Fair Airport Scheduling Algorithms *

Pawan Goyal
AT&T Labs - Research
Florham Park, NJ 07932

Harrick M. Vin
Department of Computer Sciences
University of Texas at Austin
Taylor Hall 2.124, Austin TX 78712

Abstract

We design a class of Fair Airport (FA) algorithms that combines Start-time Fair Queuing with a non-work conserving algorithm in the Rate Controlled Service Discipline (RCSD) class. We derive the fairness and deadline guarantees for FA servers. We demonstrate that by appropriately choosing an algorithm from the RCSD class, algorithms that either allocate only rate or achieve separation of rate and delay allocation and are fair over Fluctuation Constrained variable rate servers can be designed. This method leads to the design of the first fair algorithm that achieves separation of rate and delay allocation. We also show that the FA class contains algorithms with different implementation complexity and performance guarantees and thus enables tradeoffs between the two. Furthermore, since FA contains algorithms that are fair over variable rate servers, they can be employed for achieving hierarchical link sharing. Finally, we demonstrate that the FA algorithms can be generalized to achieve objectives such as hierarchical and/or prioritized fair allocation of residual bandwidth.

1 Introduction

Integrated services networks are required to support a variety of applications (e.g., audio and video conferencing, multimedia information retrieval, ftp, telnet, WWW, etc.) with a wide range of Quality of Service (QoS) requirements. Whereas continuous media applications such as audio and video conferencing require a network to provide QoS guarantees with respect to bandwidth, packet delay, and loss; applications such as telnet and WWW require low packet delay and loss. Throughput intensive applications like ftp, on the other hand, require network resources to be allocated such that the throughput is maximized. A network meets these requirements primarily by appropriately *scheduling* its resources.

An examination of application and network requirements in [15] revealed that a suitable packet scheduling algorithm for integrated services network should: (1) achieve low average as well as maximum delay for low throughput applications (e.g., interactive audio, telnet, etc.); (2) be fair over variable rate servers; and (3) be computationally efficient. To meet these requirements, Start-time Fair Queuing (SFQ) was developed in [15]. Though SFQ meets these requirements, the maximum delay of packets in SFQ depends on the maximum packet length of all the flows [1] at a server [2]. Consequently, if SFQ is employed to provide a priori bounds on packet delay of flows, the maximum packet length and the number of flows at a server would have to be estimated. This estimate may be large and hence the bounds on packet delay may be large. Hence, in networking environments where one of the main objective is to provide pre-specified bounds on packet delay, it may be desirable to employ a scheduling algorithm from one of the following classes.

- Scheduling algorithms that allocate only rate and have delay guarantee similar to Weighted Fair Queuing (WFQ): Several fair algorithms that have delay guarantee similar to WFQ are known (for example, WFQ [8, 18], FFQ

* A preliminary version of this paper appeared in the Proceedings of Network and Operating System Support for Digital Audio and Video, May 1997. This research was supported in part by IBM Graduate Fellowship, IBM Faculty Development Award, Intel, the National Science Foundation (Research Initiation Award CCR-9409666), NASA, Mitsubishi Electric Research Laboratories (MERL), and Sun Microsystems Inc.

[1] A flow is the sequence of packets transmitted by a source to a destination.

[2] We refer to the output port of a switch as a server.

[20], SPFQ [19], WF^2Q [3], WF^2Q+ [4], Leap Forward Virtual Clock [22], etc.). Most of these algorithms are either unfair over variable rate servers or require the number of bits transmitted to be counted to achieve fairness over variable rate servers [15]. Since counting the number of bits transmitted may be expensive, algorithms that do not have this requirement for achieving fairness over variable rate servers are desirable [3].

- Scheduling algorithms that achieve separation of rate and delay allocation: The algorithms that have delay guarantee similar to WFQ provide low delay to low throughput flows by reserving higher rate; this may result into low utilization of network resources. Hence, algorithms that achieve separation of rate and delay allocation and provide lower maximum delay to packets of a flow without increasing the rate reservation are desirable. Though unfair algorithms that achieve separation of rate and delay allocation are known (e.g. Delay EDD [25]), fair algorithms that achieve the same objective are not [4].

To address these limitations, we design a class of Fair Airport (FA) algorithms. An algorithm in the FA class combines SFQ with a non-work conserving algorithm in the Rate Controlled Service Discipline (RCSD) class [10, 24]. We derive the fairness and deadline guarantees for FA servers. We demonstrate that by appropriately choosing an algorithm from the RCSD class, algorithms that either allocate only rate or achieve separation of rate and delay allocation and are fair over Fluctuation Constrained variable rate servers can be designed. This method leads to the design of the *first* fair algorithm that achieves separation of rate and delay allocation. We also show that the FA class contains algorithms with different implementation complexity and performance guarantees and thus enables tradeoffs between the two. Furthermore, since FA contains algorithms that are fair over variable rate servers, they can be employed for achieving hierarchical link sharing [15]. Finally, the FA algorithms can be generalized to achieve objectives such as hierarchical and/or prioritized fair allocation of residual bandwidth. This results into algorithms that achieve objectives similar to those outlined in [21].

The rest of the paper is structured as follows. We present the design of FA algorithms in Section 2. We then present their implementation complexity, fairness guarantee, and deadline properties in Sections 5, 3, and 4, respectively. Two instantiations of FA algorithms are presented in Section 6 and salient features of the FA algorithms are summarized in Section 7. Finally, Section 8 summarizes the results of this paper.

2 Design of Fair Airport Scheduling Algorithms

First, consider the definition of a fair algorithm. Intuitively, allocation of link bandwidth is fair if equal bandwidth is allocated in every time interval to all the flows. This concept generalizes to weighted fairness in which the bandwidth must be allocated in proportion to the *weights* associated with the flows. Formally, if r_f is the weight of flow f and $W_f(t_1, t_2)$ is the aggregate service (in bits) received by it in the interval $[t_1, t_2]$, then an allocation is fair if, for all intervals $[t_1, t_2]$ in which both flows f and m are backlogged, $\frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} = 0$. Clearly, this is an idealized definition of fairness as it assumes that flows can be served in infinitesimally divisible units. The objective of fair *packet* scheduling algorithms is to ensure that $\mid \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \mid$ is as close to 0 as possible [5].

To gain an intuitive understanding of the design of Fair Airport[6] scheduling algorithms, consider the class of Rate Controlled Service Disciplines (RCSD) [11, 24]. A RCSD scheduling algorithm employs a rate controller (also referred to as a shaper) for each flow (see Figure 1). A packet on arrival joins the rate controller for its flow. The rate controller delays packets to ensure that the output traffic from the flow conforms to specified constraints. Once a packet passes through the rate controller, it joins a queue that we refer to as Guaranteed Service Queue (GSQ) [7]. Packets are serviced from GSQ using a work conserving GSQ scheduling algorithm.

