period. Assume that the busy period under consideration began at time zero. Let \( p_j \) be the \( j \)th packet transmitted by the packet server during the busy period, and \( t_j \) the time at which \( p_j \) completes transmission in the packet server. Let \( r_j \) denote the time when this packet \( p_j \) completes service in the fluid server. We use \( L_j \) to denote the length of the packet \( p_j \), and \( F_j \) the timestamp (finishing potential) of \( p_j \). Then, we have
\[ t_k = \sum_{j=1}^{k} L_j / r. \]  
(3.20)

Consider any sequence of packets \( p_1, p_2, \ldots, p_k \) transmitted by the packet server during the busy period. Let the last packet \( p_k \) in this sequence be from session \( i \). We will show that
\[ \bar{W}_i^F(t) - \bar{W}_i^F(t) \leq L_{\text{max}}. \]  
(3.21)

Since the maximum lag in service in the packet server occurs when a packet from the session begins service in the packet server, this is sufficient to prove the lemma.

We will distinguish two cases.

Case 1: \( F_j \leq F_k \) for \( 1 \leq j < k \). Let the packet \( p_k \) be the \( n \)th packet of session \( i \). We will prove the result by contradiction. Assume, if possible, that \( t < t_{k-1} \) is the first time at which the service offered in the fluid server to session \( i \) exceeds the service offered to session \( i \) in the packet server at time \( t_{k-1} \) by more than \( L_i \). That is
\[ \bar{W}_i^F(t) - \bar{W}_i^F(t) > L_i. \]  
(3.22)

where \( L_i \) is the maximum size of a session \( i \) packet. Then, the fluid server must have started service of the \((n+1)\)th packet of session \( i \) by time \( t \). Also, since the fluid server serves packets in increasing order of potential, session \( i \) belongs to the set of sessions with the minimum potential at time \( t \). Let us denote with \( p_m \) the packet that is being serviced by the packet server at time \( t \). Then this packet has a timestamp less than or equal to \( F_k \). This means that all packets \( p_1, p_2, \ldots, p_m \) have already arrived by time \( t \) and have been served by the fluid server. Thus, for the total service offered by the fluid server from the start of the busy period until \( t \), we can write
\[ \sum_{j=1}^{N} \bar{W}_j^F(0, t) > \sum_{j=1}^{m} L_j + L_i. \]  
(3.23)
Similarly, for the service offered by the packet server
\[ \sum_{j=1}^{N} \hat{W}_j^P(0, t) \leq \sum_{j=1}^{m} L_j. \]  
(3.24)

Since both servers are work conserving, the left-hand sides of (3.23) and (3.24) must be equal, giving \( L^i \leq 0 \). Thus, a time \( t \leq l_{k-1} \) cannot be found satisfying (3.22).

Case 2: Before time \( l_k \), the packet server has transmitted at least one packet with timestamp larger than that of packet \( p_l \). Let \( p_{m} \) be the latest packet transmitted with \( P_{m} > P_{k} \). Then, packets \( p_{m+1}, p_{m+2}, \ldots, p_{k} \) must have arrived after time \( l_{m-1} \) when the packet \( p_{m} \) started service in the packet server. In addition, session \( i \) had no packets queued in the packet server at time \( l_{m-1} \). Therefore, we must have
\[ \hat{W}_i^F(0, t_{m-1}) \geq \hat{W}_i^F(0, t_{m-1}). \]  
(3.25)

We will now show that at any time \( t \), \( t_{m-1} \leq t \leq t_{k-1} \)
\[ \hat{W}_i^F(t_{m-1}, t) \leq L_{\max}. \]  
(3.26)

We will show this by contradiction. Let \( t \) be the first instant at which \( \hat{W}_i^F(t_{m-1}, t) > L_{\max} \). Let packet \( p_n \) be the packet under service in the packet server at time \( t \). Then, \( l_{n-1} \leq t < l_n \). Packets \( p_{n+1}, p_{n+2}, \ldots, p_n \) have timestamps less than \( P_{k} \) and have already arrived in the system. Since the fluid server is serving session \( i \) at time \( t \), session \( i \) has the lowest potential at that time. However, from hypothesis, the potential of session \( i \) at time \( t \) can only be larger than \( P_{k} \). This means that the fluid server has already completed serving the packets \( p_{m+1}, p_{m+2}, \ldots, p_n \). Therefore, we can write
\[ \sum_{j=1}^{N} \hat{W}_j^F(t_{m-1}, t) \geq \sum_{j=m+1}^{n} L_j + \hat{W}_i^F(t_{m-1}, t) \]
\[ > \sum_{j=m+1}^{n} L_j + L_{\max}. \]  
(3.27)

