A Self-Clocked Fair Queueing Scheme for Broadband Applications

S. Jamaloddin Golestani
Bellcore
445 South Street
Morristown, NJ 07960-6438

Abstract

An efficient fair queueing scheme which is feasible for broadband implementation is proposed and its performance is analyzed. We define fairness in a self-contained manner, eliminating the need for the hypothetical fluid-flow reference system used in the present state of art and thereby removing the associated computational complexity. The scheme is based on the adoption of an internally generated virtual time *as the index of work progress, hence the name* self-clocked fair queueing. *We prove that the scheme possesses the desired fairness property and is nearly optimal, in the sense that the maximum permissible disparity among the normalized services offered to the backlogged sessions is never more than two times the corresponding figure in any packet-based queueing system.*

1 Introduction

Queueing systems have been traditionally studied by using probabilistic methods of analysis. For the queueing systems encountered in conventional voice and data networks, this type of analysis has been adequate, since the performance measures of interest are conveniently expressed in probabilistic terms. However, the emergence of new technologies in the field of communications and the possibility of integrating a wide range of services into multi-media packet-switched networks have given rise to the need for non-probabilistic measures of performance, such as the maximum end-to-end delay or the minimum throughput. Theoretically, such measures of performance can still be expressed in probabilistic terms, by requiring that they be violated, only, by a small probability such as 10^{-10}. From a practical standpoint, however, statistical modeling and analysis or simulation of queueing networks with such precision is often infeasible, calling for alternative approaches to the study of queueing systems.

Recently, there have been attempts to study the worst-case behavior of queueing networks in deterministic ways, and to develop service disciplines which can provide worst-case performance guarantees [1, 2, 4, 11, 5, 6, 8, 9, 10]. Usually, in this type of analysis, the input traffic to the queue also has to be modeled in a deterministic way, by specifying some kind of permissible traffic envelope, which is never exceeded by the input. A recent study in this area is the work of Parekh and Gallager [9, 10] which analyzes the performance of a queueing network with fair queueing service discipline and derives upper bounds on the end-to-end delays when the input traffic streams conform to the leaky bucket characterization. It is, therefore, shown that fair queueing can be used in conjunction with the leaky bucket admission policy to enforce maximum delay guarantees in a packet network. However, the fair queueing scheme proposed in [9, 10] is based on a hypothetical fluid-flow reference system to determine the fair order of packet transmissions. This approach leads to considerable computational complexity and renders the scheme infeasible for high speed applications.

In this paper, we develop a self-contained approach to fair queueing which does not involve a hypothetical queueing system as reference in defining fairness. Our approach leads to a fair queueing scheme with a much simpler implementation, thereby enabling the application of fair queueing to high speed networks. The performance of the scheme is analyzed and its fairness properties are established.

While the recent studies highlight the potential benefit of fair queueing in the provision of worst-case performance guarantees, fair queueing was originally developed as an attempt to maintain fairness in the amount of services provided at a service point to the competing users. Unlike the FIFO queueing discipline where a session can increase its share of service by presenting more demand and keeping a larger number of packets in the queue, the primary goal in fair queueing is to serve sessions in proportion to some prespecified service shares, independent of the queueing load presented by the sessions. Round robin service discipline [7], which is an early form of fair queueing, assumes

0743-166X/94 $3.00 © 1994 IEEE

an equal service share for all the sessions and an equal length for all the packets. It provides service to the sessions in a round robin fashion, picking one packet for service from each session with backlogged traffic, and then proceeding to the next session.

When the lengths of packets are not the same and/or the service shares assigned to the sessions are not equal, the definition of fair queueing and the right order of providing service to the sessions becomes a more subtle matter. To formulate fair queueing in this more general case, Demers, et al, [3] first apply the notion of fairness to an idealized fluid-flow traffic environment, and then use the outcome to specify fair queueing for the actual packet-based traffic scenario. With a fluid flow model of traffic, the service may be offered to sessions in arbitrarily small increments. Equivalently, it may be assumed that multiple sessions can receive service in parallel. As the result, it is possible to divide the service among the sessions, at all times, exactly in proportion to the specified service shares. In this paper, we shall refer to this form of service discipline as *fluid-flow fair queueing* (FFQ).

Obviously, fluid-flow fair queueing cannot be applied to the actual packet-based traffic scenarios, where only one session can receive service at a time, and where an entire unit of traffic, referred to here as a packet, must be served before another unit is picked up for service. Demers, et al, [3] extend the definition of fair queueing to this case by requiring that packets be picked up for service in the order that they would finish service according to the FFQ scheme in the fluid flow scenario. The same approach has later been adopted in [9, 10]. We shall refer to this scheme, following Demers, et al, as *packet-by-packet fair queueing* (PFQ). The FFQ and PFQ schemes, as called here, are referred to in [9, 10] as *generalized processor sharing* (GPS) and *packet-by-packet generalized processor sharing* (PGPS), respectively.

Another work, which shall be cited in connection with fair queueing, is Zhang's *virtual clock* scheme [12]. Even though this scheme, in spite of the declared objective, does not provide fair services to the users, the notion of virtual clock, first adopted by her, has proven to be an effective tool for formulating fairness and representing the progress of work in the queueing system. The similar notion of *virtual time* has been defined and used by Parekh and Gallager as a major tool in developing a realization for the PFQ scheme. It should be noted that the virtual time used in the realization of the PFQ scheme is defined in association with the FFQ scheme. This arrangement is only natural because the FFQ scheme serves as the reference for specifying the order of services in the PFQ scheme. However, it leads to considerable computational complexity, especially at high transmission speeds, due to the need for simulating events in the hypothetical FFQ system.

This paper presents a different approach to defining fair queueing in a packet-based traffic environment. We define fair queueing in a self-contained manner and avoid using a hypothetical queueing system as reference in determining the fair order of services. This objective is accomplished by adopting a different notion of virtual time. Instead of linking virtual time to the work progress in the FFQ system, we use a virtual time function which depends on the progress of work in the actual packet-based queueing system. Referring to this feature, we call the scheme *self-clocked fair queueing* (SCFQ). In addition, the internal generation of the virtual time involves negligible overhead, as the virtual time is simply extracted from the packet situated at the head of the queue. This approach eliminates the computational complexity that is associated with the PFQ scheme and provides a simple and feasible method for the realization of fair queueing in broadband packet networks, such as the ATM.