Observe that the rate controller limits the service received by a flow from the GSQ in any time interval. Hence, intuitively, the difference in service received by any two flows in any time interval from the GSQ is bounded, i.e.,

[3]The implementation of WF^2Q+ in [4] that reduces the overhead of counting the number of bits by performing this operation only at packet boundaries violates the necessary assumption stated in [2, 4] that the virtual time function should have a minimum slope of 1.

[4]Subsequent to our work, a fair algorithm that achieves separation of rate and delay allocation has been proposed in [21].

[5]Several definitions of fair algorithms have been introduced in the literature. A comparison of various definitions and justifications for the appropriateness of this definition is presented in [12].

[6]The class of Fair Airport scheduling algorithms derives its name from the class of Airport scheduling algorithms presented in [7].

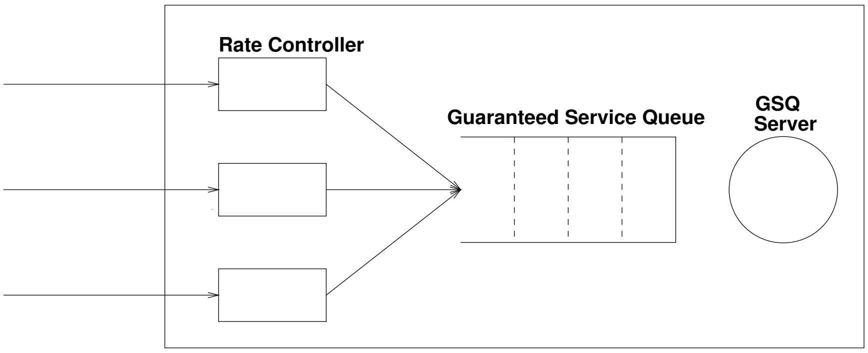

Figure 1: Non-work conserving RCSD

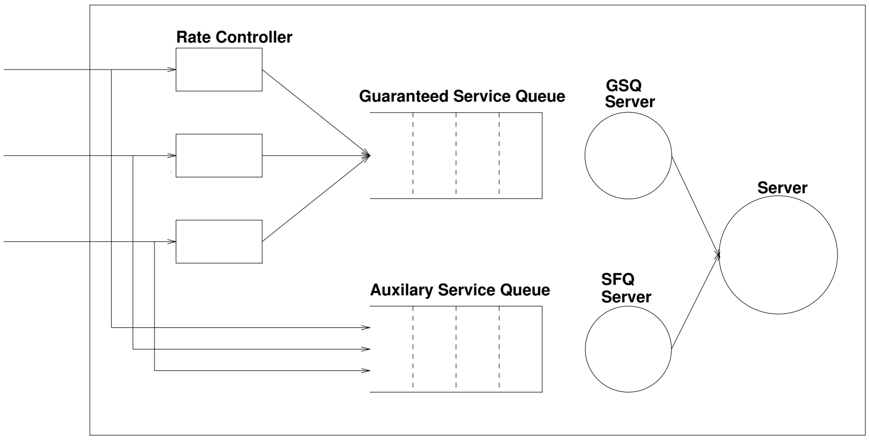

Figure 2: Fair Airport Scheduling Algorithms

the service received from GSQ is fair (we will demonstrate this formally in Section 3). Furthermore, since different scheduling algorithms can be employed for GSQ, different delay guarantees can be provided to flows. Thus, RCSD scheduling algorithms are fair and can provide different delay guarantees. They are, however, non-work conserving since GSQ may be empty and consequently the server may idle even when packets are backlogged at the rate controller. Hence, the challenge is to make the algorithm work conserving while ensuring that it remains fair.

RCSD can be made work conserving by introducing an Auxiliary Service Queue (ASQ) (this approach was first introduced in [7] and later employed in [11]). In such a case, packets on arrival not only join a rate controller, but also ASQ. When there are no packets in the GSQ, packets are scheduled from the ASQ. We assume that the packets of a flow are scheduled from ASQ in sequence. Since each packet joins both ASQ and GSQ, this ensures that packets of a flow will not be reordered. This makes RCSD work conserving while ensuring that packets of a flow are not reordered. We now need to ensure that it continues to remain fair. We can achieve this objective by employing a *fair scheduling algorithm* for the ASQ. In such a case, the difference in service received by two flows from the ASQ will be bounded. Since the difference in service received by flows from GSQ is bounded, we conclude that the difference in aggregate service received by two flows from the server will be bounded, i.e., the resulting algorithm will be fair. Furthermore, since packets are transmitted only earlier from the ASQ than they are guaranteed to depart from the GSQ, its delay guarantee for flows will be the same as the non-work conserving algorithm. Since non-work conserving RCSD can provide different delay guarantees to a flow, we conclude that we can design fair algorithms with different delay guarantees.

Thus, Fair Airport (FA) scheduling algorithms, informally, are defined as fair work conserving RCSD algorithms (see Figure 2). We now present the detailed design of FA scheduling algorithms.

RCSD algorithms may employ rate controllers that enforce different traffic constraints. In FA, we restrict our focus to rate controllers that ensure that the output traffic for a flow is at most at its reserved rate. In such rate controllers, a packet departs at its *expected arrival time*. If r_f is the rate reserved for flow f, then expected arrival time of a packet is informally defined as the time at which it begins service in a server of rate r_f. To formally define this notion, let p_f^j be the j^{th} packet of flow f and $A(p_f^j)$ be its arrival time. Then, the expected arrival time of packet p_f^j, denoted by $EAT(p_f^j, r_f)$, is given as:

$$EAT(p_f^j, r_f) = \max\left\{A(p_f^j), EAT(p_f^{j-1}, r_f) + \frac{l_f^{j-1}}{r_f}\right\} \tag{1}$$

where $EAT(p_f^0, r_f) + \frac{l_f^0}{r_f} = 0$. Then, departure time of packet p_f^j from the rate controller, denoted by $L_{RC}(p_f^j)$, is given as:

$$L_{RC}(p_f^j) = EAT(p_f^j, r_f)$$

Now consider the detailed design of FA algorithms. There are three main components of FA: rate controller, GSQ scheduling algorithm, and ASQ scheduling algorithm. Consider the design of each of the three components:

- *Rate controller*: In FA, a flow receives service through GSQ and ASQ. To ensure that GSQ continues to remain fair, service received by a flow from ASQ should not effect the service received by it from GSQ. Hence, if a packet is serviced from ASQ, it should not effect the departure time of the next packet from the rate controller. We achieve this objective by changing the departure time of a packet from the rate controller to its expected arrival time computed using only the subsequence of packets served through GSQ.