On the other hand, for the packet server we can write
\[ \sum_{j=1}^{N} \hat{W}_j^P(t_{m-1}, t) \leq \sum_{j=m+1}^{n} L_j. \]  
(3.28)

Again, the left-hand sides of (3.27) and (3.28) must be equal, giving rise to \( L^m > L_{\max} \), a contradiction.

From (3.25) and (3.28), we arrive at
\[ \hat{W}_i^F(0, t_{k-1}) \leq \hat{W}_i^F(0, t_{k-1}) + L_{\max}. \]  
(3.29)

This concludes the proof of Lemma 3.

It should be noted that Lemma 3 holds in the service domain but may not hold in the time domain. That is, the departures of a packet from the fluid and packet servers may be apart by an arbitrary interval of time. This is weaker than that in WFQ, where the bound holds in both the service and time domains [1]. However, we will show in Section III-C that the weaker bound is adequate for achieving latency equal to that of WFQ.

2We must caution the reader here that [21, Lemma 3] incorrectly states that the service discrepancy bound holds in the time domain.

In order to complete we also have to bound the amount by which the service of a session in the packet server can be achieved of that in the fluid server. This bound will be used later in the fairness analysis of RPS's. Packets are serviced in PRPS in increasing order of their finishing potentials. If packets from multiple sessions have the same finishing potential, then one of them will be selected for transmission first by the 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 session receives in the fluid server. The latter, in turn, requires knowledge of the potentials of the other sessions sharing the same outgoing link. We will use the following lemma to derive such an upper bound.

Lemma 4: Let \( [0, t] \) be an interval within a system busy period in the fluid server. Let \( t \) be a session backlogged in the fluid server at time \( t \) such that \( i \) received more service in the 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 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 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 \( j \).

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 RPS's, these sessions are not receiving service at time \( t \) in the fluid server. Let \( \tau \) 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 sessions in the set \( S \) were serviced by the fluid server. Thus, we can write
\[ \hat{W}_k^F(\tau, t) \leq \hat{W}_k^F(\tau, t) \forall k \in S. \]  
(3.30)

Furthermore, the current session \( i \) was not backlogged in the fluid server just before time \( \tau \); otherwise, a session from the set \( S \) would not have been serviced just before \( \tau \). Therefore
\[ \hat{W}_i^F(0, \tau) \geq \hat{W}_i^F(0, \tau). \]  
(3.31)

But we also know that at time \( t \), session \( i \) has received more service in the packet server than in the fluid server. Thus
\[ \hat{W}_i^P(0, t) < \hat{W}_i^P(0, t). \]  
(3.32)

Subtracting (3.31) from (3.32), we arrive at
\[ \hat{W}_i^F(\tau, t) < \hat{W}_i^F(\tau, t). \]  
(3.33)

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

$$W_{Fj}^F(\tau, t) > W_{Fj}^F(\tau, t). \quad (3.34)$$

Since this session does not belong in the set $S$, it can only have potential $\tau_j(t) > \tau_j(t)$. Notice also that session $j$ became backlogged in the fluid server at or after time $\tau$. Therefore

$$W_{Fj}^F(0, \tau) \geq W_{Fj}^F(0, \tau). \quad (3.35)$$

By adding (3.34) and (3.35)

$$\frac{W_{Fj}^P(0, t)}{\rho_j} > \frac{W_{Fj}^P(0, t)}{\rho_j}. \quad (3.36)$$

We will now use the above lemma and a method similar to the one presented in [22] for the WFQ server to find an upper bound for the amount of service that a session may receive in PRPS as compared to that in the fluid server.

**Lemma 5:** At any time $t$

$$\frac{W_{Fj}^P(0, t) - W_{Fj}^F(0, t)}{\rho_j} \leq \min \left( (N-1)L_{max}, \rho_j \max_{1 \leq n \leq N} \left( \frac{L_n}{\rho_n} \right) \right).$$

**Proof Sketch:** We will provide some intuition on these bounds and refer the reader to [23] for a formal proof. Consider any session $i$, backlogged in both servers. By Lemma 3, any backlogged session in the 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 server. Since the server is work conserving, session $i$ can therefore be ahead in the packet server by as much as $(N-1)L_{max}$, where $N$ is the number of sessions sharing the outgoing link.