The rest of this paper is organized as follows. Section 2 describes the FFQ and PFQ schemes and develops a virtual time implementation for the latter. The main ideas and results in this section are due to Parekh and Gallager [9] and Demers, et al, [3], even though we have adopted a somewhat different presentation, setting the stage for the rest of the paper. In Section 3, after studying the computational complexity of the PFQ scheme, the self-clocked fair queueing (SCFQ) scheme is introduced. Section 4 is devoted to the analysis of the SCFQ scheme. In this section, we prove that the services received by any pair of backlogged sessions, normalized to the corresponding service shares, stay close to each other. Moreover, it is shown that the SCFQ scheme is nearly optimal in the sense that the maximum permissible difference among the normalized services offered to the backlogged sessions is never more than two times the corresponding figure for any packet-based queueing system. The paper is finished with some concluding remarks in Section 5.

2 Fair Queueing Systems

We begin by introducing some notations. Consider the queueing system at a link with the transmission speed C. Let us denote by $\mathcal{K}$ the set of sessions k set up on this link, and by $r_k, \quad k \in \mathcal{K}$, the service share allocated to session k. Define by $A_k(t), \quad t > 0$, the aggregated length of packets of session k arrived dur-

ing $[0,t)$. Similarly, define by $W_k(t)$, $t > 0$, the aggregated length of the traffic of session k transmitted during $(0,t)$. Notice that included in $A_k(t)$ are only the packets which are fully received prior to time t, since the arrival time of a packet is determined by the reception of its last bit. In contrast, our definition of $W_k(t)$ includes any part of the traffic from k transmitted by time t, whether or not it encompasses complete packets. We treat $W_k(t)$ as a continuous function, so it may be defined, more accurately, as the time spent by the server on session k during$(0,t)$, times the server speed C. Let

$$Q_k(t) \triangleq A_k(t) - W_k(t), \qquad k \in \mathcal{K}, \quad (1)$$

$$w_k(t) \triangleq \frac{1}{r_k} W_k(t), \qquad k \in \mathcal{K}, \quad (2)$$

and

$$w_k(t_1,t_2) \triangleq w_k(t_2) - w_k(t_1), \qquad k \in \mathcal{K}. \quad (3)$$

Assuming that the queue has been empty at time 0, $Q_k(t)$ is the total length of the backlogged traffic at time t associated with session k, including residue of any packet partially transmitted before t. $w_k(t)$ represents the total service provided to session k during $(0,t)$ normalized to the corresponding transmission rate. We refer to w_k as the *normalized service* received by session k. Accordingly, $w_k(t_1,t_2)$, $t_2 > t_1$, is the normalized service received by k during (t_1,t_2). Define a session k to be *backlogged* at time t if $Q_k(t) > 0$; otherwise call it *absent* at t. Finally, let us define $\mathcal{B}(t)$ as the set of sessions which are backlogged at t, $\mathcal{B}(t_1,t_2)$ as the set of sessions which are backlogged during the entire interval (t_1,t_2), and $\mathcal{A}(t_1,t_2)$ as the set of sessions which are absent during the entire interval (t_1,t_2), i.e.,

$$\mathcal{B}(t) \triangleq \{k, \text{ s.t. } Q_k(t) > 0\}, \quad (4)$$

$$\mathcal{B}(t_1,t_2) \triangleq \{k, \text{ s.t. } Q_k(\tau) > 0, \text{ for } t_1 < \tau < t_2\}, \quad (5)$$

$$\mathcal{A}(t_1,t_2) \triangleq \{k, \text{ s.t. } Q_k(\tau) = 0, \text{ for } t_1 < \tau < t_2\}. \quad (6)$$

Fluid-flow fair queueing (FFQ) is defined as the service discipline according to which the normalized services $w_k(t)$ received by different backlogged sessions k increase in time with the same rate, i.e., for any interval (t_1,t_2),

$$w_k(t_1,t_2) = w_j(t_1,t_2), \qquad k,j \in \mathcal{B}(t_1,t_2). \quad (7)$$

While a session is not backlogged, it receives no service, i.e.,

$$w_k(t_1,t_2) = 0, \qquad k \in \mathcal{A}(t_1,t_2). \quad (8)$$

The above notion of ideal fairness is only applicable to a hypothetical fluid-flow traffic scenario where service can be offered to sessions in arbitrarily small increments. In a real packet network, entire packets of a session have to be transmitted before the service may be shifted to another session. Therefore, it is not possible to satisfy (7), exactly and for all intervals of time. It is possible, however, to attempt at keeping the normalized services $w_k(t_1,t_2)$ received by different backlogged sessions k close to each other. Depending on how exactly one tries to accomplish this task, different fair queueing algorithms may be conceived.

The *packet-by-packet fair queueing* (PFQ) algorithm, originally proposed by Demers, et al [3], and later studied by Parekh and Gallager [9], is defined as follows. First a fluid-flow fair queueing (FFQ) system is considered and the order that packets *finish service* in accordance with the fair queueing rule of (7) is determined. Notice that in this FFQ system, service can be provided in arbitrarily small increments and several packets may be served in parallel. In the actual packet-based queueing system, each time the server becomes free, service is offered to that packet which would be the first, among the packets present in the queue, to finish service in the hypothetical FFQ system.

It has been proven [3, 9] that in the PFQ system, as defined above, each packet will finish service within θ seconds of its finishing time in the corresponding FFQ system, where θ is the transmission time of a packet with the largest possible size. Based on this result, it can be easily shown that the PFQ scheme conforms to our notion of fairness, i.e., the normalized service $w_k(t_1,t_2)$ received by different sessions k that are backlogged during (t_1,t_2) remain close to each other. Unfortunately, the realization of the PFQ scheme is not simple since, by definition, it requires that the events in the corresponding FFQ system be simulated in the real-time.

The simplest realization that has been suggested for the PFQ scheme [9] is based on the evaluation of a time function associated with the corresponding FFQ system, which represents the progress of work in that system. This function, called *virtual time*, has a rate of increase in time equal to that of the normalized service received by any backlogged session in the FFQ system. Next, we present some definitions.

Definition 1 *Any maximal interval of time during which the server is busy without interruption, is called a busy period.*

Clearly, busy periods only depend on the traffic arrival pattern and the server speed, and are independent of the specific queueing scheme used, provided it is work-conserving. Therefore, we conclude that the

busy periods of the PFQ and the associated FFQ system coincide with each other. In this paper, each time a busy period is considered, without loss of generality, we assume that it starts at $t = 0$.

We define the *virtual time* of the FFQ system, $v(t)$, as a function of time which changes with a rate equal to the rate of increase of $w_k(t)$, for any backlogged session k, i.e.,

$$\frac{d}{dt}v(t) \triangleq \frac{d}{dt}w_k(t), \qquad k \in \mathcal{B}(t). \tag{9}$$

Since the existence of the derivative on the right hand side of (9) has not been established, we provide a more formal definition:

Definition 2 *Consider a busy period of the FFQ system, beginning at $t = 0$. The* virtual time *of the FFQ system, $v(t)$, is defined as the function which satisfies the following:*

$$v(0) = 0, \tag{10}$$

$$v(t_2) - v(t_1) = w_k(t_1, t_2), \quad k \in \mathcal{B}(t_1, t_2), \tag{11}$$

where (t_1, t_2) is an arbitrary subinterval of the busy period.