  Let $EAT^{RC}(p_f^j, r_f)$ denote the expected arrival time of packet p_f^j computed at the rate controller using only the subsequence of packets of flow f that were serviced through GSQ prior to p_f^j. Hence, if $p_f^{\theta(j)}$ is the packet prior to p_f^j that was served via GSQ, then using (1) we get:

  $$EAT^{RC}(p_f^j, r_f) = \max\{A(p_f^j), EAT^{RC}(p_f^{\theta(j)}, r_f) + \frac{l_f^{\theta(j)}}{r_f}\}$$

  To illustrate the computation of the expected arrival time at the rate controller, let flow f with rate of $1pkt/s$ send 5 packets at time 0. Let p_f^3 and p_f^4 be served from the ASQ. Then, the expected arrival time of p_f^1 is 0, p_f^2 is 1, p_f^4 is 2, and p_f^5 is 3. We define the departure time of p_f^j from the rate controller as follows:

  $$EAT^{RC}(p_f^j, r_f) - \epsilon \leq L_{RC}(p_f^j) \leq EAT^{RC}(p_f^j, r_f) \tag{2}$$

  To enable efficient implementation of the rate controller, we have allowed the departure time from the rate controller to be at most ϵ before the expected arrival time.

- *GSQ scheduling algorithm*: We had informally argued that the delay guarantee of FA algorithm will be the same as that of the GSQ scheduling algorithm. Hence, to enable design of FA algorithms with different delay guarantees, we do not specify the GSQ scheduling algorithm completely. Any scheduling algorithm can be employed as long as it guarantees that the departure time of packet p_f^j from GSQ, denoted by $L_{GSQ}(p_f^j)$, is:

  $$L_{GSQ}(p_f^j, r_f) \leq EAT^{GSQ}(p_f^j, r_f) + \beta_f^j \tag{3}$$

  where $EAT^{GSQ}(p_f^j, r_f)$ is the expected arrival time of p_f^j computed at the GSQ.

- *ASQ scheduling algorithm*: The ASQ scheduling algorithm should have two characteristics:
  - *It must be fair over variable rate servers.* Since GSQ is given priority over ASQ, the bandwidth of the ASQ server varies over time. Hence, the ASQ scheduling algorithm must be fair over variable rate servers.
  - *It should not utilize the length of a packet in determining when the packet is scheduled.* In FA, a packet that is in the ASQ may be served from the GSQ. To ensure that the ASQ continues to remain fair, such a packet should not effect the service received by a flow from the ASQ. This can be achieved by serving the next packet of the flow when the previous packet would have been scheduled if it had not been serviced via GSQ. Since the packets of a flow may have variable length, this will be feasible if the ASQ scheduling algorithm does not utilize the length of a packet in determining when the packet is scheduled.

SFQ algorithm presented in [15] meets the above requirements. To observe this consider the definition of SFQ. SFQ assigns a start tag and a finish tag to each packet and schedules packets in increasing order of start tags. The start and finish tags of packet p_f^j, denoted by $S(p_f^j)$ and $F(p_f^j)$, respectively, are defined as:

$$S(p_f^j) = \max\{v(A(p_f^j)), F(p_f^{j-1})\} \tag{4}$$

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{r_f} \tag{5}$$

where $v(t)$ is the virtual time of the server and is defined to be the start tag of packet in service at time t.

Observe that SFQ does not make any assumptions about server behaviour. Hence, it is fair over variable rate servers (see [15] for a formal demonstration). Furthermore, since it schedules packets in increasing order of start tags, it does not utilize the length of a packet in determining when it is scheduled. Hence, SFQ meets these two requirements and we employ it as the ASQ scheduling algorithm. To ensure that packets that are serviced from GSQ do not effect the service received by a flow from the ASQ, the start and finish tags of packets of a flow are computed using only the subsequence of packets served from the ASQ.

Hence, the class of FA scheduling algorithms is defined as follows:

1. On arrival, a packet of a flow joins the rate controller for the flow and ASQ. The departure time of packet p_f^j from the rate controller is given by (2).
2. Any algorithm that satisfies (3) can be used as the GSQ scheduling algorithm.
3. The ASQ scheduling algorithm is SFQ.
4. A packet is removed from the rate controller when it starts service in the ASQ.
5. A packet that joins GSQ is dequeued from ASQ after it has been serviced by the GSQ server. On removal of such a packet, the start tag of the next packet of that flow in the ASQ (if such a packet exists) is set to the start tag of the packet being removed. (This, in effect, computes the start and finish tags based on the subsequence of packets served from ASQ.)
6. The server gives priority to GSQ over ASQ. However, it does not preempt the transmission of a packet from ASQ.

To summarize, FA algorithms are work conserving realizations of RCSD algorithms that employ SFQ as the ASQ scheduler. To the best of our knowledge, SFQ is the only scheduling algorithm that meets the requirements of the ASQ scheduling algorithm that we had outlined earlier. Hence, SFQ is central to the design of FA algorithms.

In the following sections, we derive the fairness and deadline guarantees and then determine the implementation complexity of the FA algorithms.

3 Fairness Guarantee

To prove fairness guarantee of the Fair Airport algorithm, we first derive lower and upper bounds on the service received by a flow through GSQ in Lemmas 1 and 2, respectively. We then prove the fairness guarantee in Theorem 1. In what follows, a flow will be considered backlogged, unless stated otherwise, if it is backlogged at the FA server. Observe that if a flow is backlogged, it may or may not be backlogged at the GSQ server. Let $W_f^{GSQ}(t_1, t_2)$ and $W_f^{ASQ}(t_1, t_2)$ be the service received by flow f during $[t_1, t_2]$ from GSQ and ASQ, respectively. Let $\beta_f^{max} = \max\{\beta_f^j\}$ and $l_f^{max} = \max\{l_f^j\}$.

Lemma 1 *If the GSQ scheduler ensures (3), then for all intervals $[t_1, t_2]$ in which the flow is backlogged throughout the interval:*

$$r_f(t_2 - t_1) - r_f(\beta_f^{max} + \epsilon) - l_f^{max} \leq W_f^{GSQ}(t_1, t_2)$$

Proof: Let packet p_f^j refer to the j^{th} packet of flow f serviced by GSQ. Since flow f is backlogged during $[t_1, t_2]$ and the rate controller lets a packet into the GSQ at most ϵ before its expected arrival time, the maximum value of the expected arrival time of a packet that is served before time t is $t + \epsilon$. Hence, the maximum value of the expected arrival time of the first packet that is scheduled in GSQ in the interval $[t_1, t_2]$, denoted by p_f^m, is:

$$EAT^{GSQ}(p_f^m, r_f) \leq t_1 + \epsilon + \frac{l_f^{max}}{r_f} \tag{6}$$

Since our objective is to determine a lower bound on the service received by a flow, we need to determine the last packet, denoted by p_f^n, that can be guaranteed to be served via GSQ in $[t_1, t_2]$. Since GSQ scheduler guarantees that packet p_f^n will depart by $EAT^{GSQ}(p_f^n, r_f) + \beta_f^n$, clearly p_f^n must be such that:

$$EAT^{GSQ}(p_f^n, r_f) + \beta_f^n \leq t_2 \tag{7}$$

Also, since p_f^n is the last packet that can be guaranteed to be served via GSQ by t_2, packet p_f^{n+1} should be such that it can not be guaranteed to depart by t_2. Hence,

$$t_2 \leq EAT^{GSQ}(p_f^n, r_f) + \frac{l_f^n}{r_f} + \beta_f^{n+1} \tag{8}$$

Since p_f^m begins service after t_1 and p_f^n departs by t_2, we get $W_f^{GSQ}(t_1, t_2) \geq \sum_{k=0}^{k=m-n} l_f^{m+k}$. Now, since flow is continuously backlogged in $[t_1, t_2]$,

$$EAT^{GSQ}(p_f^n, r_f) + \frac{l_f^n}{r_f} = EAT^{GSQ}(p_f^m, r_f) + \frac{\sum_{k=0}^{k=m-n-1} l_f^{m+k}}{r_f} + \frac{l_f^n}{r_f}$$