The bound of $(N-1)L_{max}$ 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

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

Assume two packets arrive at the server simultaneously, one from session $i$ and the other from 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 server transmits the session $j$ packet first, the service received by session $j$ in the packet server can be ahead by $(L_i/\rho_i)p_j$. This reasoning gives rise to the second upper bound of Lemma 5.

**C. Delay Analysis of PRPS**

Based on the bounds on the discrepancy between the service offered by the packet and fluid servers at any time during a session busy period, we can bound the latency of a PRPS. We will now show that a PRPS is an $\mathcal{LR}$ server with a latency of $L_i/\rho_i + L_{max}/r$, the same as that of a WFQ server.

**Lemma 6:** A PRPS is an $\mathcal{LR}$ server with a latency of $L_i/\rho_i + L_{max}/r$.

**Proof:** The proof is similar to that of Lemma 3. We will use the same notations as in the proof of Lemma 3. Consider any sequence of packets $p_1, p_2, \cdots, p_k$ transmitted by the packet server during a system busy period that began at time zero. Let the last packet $p_k$ in this sequence be from session $i$. Assume that the busy period in progress for session $i$ during the service of this packet began at time $\tau$. Let $t$ be any time during the transmission of the packet $p_k$ in the packet server. As before, we will use $\tilde{W}_{Fj}^P(\tau, t)$ to denote the service provided by the fluid server to the packets of this busy period of session $i$ until time $t$. Let $\tilde{W}_{Fj}^P(\tau, t)$ denote the corresponding service in the packet server, without considering the partial service received by packets in service, and $\tilde{W}_{Fj}^P(\tau, t)$ when this partial service is taken into account. We need to show that

$$W_{Fj}^P(\tau, t) \geq \max [0, \rho_i(t - \tau - L_i/\rho_i - L_{max}/r)]. \quad (3.37)$$

at any time $t$ during the transmission of packet $p_k$ in the packet server.

We will distinguish two cases.

**Case 1:** $Fj \leq Fk$, for $1 \leq j < k$. In this case we will first show that the packet $p_k$ cannot complete service in the fluid server before it completes service in the packet server. That is

$$\tau_k \geq t_k. \quad (3.38)$$

We will prove this by contradiction. Assume $\tau_k < t_k$. Let $p_n$ be the packet in service in the packet server at time $\tau_k$. If $\tau_k \geq t_{k-1}$, then the total service offered by the fluid server during the system busy period until time $\tau_k$ must satisfy

$$\sum_{j=1}^{N} \tilde{W}_{Fj}^F(0, \tau_k) > \sum_{j=1}^{k} L^j. \quad (3.39)$$

This inequality holds because packets $p_1, p_2, \cdots, p_{k-1}$ have already arrived by time $\tau_k$ and have finishing potentials less than or equal to that of $p_k$. Similarly, for the packet server

$$\sum_{j=1}^{N} \tilde{W}_{Fj}^P(0, \tau_k) \leq \sum_{j=1}^{k} L^j. \quad (3.40)$$

The left-hand sides of (3.39) and (3.40) must be equal, resulting in a contradiction. Now assume $\tau_k < t_{k-1}$. Let $p_n$ be the packet in service in the packet server at time $\tau_k$. By summing up the total service offered by the two servers in the same manner, we can arrive at the inequalities

$$\sum_{j=1}^{N} \tilde{W}_{Fj}^F(0, \tau_k) > \sum_{j=1}^{n} L^j + L^k$$

$$\sum_{j=1}^{N} \tilde{W}_{Fj}^P(0, \tau_k) \leq \sum_{j=1}^{n} L^j$$

which again results in a contradiction.