Therefore, while a session is backlogged in the FFQ system, the normalized service it receives is equal to the growth of the virtual time of the system. To derive an expression for the evaluation of $v(t)$, consider any subinterval (t_1, t_2) of the busy period during which no session changes status. It follows that each session k belongs to either $\mathcal{B}(t_1, t_2)$ or $\mathcal{A}(t_1, t_2)$, i.e., $\mathcal{B}(t_1, t_2) \cup \mathcal{A}(t_1, t_2) = \mathcal{K}$. Multiplying both sides of (11) by r_k, and summing up the equation over sessions $k \in \mathcal{B}(t_1, t_2)$, we get

$$\begin{aligned}(v(t_2) - v(t_1)) \sum_{k \in \mathcal{B}(t_1,t_2)} r_k &= \sum_{k \in \mathcal{B}(t_1,t_2)} r_k \cdot w_k(t_1, t_2) \\ &= \sum_{k \in \mathcal{B}(t_1,t_2)} r_k \cdot w_k(t_1, t_2) + \sum_{k \in \mathcal{B}(t_1,t_2)} r_k \cdot w_k(t_1, t_2) \\ &= \sum_{k \in \mathcal{K}} r_k \cdot w_k(t_1, t_2), \end{aligned} \tag{12}$$

where the second equality follows from (8). Next, notice that $\sum_{k \in \mathcal{K}} r_k \cdot w_k(t_1, t_2)$ is the total work done in the system during (t_1, t_2). Since (t_1, t_2) is contained in a busy period,

$$\sum_{k \in \mathcal{K}} r_k \cdot w_k(t_1, t_2) = C \cdot (t_2 - t_1). \tag{13}$$

It follows that

$$v(t_2) - v(t_1) = C \cdot (t_2 - t_1) \left(\sum_{k \in \mathcal{B}(t_1,t_2)} r_k \right)^{-1}. \tag{14}$$

We conclude that $v(t)$ is a piecewise linear function with the slope

$$\frac{dv(t)}{dt} = C \cdot \left(\sum_{k \in \mathcal{B}(t)} r_k \right)^{-1}, \tag{15}$$

which changes whenever the set of backlogged sessions, $\mathcal{B}(t)$, undergoes some change.

Now consider a busy period beginning at $t = 0$, a session $k \in \mathcal{K}$, and the sequence of packets of k which arrive during this busy period. Denote the i'th packet of the sequence by p_k^i, its arrival time by a_k^i, the time it finishes service in the FFQ system by d_k^i, and its length by L_k^i. Define F_k^i, $i = 1, 2, \cdots$, as the virtual time when packet p_k^i finishes service in the FFQ system, i.e.,

$$F_k^i \triangleq v(d_k^i), \qquad i = 1, 2, \cdots, \quad k \in \mathcal{K}. \tag{16}$$

F_k^i may be referred to as the *virtual finishing time* of packet p_k^i, in the FFQ system.

Lemma 1 *The virtual finishing times F_k^i, $i = 1, 2, \cdots$, associated with the packets of each session $k \in \mathcal{K}$, satisfy the following relationship:*

$$F_k^i = \frac{1}{r_k} L_k^i + \max\left(F_k^{i-1}, v(a_k^i)\right), \qquad i = 1, 2, \cdots, \quad k \in \mathcal{K}, \tag{17}$$

where

$$F_k^0 = 0, \qquad k \in \mathcal{K}. \tag{18}$$

Proof. Let

$$b_k^i \triangleq \max\left(a_k^i, d_k^{i-1}\right), \qquad i = 1, 2, \cdots, \tag{19}$$

where, for consistency, $d_k^0 \triangleq 0$. Service of packet p_k^i cannot start before b_k^i, since either p_k^i arrives at b_k^i, $(a_k^i = b_k^i)$, or p_k^{i-1} is in service until b_k^i, $(d_k^{i-1} = b_k^i)$. Moreover, all of the previous packets of k are completely served by b_k^i. Therefore,

$$\begin{aligned}\frac{1}{r_k} L_k^i &= w_k(b_k^i, d_k^i) \\ &= v(d_k^i) - v(b_k^i), \qquad i = 1, 2, \cdots, \end{aligned} \tag{20}$$

where the last equality follows because k is constantly backlogged during (b_k^i, d_k^i). Since, during the busy period, $v(t)$ is monotonically increasing, it follows from (19) that

$$v(b_k^i) = \max\left(v(a_k^i), v(d_k^{i-1})\right), \qquad i = 1, 2, \cdots. \tag{21}$$

We conclude from (20) and (21) that

$$v(d_k^i) = \frac{1}{r_k} L_k^i + \max\left(v(a_k^i), v(d_k^{i-1})\right), \qquad i = 1, 2, \cdots. \tag{22}$$

Now, in view of (16) and the fact that $F_k^0 = 0 = v(d_k^0)$, the lemma follows from (22).
□

The best known implementation method for the PFQ scheme follows from Lemma 1. Since $v(t)$ is an increasing function of time during a busy period, it would be identical to order the packets in terms of the corresponding finishing times d_k^i or to order them in terms of the corresponding virtual finishing times F_k^i; hence the following corollary.

Corollary 1 *In the PFQ system, each time the server becomes free, that packet is picked up for transmission which has the smallest* virtual *finishing time in the FFQ system, among the packets present in the queue.*

Equation (17) provides an iterative algorithm for computing virtual finishing time of a packet in terms of the packet length, virtual finishing time of the previous packet of the same session, and the system's virtual time when the packet arrives. Therefore, the PFQ scheme may be implemented by stamping each packet, upon arrival into the queue, with a service tag equal to the corresponding virtual finishing time, and then serving the packets from the queue in increasing order of the associated service tags. The service tag of a packet may be computed, iteratively, from (17).

3 Self-Clocked Fair Queueing

Implementation of the PFQ scheme, as outlined above, requires the evaluation of the virtual time $v(t)$ of the FFQ system, which is used in (17). According to (15), v(t) is a piecewise linear function with its slope at any point of time t inversely proportional to the sum of the service shares of sessions in the set $\mathcal{B}$(t). Whenever some session k becomes backlogged or ceases to be backlogged in the FFQ system, the slope of $v(t)$ changes, constituting a breakpoint in its piecewise linear form. Therefore, evaluation of $v(t)$ is conceptually simple; it requires keeping track of the set $\mathcal{B}$(t) and its evolution in time.