Hence, from (8) we get:

$$t_2 \leq EAT^{GSQ}(p_f^m, r_f) + \frac{\sum_{k=0}^{k=m-n} l_f^{m+k}}{r_f} + \beta_f^{n+1}$$

$$t_2 - \beta_f^{n+1} - EAT^{GSQ}(p_f^m, r_f) \leq \frac{W_f^{GSQ}(t_1, t_2)}{r_f}$$

Using (6) and $l_f^{n+1} \leq l_f^{max}$, we get

$$(t_2 - t_1) - (\beta_f^{max} + \epsilon) - \frac{l_f^{max}}{r_f} \leq \frac{W_f^{GSQ}(t_1, t_2)}{r_f}$$

Hence, the lemma follows. ■

Lemma 2 *If the GSQ scheduler ensures (3), then for any interval* $[t_1, t_2]$ *in which the flow is backlogged throughout the interval:*

$$W_f^{GSQ}(t_1, t_2) \leq r_f(t_2 - t_1) + (\beta_f^{max} + \epsilon) + l_f^{max}$$

Proof: Let packet p_f^j refer to the j^{th} packet of flow f serviced by GSQ. Since GSQ scheduler ensures that packet p_f^j departs by $EAT^{GSQ}(p_f^j, r_f) + \beta_f^j$. Hence, the first packet that finishes service within $[t_1, t_2]$, p_f^m, must be such that:

$$EAT^{GSQ}(p_f^m, r_f) + \beta_f^m \geq t_1$$

Since the rate controller ensures that at all time t, the expected arrival time of the packets in GSQ is at most $t + \epsilon$, we know that last packet that is scheduled in $[t_1, t_2]$, denoted by p_f^n, is such that:

$$EAT^{GSQ}(p_f^n, r_f) < t_2 + \epsilon$$

From the definition of expected arrival time, we know:

$$EAT^{GSQ}(p_f^n, r_f) - EAT^{GSQ}(p_f^m, r_f) \geq \sum_{k=0}^{k=m-n-1} \frac{l_f^{m+k}}{r_f}$$

$$\Rightarrow EAT^{GSQ}(p_f^n, r_f) - EAT^{GSQ}(p_f^m, r_f) + \frac{l_f^n}{r_f} \geq \sum_{k=0}^{k=m-n} \frac{l_f^{m+k}}{r_f}$$

Using the bounds on $EAT^{GSQ}(p_f^n, r_f)$ and $EAT^{GSQ}(p_f^m, r_f)$, we get:

$$(t_2 - t_1) + \epsilon + \beta_f^m + \frac{l_f^n}{r_f} \geq \sum_{k=0}^{k=m-n} \frac{l_f^{m+k}}{r_f}$$

Since packets p_f^m and p_f^n may or may not have been completely served within $[t_1, t_2]$, $\sum_{k=0}^{k=m-n} l_f^{m+k} \geq W_f^{GSQ}(t_1, t_2)$. Hence,

$$r_f(t_2 - t_1) + r_f(\beta_f^{max} + \epsilon) + l_f^{max} \geq W_f^{GSQ}(t_1, t_2)$$

■

Theorem 1 *If GSQ scheduler ensures (3), then for any interval* $[t_1, t_2]$ *such that flows* f *and* m *are backlogged during the entire interval, the difference in the service received by two flows in a FA server is given as:*

$$\left| \frac{W_f(t_1, t_2)}{r_f} - \frac{W_m(t_1, t_2)}{r_m} \right| \leq 2\left(\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}\right) + (\beta_f^{max} + \beta_m^{max}) + 2\epsilon$$

Proof: From Lemmas 1 and 2 we know bounds on $W_f^{GSQ}(t_1, t_2)$. Since $W_f(t_1, t_2) = W_f^{GSQ}(t_1, t_2) + W_f^{ASQ}(t_1, t_2)$, to derive bounds on $W_f(t_1, t_2)$, we need to derive bounds on $W_f^{ASQ}(t_1, t_2)$. To do so, observe that:

- Since a packet is removed from the ASQ only when it has been service either through ASQ or GSQ, whenever a flow is backlogged in the system, it is also backlogged in the ASQ.
- Whenever a packet is serviced from the GSQ, the start tag of the next packet in the ASQ is set to the start tag of the packet being removed. Hence Lemmas 1 and 2 of [15] continue to hold.

Since the flow is backlogged in in the interval $[t_1, t_2]$, from Lemmas 1 and 2 of [15] , we conclude:

$$r_f(v_2 - v_1) - l_f^{max} \leq W_f^{ASQ}(t_1, t_2) \leq r_f(v_2 - v_1) + l_f^{max} \tag{9}$$

where $v_2 = v(t_2)$ and $v_1 = v(t_1)$. From Lemmas 1 and 2, we get:

$$r_f(t_2 - t_1) - r_f(\beta_f^{max} + \epsilon) - l_f^{max} \leq W_f^{GSQ}(t_1, t_2) \leq r_f(t_2 - t_1) + r_f(\beta_f^{max} + \epsilon) + l_f^{max} \tag{10}$$

Since $W_f(t_1, t_2) = W_f^{GSQ}(t_1, t_2) + W_f^{ASQ}(t_1, t_2)$ and the unfairness is maximized when one flow receives the maximum possible service and the other receives the minimum, the theorem follows from (9) and (10).

■

This fairness guarantee will hold over variable rate servers also if the GSQ scheduling algorithm ensures that (3) holds when the server rate varies. In Section 6 we will present two GSQ scheduling algorithms that ensure that (3) holds over variable rate servers. Thus, FA contains algorithms that are fair over variable rate servers.