Thus, we have $\tau_k \geq t_k$. According to Theorem 2, the fluid RPS is an $\mathcal{LR}$ server with zero latency. Therefore

$$\tilde{W}_{Fj}^P(\tau, t_k) \geq \rho_i(t_k - \tau). \quad (3.41)$$
From (3.38) and (3.41), and the fact that \( \hat{W}_{i,j}^F(\tau, t_k) = \hat{W}_{i,j}^F(\tau, \tau_k) \), we get
\[
\hat{W}_{i,j}^P(\tau, t_k) \geq \rho_i(t_k - \tau). \tag{3.42}
\]
At any time \( \tau \) during the transmission of packet \( p_k \) in the packet server, we must have
\[
W_{i,j}^P(\tau, t) \geq \max[0, \hat{W}_{i,j}^P(\tau, t) - L_i] \tag{3.43}
\]
where \( L_i \) is the maximum size of a session \( i \) packet. Combining (3.42) and (3.43)
\[
W_{i,j}^P(\tau, t) \geq \rho_i(\tau - t) - L_i, \geq \max[0, \rho_i(\tau - t - L_i/\rho_i - L_{\text{max}}/\tau)].
\]

**Case 2:** Before time \( t_{k-1} \), the packet server has transmitted at least one packet with timestamp larger than that of packet \( p_k \). Let \( p_m \) be the latest packet transmitted with \( k^m > k^k \). Packets \( p_{m+1}, p_{m+2}, \ldots, p_k \) must have arrived after time \( t_{m-1} \) when the packet \( p_m \) started receiving service. Packets \( p_{m+1}, p_{m+2}, \ldots, p_{k-1} \) have finishing potentials less than that of packet \( p_k \) and may complete service in the fluid server before the former. The actual departure time \( \tau_k \) of packet \( p_k \) in the fluid server depends on the arrival times of the packets \( p_1, p_2, \ldots, p_k \). Let \( \tau_k^* \) be the maximum value of \( \tau_k \) over all arrival times of these packets that preserve the sequence of departures from the packet server. Since packets \( p_{m+1} \) through \( p_{k-1} \) have finishing potentials no larger than that of \( p_k \), it is possible that the fluid server completes packets \( p_{m+1} \) through \( p_{k-1} \) before completing \( p_k \). Hence, \( \tau_k^* \) must satisfy the inequality
\[
\tau_k^* \geq t_{m-1} + \sum_{j=m+1}^{k} \frac{L_j}{\tau}, \geq t_m - L_{\text{max}}/\tau + \sum_{j=m+1}^{k} \frac{L_j}{\tau}, \geq t_k - L_{\text{max}}/\tau. \tag{3.44}
\]
Since the fluid RPS is an \( L\mathcal{R} \) server with zero latency, we have
\[
\tau_k \leq \frac{W_{i,j}^F(\tau, \tau_k)}{\rho_i} + \tau \tag{3.45}
\]
\[
\leq \frac{W_{i,j}^P(\tau, t_k)}{\rho_i} + \tau. \tag{3.46}
\]
Using (3.43), (3.44), and (3.46), we get
\[
W_{i,j}^P(\tau, t) \geq \max[0, \rho_i(\tau - t - L_i/\rho_i - L_{\text{max}}/\tau)]. \tag{3.47}
\]

Note that this latency is the same as that of WFQ. Thus, any PRPS 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 RPS's.

**IV. Fairness of RPS**

In our definition of RPS's, 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 an RPS receives an average service at least equal to its reserved rate over any active period. 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 RPS, backlogged sessions are serviced at the same normalized rate in steady state, unfairness in service can occur only when an idle session becomes backlogged. If the estimated system potential at that time is far below that of the backlogged sessions, the new session may receive exclusive service for a long time until its potential rises to that of other backlogged sessions.

There is no common accepted method for estimating the fairness of a scheduling algorithm. In general, we would like the system to always serve sessions proportional to their reservations and never penalize sessions for bandwidth they received earlier. The measure of fairness that we will use is an extension of Golastani’s definition [12]. Let us assume that at time \( \tau \) two sessions \( i, j \) become greedy, requesting an infinite amount of bandwidth. Thus, the two sessions will be continuously backlogged in the system after time \( \tau \). A scheduler is considered to be fair if the difference in normalized service offered to the two sessions \( i, j \) during any interval of time \( (t_1, t_2) \) after time \( \tau \) is bounded. That is
\[
\left| \frac{W_i(t_1, t_2)}{\rho_i} - \frac{W_j(t_1, t_2)}{\rho_j} \right| \leq \mathcal{FR} \tag{4.1}
\]
where \( \mathcal{FR} \) 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 \( \tau \) 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 sessions have an infinite supply of packets after time \( \tau \), 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 sessions in a specified interval of time.
Let us denote with $\Delta P$ the maximum difference between the system potential and the potential of the sessions being serviced in an RPS. The following theorem formalizes our basic result on the fairness properties of RPS's.