From a practical standpoint, however, the computational complexity associated with the evaluation of $v(t)$ depends on the frequency of breakpoints in $v(t)$, i.e., the frequency of transitions in and out of the set $\mathcal{B}$(t). Unfortunately, while such transitions can be rather infrequent on the average, occasionally, a large number of them could happen during a single packet transmission time. The reason for this peculiar phenomenon is that in the FFQ system, where $v(t)$ is to be determined, packets are not served one after another; instead one packet from every backlogged session is in service, simultaneously. It is therefore possible that many packets finish service almost simultaneously, but not at exactly the same time. With each packet finishing service, the corresponding session could become absent, should there be no other packet from that session in the queue, thereby forming a new breakpoint in $v(t)$. We conclude that, in general, the number of breakpoints in $v(t)$ in an arbitrarily short period of time, can approach the total number of sessions set up on the transmission link.

The evaluation of $v(t)$ has to be performed in real time, for the implementation of the PFQ algorithm. To state this requirement more accurately, in the PFQ system, let a packet be picked up for service at t_1 and finish service at t_2. At t_2, in order to select the next packet for service, the virtual finishing time of packets arrived during (t_1, t_2) must be known. Therefore, evaluation of $v(t)$ for the interval (t_1, t_2) must be completed by t_2. According to the previous arguments, the number of breakpoints in $v(t)$ during (t_1, t_2) can be as high as the total number of sessions in $\mathcal{K}$.

We now consider the computational complexity associated with implementing the PFQ scheme in a broadband ATM network. A packet in our discussion refers to a unit of data which must be transmitted as a whole, before another unit is picked up for service. For an ATM network, the unit of data whose transmission should not be preempted by others is an ATM cell. Therefore, the term packet as used in the present context is synonymous to a cell in case of an ATM network. With respect to the above discussion, $t_2 - t_1$ is the transmission time of one cell. This time may be a fraction to a few microseconds for broadband transmission speeds, while the number of sessions can be in the hundreds. We conclude that the PFQ scheme cannot be accurately implemented in a broadband ATM network, since real-time evaluation of $v(t)$ is not feasible.

Here, we present an alternative fair queueing scheme which is much simpler, even though it conforms to our notion of fairness and provides desirable performance. The source of complexity in the PFQ scheme is that it defines fairness with reference to events in a hypothetical system, i.e., the FFQ system. This approach, combined with the fact that in the FFQ system several packets receive service simultaneously, contributes to the computational complexity of the PFQ scheme. The *self-clocked fair queueing* (SCFQ) scheme described in this section is also based

on the notion of the system's virtual time, viewed as the indicator of progress of work in the system, except that the virtual time is referenced to the actual queueing system itself, rather than to a hypothetical system. Moreover, instead of using an abstract analytical definition for the virtual time, we have adopted as the virtual time a quantity that naturally arises in the process of the algorithm. To understand the idea, observe from (16) that for the PFQ scheme the service tag stamped in a packet is equal to the virtual time when the packet finishes service. This observation suggests that the system's virtual time at any moment t may be estimated from the service tag of the packet receiving service at t. With the above remarks in mind, we now proceed to define the SCFQ scheme based on the following algorithm.

1. Each arriving packet p_k^i is tagged with a service tag $\hat{F}_k^i$ before it is placed in the queue. The packets in the queue are picked up for service in increasing order of the associated service tags.

2. For each session k, the service tags of the arriving packets are iteratively computed as

$$\hat{F}_k^i = \frac{1}{r_k} L_k^i + \max\left(\hat{F}_k^{i-1}, \hat{v}(a_k^i)\right), \qquad i = 1, 2, \cdots, \quad k \in \mathcal{K}, \tag{23}$$

where

$$\hat{F}_k^0 = 0, \qquad k \in \mathcal{K}. \tag{24}$$

3. $\hat{v}(t)$, regarded as the system's virtual time at time t, is defined equal to the service tag of the packet receiving service at that time. More specifically,

$$\hat{v}(t) \triangleq \hat{F}_\ell^j, \qquad \hat{s}_\ell^j < t \leq \hat{d}_\ell^j, \tag{25}$$

where $\hat{s}_\ell^j$ and $\hat{d}_\ell^j$ respectively denote the times packet p_ℓ^j starts and finishes service.

4. Once a busy period is over, i.e., when the server becomes free and no more packets are found in the queue, the algorithm is reinitialized by setting to zero the virtual time $\hat{v}(t)$, and the packet counts i for each session k.

Based on the previous discussions, it is intuitively expected that the above algorithm should closely resemble the performance of the FFQ and PFQ schemes. The validity of this conjecture will be shown in the next section.

It is worthwhile to develop some qualitative understanding as to how the computation of service tags, in accordance with (23) or (17), enforces fairness in the order of service provisions. Some insight into this question may be gained by first considering a simpler queueing scheme in which the packet service tags are computed as,

$$\hat{F}_k^i = \frac{1}{r_k} L_k^i + \hat{F}_k^{i-1}, \qquad i = 1, 2, \cdots, \quad k \in \mathcal{K}, \tag{26}$$

with $\hat{F}_k^0 = 0$. In this scheme, each time a packet p_k^i finishes service, its service tag $\hat{F}_k^i$ becomes equal to the total normalized service provided to k, up to that time. Therefore, by always offering service to the packet with the lowest service tag in the queue, the scheme tries to equate the normalized services of all the sessions, regardless of how long each session has been backlogged or absent. This arrangement leads to the accumulation of service credit by the absent sessions. For example, at time t, when a packet with the service tag F is in service, let a session k become backlogged for the first time, with the arrival of a long sequence of packets. According to (26), the service tags assigned to the packets of k will start incrementing from zero. Until the service tags assigned to the packets of k reach F, they will override packets from sessions who have been backlogged for some time. In order to prevent this behavior in the queueing system, the following solution should work: once a session becomes backlogged anew, the normalized service opportunity it has missed while being absent, should be added to the service tag of its first new packet. Substitution of the term $max\left(\hat{F}_k^{i-1}, \hat{v}(a_k^i)\right)$ in (23), in lieu of $\hat{F}_k^{i-1}$ in (26), accomplishes the above task exactly, since it amounts to replacing $\hat{F}_k^{i-1}$ with the service tag of the packet in service, if the latter is larger. A similar compensation takes place in (17).