4 Deadline Guarantees

A FA server provides two types of deadline guarantees:

- It guarantees a deadline to a packet based on its *expected arrival time*. Specifically, it guarantees that:

$$L_{FA}(p_f^j) \leq EAT(p_f^j, r_f^j) + \beta_f^j \tag{11}$$

  where $L_{FA}(p_f^j)$ is the departure time of packet p_f^j in a FA server and β_f^j depends on l_f^j and the properties of the server as well as the other flows at the server. Such a guarantee has been referred to as *delay guarantee* and is used to provide various QoS guarantees regardless of the behaviour of the other flows in the network [14].

- It guarantees a deadline to a packet based on its arrival time and the departure time of the previous packet. Specifically, it guarantees that:

$$L_{FA}(p_f^j) \leq \max\{L_{SFQ}(p_f^{j-1}), A(p_f^j)\} + \beta_f^j \tag{12}$$

  Such a deadline guarantee, which is referred to as *delay-cum-throughput guarantee*, is used to improve upon the performance bounds determined from delay guarantee when the actual service received by a flow is better than that guaranteed by the server [15].

Theorems 2 and 3 establish the delay and delay-cum-throughput guarantees, respectively.

Theorem 2 *The departure time of packet p_f^j in a FA server, denoted by $L_{FA}(p_f^j)$, is given as:*

$$L_{FA}(p_f^j) \leq EAT(p_f^j, r_f) + \beta_f^j \tag{13}$$

Proof: Consider packet p_f^j. It may be served through ASQ or GSQ. Let us consider the two cases:

- Packet p_f^j is serviced through GSQ: Since GSQ ensures (3):

$$L_{FA}(p_f^j) \leq EAT^{GSQ}(p_f^j, r_f) + \beta_f^j \tag{14}$$

  It has been shown in [14] that:

$$EAT^{GSQ}(p_f^j, r_f) = EAT^{RC}(p_f^j, r_f) \tag{15}$$

  when the rate controller satisfies (2). Since $EAT^{RC}(p_f^j, r_f) \leq EAT(p_f^j, r_f)$, from (14) and (15) we conclude that (13) holds for packet p_f^j.

- Packet p_f^j is serviced through ASQ: Packet p_f^j starts service from ASQ at time t only if $t < EAT^{RC}(p_f^j, r_f)$. Hence, packet p_f^j departs by $t + \beta_f^j$. Since $EAT^{RC}(p_f^j, r_f) \leq EAT(p_f^j, r_f)$, we conclude that (13) holds for packet p_f^j.

■

Theorem 3 *In a FA server the departure time of packet p_f^j is given as:*

$$L_{FA}(p_f^j) \leq \max\{L_{FA}(p_f^{j-1}), A(p_f^j)\} + \frac{l_f^{max}}{r_f} + \beta_f^j + \epsilon \tag{16}$$

where $L_{FA}(p_f^0) = 0$.

Proof: Let t be the time at which packet p_f^j becomes the first flow f packet to be scheduled, i.e.,

$$t = \max\{L_{FA}(p_f^{j-1}), A(p_f^j)\} \tag{17}$$

Then, since the last packet to be scheduled through GSQ could have a maximum packet length of l_f^{max} and expected time at the rate controller of at most $t + \epsilon$, using (2) we conclude:

$$EAT^{RC}(p_f^j) \leq t + \epsilon + \frac{l_f^{max}}{r_f}$$

Packet p_f^j may be served through the GSQ or ASQ. Let the packet be served through GSQ. Using (15) we get:

$$EAT^{GSQ}(p_f^j) \leq t + \epsilon + \frac{l_f^{max}}{r_f} \tag{18}$$

Since GSQ server guarantees that p_f^j will depart by $EAT^{GSQ}(p_f^j) + \beta_f^j$:

$$L_{FA}(p_f^j) \leq EAT^{GSQ}(p_f^j) + \beta_f^j \tag{19}$$

Using (18) and then (17) to substitute for t yields (16). Since a packet can be served from ASQ only earlier than it is guaranteed to depart from GSQ, the theorem follows. ■

Since FA guarantees a deadline to a packet based on its expected arrival time, it belongs to the class of Guaranteed Rate scheduling algorithms defined in [13, 14]. Hence, the end-to-end delay guarantee of an internetwork of servers, some of which are FA servers, can be derived using the results presented in [13, 14]. Similarly, since the delay-cum-throughput guarantee of FA satisfies the assumptions of Theorem 7 of [15], the end-to-end delay-cum-throughput guarantee of an internetwork of servers, some of which are FA servers, can be derived using the results presented in [15].

5 Implementation Complexity

Fair Airport algorithms can be implemented in two ways: rate controllers as well as GSQ and ASQ scheduling algorithms can be implemented separately or by a single data structure. First consider the implementation in which all three components are implemented separately. To determine the implementation complexity in such a case, consider the complexity of rate controller and the scheduling algorithms for GSQ and ASQ:

- Rate controller: If a packet is to be made eligible in GSQ by time t, then as per (2), it can be made eligible in the interval $[t - \epsilon, t]$. Since ϵ is non-zero, the rate controllers for all the flows can be implemented by a single calendar queue with the clock tick value of ϵ [5]. The per packet complexity of calendar queue, and hence rate controller, is $O(1)$. Observe that if ϵ was 0, then implementing rate controller for all the flows would require a priority queue of eligible times of the first packets of the flows which would increase the rate controller complexity to $O(logQ)$.

- GSQ and ASQ scheduling algorithms: SFQ requires only the packets at the head of each flow queue to be sorted. Hence, it can be implemented by using a priority queue consisting only of the packets that are at the head of each flow queue. Consequently, the per packet computational complexity is $O(logQ)$ for the ASQ scheduling algorithm. The complexity of the GSQ scheduler, on the other hand, depends on the the choice of the algorithm. If the GSQ scheduler is a static priority scheduler, then its complexity would be $O(1)$ per packet and hence the aggregate complexity would be $O(logQ)$. If the GSQ scheduler is a dynamic priority scheduler that employs a priority queue consisting only of the packets that are at the head of each flow queue, then its complexity would be $O(logQ)$ and the aggregate complexity would continue to be $O(logQ)$. An important point to observe is that if GSQ scheduler employs a priority queue, then even though there are two priority queues, every packet is not inserted into the priority queue of both SFQ and GSQ servers. If n packets of a flow arrive in the flows busy period, a total of at most $n+1$ insertions occur in either of the priority queues. This is because: (1) every packet that is inserted into the priority queue of the GSQ server is served by it, and (2) due to rule 5 of the algorithm, if k packets of a flow are inserted into SFQ priority queue during the flows busy period, then at least $k-1$ packets are served by SFQ server. Consequently, at most $n+1$ insertions occur.

Hence, if the GSQ scheduler per packet complexity is either $O(1)$ or $O(logQ)$, the per packet complexity of Fair Airport algorithms is $O(logQ)$.