**Theorem 3:** If the system potential function in an RPS never lags behind more than a finite amount $\Delta P$ from the potential of the sessions that are serviced in the system, the difference in normalized service offered to any two sessions during any interval of time that they are continuously backlogged is also bounded by $\Delta P$. That is, if $\Delta P < \infty$, then for all $i, j \in B(t_1, t_2)$ during the interval $(t_1, t_2)$

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

The proof of this theorem is omitted because of space constraints but can be found in [23]. The theorem applies to the fluid system. A real system can only use a PRPS. We will now expand the above theorem to prove that a similar relationship holds for the packet version of the algorithm. Let us define $C_i$ as

$$C_i = \min \left[ (N - 1)\frac{L_{\text{max}}}{\rho_i}, \max_{1 \leq n \leq N} \left( \frac{L_n}{\rho_n} \right) \right].$$

That is, $C_i$ is the maximum normalized service that a session can receive over any interval in the packet server in excess of that offered by the fluid server. As before, let $W_i^P(t_1, t_2)$ denote the normalized service received by session $i$ in the packet server including the partial service received by any packet currently under service.

**Theorem 4:** Let $i$ and $j$ be two sessions that became greedy at time $\tau$ in a PRPS. The following bound holds for every time interval $(t_1, t_2)$ after time $\tau$:

$$\frac{W_i^P(t_1, t_2)}{\rho_i} - \frac{W_j^P(t_1, t_2)}{\rho_j} \leq \max \left( \Delta P + C_j + L_{\text{max}} + L_j, \Delta P + C_i + L_{\text{max}} + L_i \right).$$

A detailed proof of this theorem can be found in [23]. Since WFQ is a PRPS with $\Delta P = 0$, we obtain the following result on the fairness of a WFQ scheduler by setting $\Delta P = 0$ in (4.2).

**Corollary 1:** For a WFQ scheduler

$$\frac{W_i^P(t_1, t_2)}{\rho_i} - \frac{W_j^P(t_1, t_2)}{\rho_j} \leq \max \left( C_j + L_{\text{max}} + L_j, C_i + L_{\text{max}} + L_i \right).$$

It can be shown that the above bound is tight.

**V. CONCLUSIONS**

In this paper we developed the framework of RPS's for designing schedulers with low latency and bounded unfairness. Fundamental to the definition of RPS's is the system potential function that maintains the global state of the system by tracking the service offered by the system to all sessions sharing the outgoing link. Similarly, the state of each session is represented by a session potential function. In a packet server the system potential and session 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 that the system potential function must satisfy for the server to provide a delay bound, maximum burstiness, and buffer requirements identical to those of WFQ.

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 [20], 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 RPS 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 system. The algorithms in [20], however, avoid this need by maintaining the system potential only as an approximation of the actual global state in the fluid model, and recalibrating the system potential periodically to correct any discrepancies.

In the FFQ algorithm this recalibration is done at frame boundaries, while in SPFQ the recalibration occurs at packet boundaries. This gives rise to two algorithms with the same delay bound but with slightly different fairness properties. Both algorithms, however, provide bounded unfairness and $O(1)$ timestamp computations.

**REFERENCES**


Dimitrios Stiliadis (S’95–M’96) received the Diploma in computer engineering and information science from the University of Patras, Greece, in 1992, and the M.S. and Ph.D. degrees in computer engineering from the University of California, Santa Cruz, in 1994 and 1996, respectively.

Since June 1996, he has been with the High-Speed Networks Research Department, Bell Laboratories, Holmdel, NJ, working on parallel architectures for packet forwarding engines. His current research interests are on high-speed router architectures, traffic scheduling, and packet switching systems.

Anujan Varma (S’83–M’95) received the Ph.D. degree in computer engineering from University of Southern California, Los Angeles, in 1986.

He is with the University of California, Santa Cruz, as a Professor of computer engineering. He was with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, until 1991. His research interests lie in the areas of packet switching, traffic management in broadband networks, and optical networks. He has consulted extensively for the networking and semiconductor industries. He has published more than 70 papers and holds six patents.

Dr. Varma has received several awards including the National Science Foundation Young Investigator Award, the IEEE Darlington Award, and a teaching innovation award from the University of California. He is a member of the Association for Computing Machinery (ACM).