The unfair outcome of computing service tags in accordance with (26) has been previously noted by Zhang in her paper on virtual clock algorithm [12], who then suggests that the term $\hat{F}_k^{i-1}$ in (26) should be replaced with $max\left(\hat{F}_k^{i-1}, a_k^i\right)$, to circumvent the problem of credit accumulation by the bursty users. This solution does not work, however, since contrary to the virtual time, the real time a_k^i is not a true representation of the progress of work in the system upon arrival of packet p_k^i. We see that the virtual clock algorithm, structurally, gets close to what is needed to accomplish fairness, but falls short of actually providing it.

Before turning to the next section, we would like to caution against the hasty conclusion that the SCFQ algorithm is identical to the PFQ scheme, or that $\hat{v}(t)$, as determined in the process of the algorithm, is equal to the virtual time of the FFQ system. In fact we

have shown that the difference between $\hat{v}(t)$ and $v(t)$ is not necessarily bounded and may approach infinity! Another hasty conclusion that one might draw is that a new fair queueing scheme results by defining $\hat{v}(t)$ equal to the service tag of the most recent packet transmitted *prior* to t, instead of the service tag of the packet receiving service *at* time t. We have shown that this slight modification is sufficient to completely break down the fairness property of the SCFQ algorithm! A detailed discussion of these important and counter-intuitive observations are deferred to the forthcoming publications.

4 Fairness Analysis of the SCFQ Scheme

The rest of this paper is partly concerned with comparing the performance of the SCFQ system and the hypothetical FFQ system associated with it. By the FFQ system associated with the SCFQ system, we mean a queueing system with the same service speed, same set of sessions and service shares, and same pattern of arriving traffic, but the FFQ scheme replacing the SCFQ scheme. While the set $\mathcal{K}$ and parameters r_k, a_k^i, and L_k^i are identical for these two systems, other parameters are not. To distinguish between the two systems, we use a hat sign ($\hat{}$) over parameters associated with the SCFQ system. For example, while $Q_k(t)$ stands for the size of backlogged traffic of session k at time t in the FFQ system, the corresponding parameter in the SCFQ system is shown by $\hat{Q}_k(t)$. Note that the busy periods of the SCFQ system are identical to those of the FFQ or PFQ systems, since the SCFQ scheme is also work-conserving.

Whenever we need to generically speak of a parameter in either system, we use a superscript $\mathcal{S}$ over that parameter, where $\mathcal{S}$ may stand for either of the two queueing systems. For example, $v^{\mathcal{S}}(t)$ would represent both $v(t)$ and $\hat{v}(t)$, depending on the system represented by $\mathcal{S}$. Another notational convention that we use is the double-argument function $f(t_1,t_2)$ to represent $f(t_2)-f(t_1)$, where $f(t)$ could be any function of time. The only exception to this notation is the set $\mathcal{B}(t_1,t_2)$, already defined in a different way.

To be precise in the analysis, we need to assume that no packet arrives at exactly the same time as the time when a packet starts service in the SCFQ system. The obvious exception is at the beginning of a busy period, when the queue is empty and, by definition, the first arriving packet is immediately picked up for service. The above assumption is necessary to keep the definition of the SCFQ scheme and some of its properties intact. However, it does not constitute a limitation to the practicality of the scheme. From a practical stand point, this assumption is equivalent to enforcing mutually exclusive access to the variable $\hat{v}(t)$ in the implementation of the algorithm responsible for updating service tags and the system's virtual time. This mutually exclusive access to $\hat{v}(t)$ is necessary, in any case.

We now introduce some new definitions which shall facilitate our analysis of the SCFQ scheme.

Definition 3 *During a busy period beginning with $t=0$, the* missed normalized service *of session k in system $\mathcal{S}$, $u_k^{\mathcal{S}}(t)$, is defined as the function which satisfies the following:*

$$u_k^{\mathcal{S}}(0) = 0, \qquad k \in \mathcal{K}, \tag{27}$$

$$u_k^{\mathcal{S}}(t_1,t_2) = \begin{cases} 0, & k \in \mathcal{B}^{\mathcal{S}}(t_1,t_2), \\ v^{\mathcal{S}}(t_1,t_2), & k \in \mathcal{A}^{\mathcal{S}}(t_1,t_2), \end{cases} \tag{28}$$

where (t_1,t_2) is any subinterval of the busy period during which k remains either backlogged or absent.

Notice from (28) that $u_k^{\mathcal{S}}(t)$ equals the total growth of the system's virtual time up to time t, while k has not been backlogged. Therefore, it represents the service opportunities missed by session k during $(0,t)$ due to being absent; hence the name *missed normalized service*. In order to check the fairness of queueing system $\mathcal{S}$, it is not reasonable to compare the received normalized services $w_k^{\mathcal{S}}(t)$, $k \in \mathcal{K}$, with each other or with $v^{\mathcal{S}}(t)$, since the service opportunities missed by sessions during their absence intervals should also be accounted for. The following definitions are motivated by this observation:

Definition 4 *The* virtual time of session k *in system $\mathcal{S}$, $v_k^{\mathcal{S}}(t)$, is defined as the sum of the missed and the received normalized services of k in system $\mathcal{S}$, i.e.,*

$$v_k^{\mathcal{S}}(t) \triangleq u_k^{\mathcal{S}}(t) + w_k^{\mathcal{S}}(t), \qquad k \in \mathcal{K}. \tag{29}$$

Definition 5 *The* service lag *of a session k in system $\mathcal{S}$ is defined as the difference between the system's and the session's virtual times, i.e.,*

$$\delta_k^{\mathcal{S}}(t) \triangleq v^{\mathcal{S}}(t) - v_k^{\mathcal{S}}(t), \qquad k \in \mathcal{K}. \tag{30}$$

The packetized nature of traffic arrivals leads to a staircase shape for the arrival functions $A_k(t)$. On the other hand, $W_k^{\mathcal{S}}(t)$ is a continuous function. Therefore, in view of (1) as applied to the queueing system

$\mathcal{S}$, we can argue that while $Q_k^{\mathcal{S}}(t) = 0$, no service can be provided to session k, i.e.,

$$w_k(t_1, t_2) = 0, \qquad k \in \mathcal{A}^{\mathcal{S}}(t_1, t_2). \tag{31}$$

Combining (27)–(29) and (31), we get the following corollary:

Corollary 2 *The session virtual times $v_k^{\mathcal{S}}(t)$ satisfy the following:*

$$v_k^{\mathcal{S}}(0) = 0, \qquad k \in \mathcal{K}, \tag{32}$$

$$v_k^{\mathcal{S}}(t_1, t_2) = \begin{cases} w_k^{\mathcal{S}}(t_1, t_2), & k \in \mathcal{B}^{\mathcal{S}}(t_1, t_2), \\ v^{\mathcal{S}}(t_1, t_2), & k \in \mathcal{A}^{\mathcal{S}}(t_1, t_2). \end{cases} \tag{33}$$