A disadvantage of implementing rate controllers and GSQ and ASQ scheduling algorithms separately is that $O(Q)$ computation may have to be performed within a single packet transmission time. To observe this, note that it is possible that the expected arrival time (at the rate controller) of all the packets that are at the head of each flow queue may be same. Consequently, Q packets may join the GSQ during a single packet transmission time. Hence, in the worst-case, $O(Q)$ computation may have to be performed in a single packet transmission time. This worst-case complexity can be reduced to $O(logQ)$ by implementing the rate controllers and the GSQ and ASQ scheduling algorithms using a single augmented binary search tree [1, 6, 17]. We illustrate this implementation using an augmented red-black tree which is a balanced binary search tree [6].

To implement FA using a red-black tree, we assume that the GSQ scheduling algorithm assigns a priority value to each packet and schedules packets in increasing order of priority. Furthermore, the priority value of a packet in GSQ can be determined when the packet becomes the first packet in its flow queue. Finally, we assume that GSQ scheduler schedules packets of a flow in FIFO order. Since SFQ (ASQ scheduler) also schedules packets of a flow in FIFO order, to determine the packet to be scheduled, it suffices to determine the flow to be scheduled. Hence, for each flow, let there be the following three variables:

- $\mathrm{eligibility\_time}$: This is the time at which the packet at the head of the flow queue will depart from the rate controller.
- $\mathrm{gsq\_priority}$: This is the priority value assigned by GSQ scheduler to the packet at the head of the flow queue.
- $\mathrm{asq\_priority}$: This is the start tag of the packet at the head of its flow queue.

Let all backlogged flows be inserted in a red-black tree with $\mathrm{eligibility\_time}$ as the search key. Operations such as deletion/insertion of a node and finding the node (flow) with the minimum $\mathrm{eligibility\_time}$ value can be performed in time $O(logQ)$ [6].

In FA at time t, the packet that has the smallest $\mathrm{gsq\_priority}$ value among the set of packets that have eligibility time at most t is scheduled. If such a packet does not exist, then the packet with the minimum $\mathrm{asq\_priority}$ is scheduled. To perform this search operation in time $O(logQ)$ we augment the nodes of the red-black tree with three fields:

- $\mathrm{min\_eligibility\_time\_node}$: This is the node with the minimum $\mathrm{eligibility\_time}$ value in the subtree rooted at the node.
- $\mathrm{min\_gsq\_priority\_node}$: This is the node with the minimum $\mathrm{gsq\_priority}$ value in the subtree rooted at the node.
- $\mathrm{min\_asq\_priority\_node}$: This is the node with the minimum $\mathrm{asq\_priority}$ value in the subtree rooted at the node.

Figure 3 presents the algorithm for determining the flow to be scheduled at time t that uses these additional fields (R is the root node of the tree and left and right denote the left and right child of a node, respectively). The essential idea in the algorithm is to traverse the tree structure to determine the node with the smallest eligibility time greater than t. During the tree traversal, we keep track of the node with minimum GSQ priority that has eligibility time at most t. The tree traversal is performed by the while loop in the algorithm and the else clause of the if statement in the while loop tracks the node with minimum GSQ priority value.

To determine the complexity of the algorithm, observe that the while loop in the algorithm is executed at most N times, where N is the depth of the tree. Since $N = O(logQ)$ for a red-black tree [6], we conclude that the complexity of the algorithm is $O(logQ)$. The additional fields in the red-black tree may increase the complexity of the insertion and deletion operations. However, since these fields for a node can be computed using only the fields of the node and its immediate children, from Theorem 15.1 of [6] we know that the complexity of the insertion/deletion operations continues to remain $O(logQ)$. Since within a single packet transmission time the number of packets that can arrive at the server, and consequently the number of insertion operations, is independent of the number of flows, we conclude that $O(logQ)$ computation is performed within a single packet transmission time.

To summarize, the per packet complexity of FA algorithm in either implementations is $O(logQ)$. However, the maximum computation that may have to be performed within a single packet transmission time is $O(Q)$ and $O(logQ)$ for separate and combined implementations, respectively.

```
Schedule(R,t)
{
    min_node= R.min_eligibility_time_node
    if ( t > min_node.eligibility_time )
        return R.min_asq_priority_node
    else
    {
        min = ∞
        min_node = null
        cur = R
        while (cur ≠ null)
        {
            if (cur.eligibility_time > t)
                cur= cur.left
            else
            {
                if ( cur.gsq_priority < min )
                {
                    min = cur.gsq_priority
                    min_node = cur
                }
                if ( cur.left ≠ null)
                {
                tmp = cur.left.min_gsq_priority_node
                if (tmp.gsq_priority < min)
                {
                    min = tmp.gsq_priority
                    min_node = tmp
                }
                }
                cur = cur.right
            }
        }
    return min_node
    }
}
```

Figure 3: An $O(logQ)$ algorithm to determine the packet to be scheduled in FA

6 Two Instantiations of Fair Airport

In this section, we instantiate the GSQ scheduling algorithm to Virtual Clock and Delay EDD to demonstrate that FA algorithms can achieve the delay guarantee of WFQ as well as separation of rate and delay allocation, respectively.

To demonstrate that Virtual Clock and Delay EDD as GSQ schedulers satisfy (3), we need to characterize the behaviour of servers. To ensure that the fairness and delay guarantees of FA servers hold over variable rate servers, we characterize the server as a Fluctuation Constrained (FC) server. A FC server has two parameters; average rate C (bits/s) and burstiness $\delta(C)$ (s). Intuitively, in a FC server the time taken to serve packets of aggregate length w in a busy period, can exceed the time taken in an equivalent constant rate server by at most $\delta(C)$. Formally,

Definition 1 A server is a Fluctuation Constrained (FC) server with parameters $(C, \delta(C))$, if the time taken to serve packets of aggregate length w in a busy period, denoted by $T(w)$, satisfies:

$$T(w) \leq \frac{w}{C} + \delta(C) \tag{20}$$

The delay guarantee of Virtual Clock and Delay EDD when the servers are FC are as follows.

- Virtual Clock: The Virtual Clock scheduling algorithm is defined as follows:
  1. Packet p_f^j on its arrival is stamped with its virtual clock value, denoted by $VC(p_f^j, r_f)$, which is defined as:
  $$VC(p_f^j, r_f) = EAT(p_f^j, r_f) + \frac{l_f^j}{r_f} \tag{21}$$
  2. Packets are scheduled in increasing order of virtual clock values.

  To derive the delay guarantee of Virtual Clock, define rate function of flow f, denoted by $R_f(t)$, as:

  $$R_f(t) = \begin{cases} r_f & \text{if } \exists j \ni \left(A(p_f^j) \leq t\right) \wedge \left(VC(p_f^{j-1}, r_f) < t \leq VC(p_f^j, r_f)\right) \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

  Theorem 4 proves the delay guarantee of a FC Virtual Clock server.

  Theorem 4 *If Q is the set of flows served by a FC VC server with parameters $(C, \delta(C))$ and $\sum_{n \in Q} R_n(t) \leq C$ for all t, then the departure time of packet p_f^j at the server, denoted by $L_{VC}(p_f^j)$, is given as:*

  $$L_{VC}(p_f^j) \leq EAT(p_f^j, r_f) + \frac{l_f^j}{r_f} + \frac{l_{max}}{C} + \delta(C) \tag{23}$$

  Proof: It is easily observed from (21), i.e., the definition of virtual clock value and (22) that the cumulative length of all flow n packets that arrive in interval $[t_1, t_2]$ and have virtual clock value no greater than t_2, denoted by $AP_n(t_1, t_2)$, is given as:

  $$AP_n(t_1, t_2) \leq \int_{t_1}^{t_2} R_n(t) dt$$

  Consider packet p_f^j served in the busy period beginning at time t_0. Let $t_2 = VC(p_f^j, r_f)$ and let t_1 be the largest time less than t_2 at which a packet that has virtual clock value greater than t_2 is scheduled in the busy period. If such a packet does not exist, set $t_1 = t_0^-$. Then, since packets are served in the increasing order of virtual clock values, all the packets scheduled in the interval $(t_1, L_{VC}(p_f^j)]$ arrive in the interval $(t_1, t_2]$ and have virtual clock value at most t_2. Hence, the aggregate length of packets scheduled in the interval $(t_1, L_{VC}(p_f^j)]$

is $\sum_{n \in Q} AP_n(t_1^+, t_2)$. Consequently, the aggregate length of packets served in the interval $[t_1, L_{VC}(p_f^j)]$ is at most $l_{max} + \sum_{n \in Q} AP_n(t_1^+, t_2)$. Since the server is FC, we get:

$$
\begin{aligned}
L_{VC}(p_f^j) &\leq t_1 + \frac{l_{max} + \sum_{n \in Q} AP_n(t_1^+, t_2)}{C} + \delta(C) \\
&\leq t_1 + \frac{\int_{t_1^+}^{t_2} \sum_{n \in Q} R_n(t)dt}{C} + \frac{l_{max}}{C} + \delta(C) \\
&\leq t_1 + \frac{C(t_2 - t_1)}{C} + \frac{l_{max}}{C} + \delta(C) \\
&\leq t_2 + \frac{l_{max}}{C} + \delta(C)
\end{aligned}
$$

Since $t_2 = VC(p_f^j, r_f) = EAT(p_f^j, r_f) + \frac{l_f^j}{r_f}$, the theorem follows. ■

We observe that this proof method can also be employed to derive the delay guarantee of generalized Virtual Clock scheduling algorithm, presented in [14] for FC servers.

Thus, Virtual Clock satisfies (3) with $\beta_f^j = \frac{l_f^j}{r_f} + \frac{l_{max}}{C} + \delta(C)$. Hence, using Theorem (2) we conclude that the departure time of packet in the FA server is given as:

$$L_{FA}(p_f^j) \leq EAT(p_f^j, r_f) + \frac{l_f^j}{r_f} + \frac{l_{max}}{C} + \delta(C)$$

This is the same delay guarantee as WFQ when $\delta(C) = 0$. Since the admission control equation requires only checking that aggregate bandwidth does not exceed the capacity, we conclude that a FA algorithm with Virtual Clock as the GSQ scheduler achieves the delay guarantee of WFQ with $O(1)$ complexity admission control algorithm.

- Delay EDD: Delay EDD is defined only when packet lengths of a flow do not vary. Hence, we assume that $l_f = l_f^j$. Then Delay-EDD is defined as follows:

  1. If d_f is the delay bound for flow f at server i, then packet p_f^j, on arrival, is stamped with a deadline, denoted by $D(p_f^j)$, which is computed as follows:

  $$D(p_f^j) = EAT(p_f^j, r_f) + d_f \tag{24}$$

  2. Packets are served in the increasing order of deadline.

The following theorem was proved in [15].

Theorem 5 *If Q is the set of flows serviced by the server and*

$$\forall t > 0: \quad \sum_{n \in Q} \max\{0, \left\lceil \frac{(t - d_n) r_n}{l_n} \right\rceil \frac{l_n}{C}\} \leq t \tag{25}$$

and the server is a $(C, \delta(C))$ Fluctuation Constrained server, then the time at which the transmission of packet p_f^j is completed in Delay EDD, denoted by $L_{EDD}(p_f^j)$ is:

$$L_{EDD}(p_f^j) \leq EAT(p_f^j, r_f) + d_f + \frac{l_{max}}{C} + \delta(C) \tag{26}$$

where l_{max} is the maximum length of packet served by the server.

Thus, Delay EDD satisfies (3) with $\beta_f^j = d_f + \frac{l_{max}}{C} + \delta(C)$. Hence, using Theorem (2) we conclude that the departure time of packet in the FA server is given as:

$$L_{FA}(p_f^j) \leq EAT(p_f^j, r_f) + d_f + \frac{l_{max}}{C} + \delta(C)$$

Since d_f can be chosen to be less than $\frac{l_f}{r_f}$, equal to $\frac{l_f}{r_f}$, or more than $\frac{l_f}{r_f}$, separation of rate and delay allocation in FA with Delay EDD as the GSQ scheduler is achieved.

Due to high computational complexity, it may not be feasible to employ (25) as the schedulability test. Hence, conditions stronger than (25) that have lower computational complexity have been developed in [26]. The delay guarantee holds under the stronger conditions as well. The computational complexity of the stronger conditions is $O(Q)$.

7 Salient Features of Fair Airport

Some of the salient features of the class of Fair Airport algorithms are as follows.

- As demonstrated in Section 6, it contains algorithms that have the delay guarantee of WFQ as well as those that achieve separation of rate and delay allocation.
- As demonstrated in Section 6, Virtual Clock and Delay EDD ensure (3) when the server is Fluctuation Constrained. Hence, FA contains scheduling algorithms that are fair over variable rate servers.
- It contains algorithms with different implementation complexity and performance guarantees. For example, a static priority GSQ scheduler is more efficient than a Delay EDD scheduler. However, whereas the former can support only predetermined set of delay vectors, the latter can support any [23].
- The concept of Fair Airport algorithm is general. The rate controller can be generalized as in [11] to enforce different traffic constraints. It can be shown that the generalized FA class continues to remain fair.
- The main concept of FA algorithms of using GSQ to provide deadline guarantees and ASQ to achieve desired allocation of residual bandwidth is general. By appropriately designing the ASQ scheduling algorithm, several policy objectives for residual bandwidth allocation can be achieved without effecting the deadline guarantees for a flow. To illustrate, ASQ scheduling algorithms can be designed to achieve:
  - *Prioritized fair allocation of residual bandwidth*: Prioritized fair allocation of residual bandwidth may be desirable to support high throughput flows in max-min fair networks [16]. In such a case, the set of flows served by a server are partitioned into various priority classes and the objective is to ensure that bandwidth is allocated fairly among flows in the same priority class. This objective can be achieved by an ASQ scheduling algorithm that provides non-preemptive static priority to the flows of higher priority classes and schedules packets belonging to the same priority class using SFQ (see Figure 4). It can be observed from the proof of the fairness guarantee of FA algorithms that the fairness guarantee continues to hold for any two flows that belong to the same priority class. Thus, prioritized fair allocation of residual bandwidth is achieved.
  - *Hierarchical allocation of residual bandwidth*: Hierarchical allocation of link bandwidth is desirable when several administrative domains or organizations share a link bandwidth [9, 15]. The requirements of hierarchical allocation are specified by a tree structure. The objective is to allocate the bandwidth of a node among its child nodes in proportion to the weights assigned to them. The objective of hierarchically allocating residual bandwidth can be achieved by employing Hierarchical SFQ presented in [15] as the ASQ scheduling algorithm. Since Hierarchical SFQ does not employ the length of a packet in determining its scheduling priority it meets the requirement of an ASQ scheduling algorithm outlined in Section 2 and, hence, can be used as an ASQ scheduling algorithm. Since ASQ server is a variable rate server and Hierarchical SFQ achieves hierarchical allocation of bandwidth even when server rate varies over time, we conclude that with Hierarchical SFQ as the ASQ scheduling algorithm, hierarchical allocation of residual bandwidth is achieved. We observe that this results into a scheduling algorithm with properties similar to