In regard to the FFQ system, we notice from (11) and (33) that

$$v_k(t_1, t_2) = v(t_1, t_2), \quad k \in \mathcal{B}(t_1, t_2) \cup \mathcal{A}(t_1, t_2). \tag{34}$$

Next, notice that a busy period can always be divided into subintervals (t_j, t_{j+1}), $j = 1, 2, \cdots$, during which no session changes status and $\mathcal{B}(t_j, t_{j+1}) \cup \mathcal{A}(t_j, t_{j+1}) = \mathcal{K}$. Applying (34) to such subintervals and summing up, leads to the following result:

Corollary 3 *In the FFQ system, the virtual time of each session is always equal to the virtual time of the system, i.e.,*

$$v_k(t) = v(t), \qquad k \in \mathcal{K}. \tag{35}$$

Equivalently, the service lag of each session is always zero:

$$\delta_k(t) = 0, \qquad k \in \mathcal{K}. \tag{36}$$

While for a fluid-flow fair queueing system the service lag of each session always remains zero, for other queueing systems, a session's service lag is the indication of how far behind that session is in comparison to the progress of work in the system, as measured by the system's virtual time.

Theorem 1 *The service lag of each session k in the SCFQ system is bounded as follows:*

$$0 \le \hat{\delta}_k(t) \le \frac{1}{r_k} L_k^{\max}, \qquad k \in \mathcal{K}, \tag{37}$$

where $L_k^{\max}$ is the maximum size of packets of session k.

This theorem, which lays down the basis for the fairness of the SCFQ scheme, is proven through a sequence of 5 lemmas.

Lemma 2 *During each busy period of the SCFQ system, $\hat{v}(t)$ is a nondecreasing function of time.*

Proof. Consider a busy period and any pair of packets p and p', consecutively served in this period. Let the transmission of p and p' start at t and t', $t' > t$, and let the service tags associated with them be $\hat{F}$ and $\hat{F}'$, respectively. Denote the arrival time of p' by a'. We argue that

$$\hat{F}' \ge \hat{F}. \tag{38}$$

We have previously assumed that $a' \neq t$. If $a' < t$, (38) follows since packet p must have the smallest service tag of any packet in the queue at time t. Otherwise, $t < a' \le t'$, and according to (23), $\hat{F}' > \hat{v}(a')$. Also, according to (25), $\hat{v}(a') = \hat{F}$. Therefore, (38) is valid in either case. Since p and p' are an arbitrary pair of consecutively served packets, we conclude that the service tags of all packets transmitted during a busy period form a nondecreasing sequence. The lemma now follows from the definition of $\hat{v}(t)$ in (25). □

Lemma 3 *For each session k and packet p_k^i,*

$$\hat{u}_k(\hat{d}_k^{i-1}, \hat{d}_k^i) = \max\left(0, \hat{v}(\hat{d}_k^{i-1}, \hat{a}_k^i)\right), \quad k \in \mathcal{K}, \quad i = 1, 2, \cdots, \tag{39}$$

where $\hat{d}_k^0 \triangleq 0$.

Proof. Since k is backlogged during $(\hat{a}_k^i, \hat{d}_k^i)$, according to (28),

$$\begin{aligned} \hat{u}_k(\hat{d}_k^{i-1}, \hat{d}_k^i) &= \hat{u}_k(\hat{d}_k^{i-1}, \hat{a}_k^i) + \hat{u}_k(\hat{a}_k^i, \hat{d}_k^i) \\ &= \hat{u}_k(\hat{d}_k^{i-1}, \hat{a}_k^i). \end{aligned} \tag{40}$$

If $\hat{a}_k^i < \hat{d}_k^{i-1}$, session k is backlogged during $(\hat{a}_k^i, \hat{d}_k^{i-1})$; otherwise k is not backlogged during $(\hat{d}_k^{i-1}, \hat{a}_k^i)$. Therefore, it follows from (28) and (40) that

$$\hat{u}_k(\hat{d}_k^{i-1}, \hat{d}_k^i) = \begin{cases} 0, & \hat{a}_k^i \le \hat{d}_k^{i-1}, \\ \hat{v}(\hat{d}_k^{i-1}, \hat{a}_k^i), & \text{otherwise.} \end{cases} \tag{41}$$

From the nondecreasing property of $\hat{v}(t)$ stated in Lemma 2, we get

$$\max\left(0, \hat{v}(\hat{d}_k^{i-1}, \hat{a}_k^i)\right) = \begin{cases} 0, & \hat{a}_k^i \le \hat{d}_k^{i-1}, \\ \hat{v}(\hat{d}_k^{i-1}, \hat{a}_k^i), & \text{otherwise.} \end{cases} \tag{42}$$

The lemma follows from (41) and (42). □

Lemma 4 *Each time a packet finishes service in the SCFQ system, the service lag of the corresponding session becomes zero, i.e.,*

$$\hat{\delta}_k(\hat{d}_k^i) = 0, \qquad \text{for all packets } p_k^i. \tag{43}$$

Proof. We observe from Definition 4 and Lemma 3 that

$$\begin{aligned}\hat{v}_k(\hat{d}_k^{i-1}, \hat{d}_k^i) &= \hat{w}_k(\hat{d}_k^{i-1}, \hat{d}_k^i) + \hat{u}_k(\hat{d}_k^{i-1}, \hat{d}_k^i) \\ &= \frac{1}{r_k} L_k^i + \max\left(0, \hat{v}(\hat{d}_k^{i-1}, \hat{a}_k^i)\right) \\ &= \frac{1}{r_k} L_k^i + \max\left(\hat{v}(\hat{d}_k^{i-1}), \hat{v}(\hat{a}_k^i)\right) - \hat{v}(\hat{d}_k^{i-1}). \end{aligned} \quad (44)$$

We also notice from the definition of $\hat{v}(t)$ and the updating rule in (23) that

$$\hat{v}(\hat{d}_k^i) = \frac{1}{r_k} L_k^i + \max\left(\hat{v}(\hat{d}_k^{i-1}), \hat{v}(\hat{a}_k^i)\right). \quad (45)$$

It follows from (44) and (45) that

$$\hat{v}_k(\hat{d}_k^i) - \hat{v}_k(\hat{d}_k^{i-1}) = \hat{v}(\hat{d}_k^i) - \hat{v}(\hat{d}_k^{i-1}). \quad (46)$$