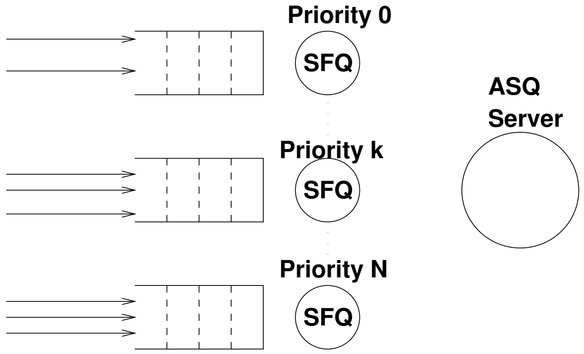

Figure 4: ASQ scheduler to achieve prioritized fair allocation of residual bandwidth

the algorithm presented in [21]. In particular, since the deadline guarantees for a flow are governed only by the GSQ scheduling algorithm, the deadline guarantees for a flow are independent of the depth of the hierarchy employed for residual bandwidth allocation.

The above examples are only illustrative in nature. ASQ scheduling algorithms that meet other policy objectives can be similarly designed.

8 Conclusions

In this paper, we designed a class of Fair Airport algorithms that combines SFQ with a non-work conserving algorithm in the RCSD class. We derived the fairness and deadline guarantees of FA servers. We demonstrated that by appropriately choosing an algorithm from the RCSD class, algorithms that either allocate only rate or achieve separation of rate and delay allocation and are fair over FC variable rate servers can be designed. This method led to the design of the *first* fair algorithm that achieves separation of rate and delay allocation. We also demonstrated that the FA class contains algorithms with different implementation complexity and performance guarantees and thus enables tradeoffs between the two. Finally, we demonstrated that the FA algorithms can be generalized to achieve objectives such as hierarchical and/or prioritized fair allocation of residual bandwidth.

References

[1] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. Fair On-Line Scheduling of a Dynamic Set of Tasks on a Single Resource. Technical Report TR-96-03, Department of Computer Sciences, The University of Texas at Austin, February 1996.

[2] J.C.R. Bennett and H. Zhang. Hierarchical Packet Fair Queuing Algorithms. In *Proceedings of SIGCOMM'96*, pages 143–156, August 1996.

[3] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case Fair Weighted Fair Queuing. In *Proceedings of INFOCOM'96*, pages 120–127, March 1996.

[4] J.C.R. Bennett and H. Zhang. Hierarchical Packet Fair Queuing Algorithms. *IEEE/ACM Transactions on Networking*, 5(5), October 1997.

[5] R. Brown. Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem. *Communications of the ACM*, 31(10):1220–1227, October 1988.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge MA, 1990.

[7] R.L. Cruz. Service Burstiness and Dynamic Burstiness Measures: A Framework. *Journal of High Speed Networks*, 2:105–127, 1992.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of ACM SIGCOMM'89*, pages 1–12, September 1989.

[9] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[10] L. Georgiadis, R. Guerin, V. Peris, and K.N. Sivarajan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. In *Proceedings of INFOCOM'96*, pages 102–110, March 1996.

[11] L. Georgiadis, R. Guerin, V. Peris, and K.N. Sivarajan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. In *Proceedings of INFOCOM'96*, pages 102–110, March 1996.

[12] P. Goyal. *Packet Scheduling Algorithms for Integrated Services Networks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 1997.

[13] P. Goyal, S. S. Lam, and H. M. Vin. Determining End-to-End Delay Bounds In Heterogeneous Networks. *ACM/Springer-Verlag Multimedia Systems Journal*, 5(3):157–163, May 1997. Also appeared in the Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video, Pages 287-298, April 1995.

[14] P. Goyal and H. M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. *IEEE/ACM Transactions on Networking*, 5(4):161–171, August 1997.

[15] P. Goyal, H. M. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. *IEEE/ACM Transactions on Networking*, 5(5), October 1997.

[16] Q. Ma, P. Steenkiste, and H. Zhang. Routing High-bandwidth Traffic in Max-min Fair Networks. In *Proceedings of SIGCOMM'96*, pages 206–217, August 1996.

[17] E. M. McCreight. Priority Search Trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[18] A.K. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[19] D. Stiliadis. *Traffic Scheduling in Packet-Switched Networks: Analysis, Design and Implementation*. PhD thesis, Department of Computer Science and Engineering, University of California at Santa Cruz, 1996.

[20] D. Stiliadis and A. Varma. Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet Switched Networks. In *Proceedings of SIGMETRICS'96*, May 1996.

[21] I. Stoica, H. Zhang, and T.S.E. Ng. A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. In *Proceedings of SIGCOMM'97*, September 1997.

[22] S. Suri, G. Varghese, and G. Chandramenon. Leap Forward Virtual Clock: A New Fair Queuing Scheme with Guaranteed Delays and Throughput fairness. In *Proceedings of INFOCOM'97*, April 1997.

[23] H. Zhang and D. Ferrari. Rate Controlled Static Priority Queueing. In *Proceedings of INFOCOM'93*, volume 2, pages 227–236, 1993.

[24] H. Zhang and D. Ferrari. Rate-controlled Service Disciplines. *Journal of High Speed Networks*, 3(4):389–412, 1994.

[25] H. Zhang and S. Keshav. Comparison of Rate-Based Service Disciplines. In *Proceedings of ACM SIGCOMM*, pages 113–121, August 1991.

[26] Q. Zheng and K. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-switching Networks. *IEEE Transactions on Communications*, 42(3):1096–1105, March 1994.