By rearranging the terms and applying Definition 5, we get

$$\hat{\delta}_k(\hat{d}_k^i) = \hat{\delta}_k(\hat{d}_k^{i-1}). \quad (47)$$

Finally, since by definition $\hat{d}_k^0 = 0$,

$$\hat{\delta}_k(\hat{d}_k^0) = \hat{v}(0) - \hat{v}_k(0) = 0, \quad (48)$$

where the last equality follows from (10) and (32). Considering that (47) holds for any $i = 1, 2, \cdots$, the lemma follows from (47) and (48).
□

Lemma 5 *While a session is absent or each time a session becomes backlogged in the SCFQ system, its service lag is zero, i.e.,*

$$\hat{\delta}_k(t) = 0, \qquad k \notin \hat{\mathcal{B}}(t), \text{or } k \text{ becomes backlogged at } t. \quad (49)$$

Proof. Consider a session $k \notin \hat{\mathcal{B}}(t)$, or a session k which becomes backlogged at t. Let there be i packets served from k during $(0, t)$. It follows that k is not backlogged in the SCFQ system during $(\hat{d}_k^i, t)$. Using (33),

$$\hat{\delta}_k(\hat{d}_k^i, t) = \hat{v}(\hat{d}_k^i, t) - \hat{v}_k(\hat{d}_k^i, t) = 0. \quad (50)$$

We conclude from (50) and Lemma 4, or (48) in case of $i = 0$, that

$$\hat{\delta}_k(t) = \hat{\delta}_k(\hat{d}_k^i) + \hat{\delta}_k(\hat{d}_k^i, t) = 0. \quad (51)$$

□

Lemma 6 *For any session $k \in \hat{\mathcal{B}}(t)$,*

$$0 \le \hat{\delta}_k(t) \le \frac{1}{r_k} L_k^i, \quad (52)$$

where p_k^i is the first packet of k to finish service in the SCFQ system after t.

Proof. Define $\hat{b}_k^i \triangleq \max\left(\hat{d}_k^{i-1}, a_k^i\right)$. First, we argue that

$$\hat{\delta}_k(\hat{b}_k^i) = 0. \quad (53)$$

If $\hat{b}_k^i = \hat{d}_k^{i-1}$, (53) follows from Lemma 4. Otherwise, $\hat{b}_k^i = \hat{a}_k^i > \hat{d}_k^{i-1}$, in which case k becomes backlogged at $\hat{b}_k^i$, and (53) follows from Lemma 5. We conclude from (53) that

$$\begin{aligned}\hat{\delta}_k(t) &= \hat{\delta}_k(\hat{b}_k^i) + \hat{\delta}_k(\hat{b}_k^i, t) \\ &= \hat{\delta}_k(\hat{b}_k^i, t) \\ &= \hat{v}(\hat{b}_k^i, t) - \hat{v}_k(\hat{b}_k^i, t). \end{aligned} \quad (54)$$

Next, it follows from the (23) and (25) that

$$\begin{aligned}\hat{v}(\hat{d}_k^i) &= \frac{1}{r_k} L_k^i + \max\left(\hat{v}(\hat{d}_k^{i-1}), \hat{v}(\hat{a}_k^i)\right) \\ &= \frac{1}{r_k} L_k^i + \hat{v}\left(\max\left(\hat{d}_k^{i-1}, \hat{a}_k^i\right)\right) \\ &= \frac{1}{r_k} L_k^i + \hat{v}(\hat{b}_k^i), \end{aligned} \quad (55)$$

where the second equality follows from the nondecreasing property of $\hat{v}(t)$. Therefore,

$$\hat{v}(\hat{b}_k^i, \hat{d}_k^i) = \frac{1}{r_k} L_k^i. \quad (56)$$

Next, notice that $\hat{d}_k^{i-1} \le t < \hat{d}_k^i$ since p_k^i is the first packet to finish service after t. Also, $a_k^i \le t$, since k is backlogged at t. It follows that

$$\hat{a}_k^i \le \hat{b}_k^i \le t < \hat{d}_k^i, \quad (57)$$

and that k is backlogged in the SCFQ system during $(\hat{b}_k^i, t)$. Since $\hat{v}(t)$ is nondecreasing, we conclude from (56) and (57) that

$$0 \le \hat{v}(\hat{b}_k^i, t) \le \hat{v}(\hat{b}_k^i, \hat{d}_k^i) = \frac{1}{r_k} L_k^i. \quad (58)$$

Since k is backlogged during $(\hat{b}_k^i, t)$, according to (33)

$$\hat{v}_k(\hat{b}_k^i, t) = \hat{w}_k(\hat{b}_k^i, t). \quad (59)$$

The lemma may now be proven by considering two alternative cases. The first case is when the service of p_k^i starts on or after t. In this case, $\hat{w}_k(\hat{b}_k^i, t) = 0$, since $\hat{d}_k^{i-1} \le \hat{b}_k^i$. So, we conclude from (59) that

$$\hat{v}_k(\hat{b}_k^i, t) = 0. \quad (60)$$

The Lemma, for this case, follows from (54), (58), and (60). The second case is when the service of p_k^i starts before t. Since, in this case, the only service

provided to k during $(\hat{b}_k^i, t)$ is partial transmission of p_k^i,

$$0 \leq \hat{v}_k(\hat{b}_k^i, t) = \hat{w}_k(\hat{b}_k^i, t) < \frac{1}{r_k} L_k^i. \qquad (61)$$

On the other hand, in this case, $\hat{v}(t) = \hat{F}_k^i = \hat{v}(\hat{d}_k^i)$. Therefore, in view of (56),

$$\hat{v}(\hat{b}_k^i, t) = \hat{v}(\hat{b}_k^i, \hat{d}_k^i) = \frac{1}{r_k} L_k^i. \qquad (62)$$

The lemma, for this case, follows from (54), (61), and (62).

□

Lemmas 5 and 6 conclude the proof of Theorem 1. Consider the differential service lag function

$$\begin{aligned} \hat{\delta}_k(t_1, t_2) &= \hat{\delta}_k(t_2) - \hat{\delta}_k(t_1) \\ &= \hat{v}(t_1, t_2) - \hat{v}_k(t_1, t_2), \quad k \in \mathcal{K}. \end{aligned} \qquad (63)$$

By Theorem 1,

$$\left|\hat{\delta}_k(t_1, t_2)\right| \leq \frac{1}{r_k} L_k^{\max}, \qquad k \in \mathcal{K}. \qquad (64)$$

Since $\hat{v}_k(t_1, t_2) = \hat{w}_k(t_1, t_2)$, for $k \in \hat{\mathcal{B}}(t_1, t_2)$, the following corollaries result from (63) and (64).

Corollary 4 *For any session $k \in \hat{\mathcal{B}}(t_1, t_2)$,*

$$|\hat{v}(t_1, t_2) - \hat{w}_k(t_1, t_2)| \leq \frac{1}{r_k} L_k^{\max}. \qquad (65)$$

Corollary 5 *For any pair of sessions k and j, $k, j \in \hat{\mathcal{B}}(t_1, t_2)$,*

$$|\hat{w}_k(t_1, t_2) - \hat{w}_j(t_1, t_2)| \leq \frac{1}{r_k} L_k^{\max} + \frac{1}{r_j} L_j^{\max}. \qquad (66)$$

Corollary 5 follows by subtracting the expressions in Corollary 4 as applied to sessions k and j. This result establishes the basic fairness property of the SCFQ scheme. It shows that the normalized services received by different backlogged sessions remain close to each other and that the disparity among them is always bounded. Notice that the upper bound in Corollary 5 is independent of the comparison interval (t_1, t_2). Therefore, as the comparison interval expands, the average rates of normalized services offered to the backlogged sessions converge and the disparity among them vanishes.

We could actually arrive at Corollary 5, in a simpler and more direct way, by starting from Lemma 2, then arguing that while a session k is backlogged $\hat{F}_k^i = \frac{1}{r_k} L_k^i + \hat{F}_k^{i-1}$, and then proceeding with some additional details. Instead, we have preferred to carry out this longer but more comprehensive analysis of the SCFQ system, since the additional results obtained here are useful.

While Corollary 5 confirms that in the SCFQ system, the disparity in the normalized services received by different backlogged sessions is bounded, it does not provide an idea of how good this bound is. This question is answered by the following theorem, stated here without proof.

Theorem 2 *Consider the class of packet-based queueing systems $\mathcal{S}$ for which, given any time interval (t_1, t_2) and any pair of backlogged sessions k and j, the following bound holds,*

$$\begin{aligned} \left|w_k^{\mathcal{S}}(t_1, t_2) - w_j^{\mathcal{S}}(t_1, t_2)\right| &\leq D^{\mathcal{S}}(k, j), \\ & k, j \in \mathcal{B}^{\mathcal{S}}(t_1, t_2), \end{aligned} \qquad (67)$$

where $D^{\mathcal{S}}(k, j)$ does not dependent on the time interval (t_1, t_2). Then, for any system $\mathcal{S}$ of this class, $D^{\mathcal{S}}(k, j)$ always satisfies the following:

$$D^{\mathcal{S}}(k, j) \geq \frac{1}{2}\left(\frac{L_k^{\max}}{r_k} + \frac{L_j^{\max}}{r_j}\right). \qquad (68)$$

By comparing (66) and (68), we conclude that the SCFQ scheme is a near-optimal fair queueing scheme, in the sense that the maximum service disparity allowed by it between any pair of backlogged sessions is never more than two times the corresponding figure in any packet-based queueing system.

5 Conclusion

In this paper, a self-clocked fair queueing scheme (SCFQ) for packet networks has been developed, which is based on a novel self-contained approach to fair queueing. Compared to the packet-by-packet fair queueing (PFQ) [3, 9], the scheme proposed here provides substantial simplicity and ease of implementation. The reason is that, in the PFQ scheme, fairness is defined in reference to the events in the hypothetical FFQ system. Accordingly, the virtual time function, which serves as the measure of the work progress in the system, has to be evaluated for the corresponding FFQ system, for every packet. We avoid this computation by using an internally generated virtual time to reflect the progress of work in the system. Hence the name self-clocked fair queueing. Furthermore, since the virtual time is simply extracted from the packet in the head of the queue, its generation involves minimal data processing. These features make

the SCFQ scheme well-suited for broadband implementation, whereas the computational complexity of the PFQ scheme makes it difficult to implement.

We have studied the performance of the SCFQ scheme from 2 perspectives. The first line of analysis compared the normalized services received by different sessions which are backlogged (in the SCFQ system itself), and proved that the service disparity among sessions is always bounded. Then, it was stated that the maximum permissible service disparity between a pair of backlogged sessions in the SCFQ scheme is never more than two times the corresponding figure for any packet-based queueing system. Therefore, we concluded that the SCFQ scheme represents a near optimal fair queueing scheme.

There are some other important properties that we have been able to demonstrate for the SCFQ scheme, which will be presented in the forthcoming publications. One property concerns the comparison of the services received in accordance with the SCFQ and the FFQ schemes, when the same set of session traffic arrivals is considered. We have shown that the disparity between services received by any session in accordance with the SCFQ and FFQ schemes is always bounded. A more specific property concerns the end-to-end session delays in a network employing the SCFQ scheme in conjunction with the leaky bucket admission policy. For such a network, we have derived end-to-end session delay bounds which are comparable to the bounds established by Parekh and Gallager [9, 10] for a network employing packet-by-packet fair queueing (PFQ) in conjunction with the leaky bucket admission policy.

Based on the above results, we conclude that the SCFQ scheme retains the desirable bounded delay property of the PFQ scheme, while eliminating its undesirable computational complexity. The bounded end-to-end delay feature is an important property useful in multi-media networks carrying real-time traffic. Hence we believe that the SCFQ scheme is well-suited for broadband multi-media networks.

Acknowledgements

The author would like to appreciate T. V. Lakshman, Yow-jian Lin, Mark W. Garrett, and Walter Willinger for helpful comments.

References

[1] R. L. Cruz. A calculus for network delay, part i: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[2] R. L. Cruz. A calculus for network delay, part ii: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

[3] A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. In *Proc. SIGCOMM'89*, pages 1–12, Austin, Texas, September 1989.

[4] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, 1990.

[5] S. J. Golestani. Congestion free communication in high speed packet networks. *IEEE Transactions on Communications*, 32(12):1802–1812, December 1991.

[6] S. J. Golestani. A framing strategy for congestion management. *IEEE Journal on Selected Areas in Communications*, 9(7):1064–1077, September 1991.

[7] E. Hahn. *Round Robin Scheduling for Fair Flow Control.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, December 1986.

[8] J. F. Kurose. On computing per-session performance bounds in high-speed multi-hop computer networks. In *Proc. ACM SIGMETRICS'92 Conf.*, pages 128–139, Newport, RI, June 1991.

[9] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks. In *Proc. IEEE INFOCOM'92*, pages 915–924, 1992.

[10] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks. In *Proc. IEEE INFOCOM'93*, pages 521–530, 1993.

[11] D. Verma, H. Zhang, and D. Ferrari. Guaranteed delay jitter bounds in packet switching networks. In *Proc. TriComm'92*, Chapel Hill, NC, April 1991.

[12] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching. *ACM Transactions on Computer Systems*, 9(2):101–124, may 1991.