Efficient Fair-Queueing Algorithms for ATM and Packet Networks

Dimitrios Stiliadis
Anujan Varma

UCSC-CRL-95-59
December 1995

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

ABSTRACT

Although Weighted Fair Queueing is regarded as an ideal scheduling algorithm in terms of its delay and fairness properties, its computation complexity is asymptotically linear in the number of connections serviced by the scheduler, thus making its implementation prohibitively expensive in high-speed networks. An algorithm that combines the delay and fairness bounds of Weighted Fair Queueing with O(1) timestamp computations had remained elusive so far. In this paper, we present two novel scheduling algorithms that have O(1) complexity for timestamp computations, and provide the same bounds on end-to-end delay and buffer requirements as those of Weighted Fair Queueing. The first algorithm, *Frame-based Fair Queueing*, uses a framing mechanism to periodically re-calibrate a global variable tracking the progress of work in the system, limiting any short-term unfairness to within a frame-period. The second algorithm, *Starting Potential-based Fair Queueing* (SPFQ), performs the re-calibration at packet boundaries, resulting in a fairness bound that is equal to that of Weighted Fair Queueing, still maintaining the $O(1)$ timestamp computations. This improved fairness bound is achieved at the expense of a slightly higher implementation cost. Thus, SPFQ is attractive over FFQ in those applications where its improved fairness properties justify the additional implementation cost. The algorithms may be used in both general packet networks with variable packet sizes and in Asynchronous Transfer Mode (ATM) networks. Both algorithms are based on the general framework of *rate-proportional servers* (RPS) introduced in [11]. Details of hardware implementations of both algorithms are presented for use in an ATM network.

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

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

Contents

List of Figures

List of Tables

1 Introduction

Traffic scheduling algorithms are a necessary part of future integrated-services networks that will provide a broad range of Quality-of-Service (QoS) guarantees. These guarantees are usually in the form of bounds on end-to-end delay, bandwidth, delay jitter (variation in delay), packet loss rate, or a combination of these parameters. Several service disciplines such as Generalized Processor Sharing (GPS) and its packet-by-packet approximation (known as Weighted Fair Queueing or PGPS) [1, 2], VirtualClock [3], Delay-Earliest-Due-Date (Delay-EDD) [4], Weighted Round Robin [5] Deficit Round Robin [6], Hierarchical-Round-Robin (HRR) [7], and Stop-and-Go queueing [8] have been proposed in the literature for solving this problem (for a survey see [9].)

The design of a traffic scheduling algorithm involves an inevitable tradeoff among its delay, complexity of implementation, and fairness. Among the three, the delay and implementation complexity are clearly the most important criteria for the selection of an algorithm for use in a real system. While the fairness properties of the algorithm affect only the short-term distribution of service offered to the connections sharing the link, a larger delay bound implies increased burstiness of the session at the output of the scheduler, thus increasing the amount of buffering needed in the switches to avoid packet losses. In addition to minimizing the end-to-end delay in a network of servers, the delay behavior of an ideal algorithm must include (i) insensitivity to traffic patterns of other sessions (isolation), (ii) delay bounds that are independent of the number of sessions sharing the outgoing link, and (iii) ability to control the delay bound of a session without depending on the internal parameters of the scheduler [10, 11].

As was discussed in the first part of this work [11], based only on the end-to-end delay bounds and fairness properties, Generalized-Processor-Sharing (GPS) is an ideal scheduling discipline [1]. The GPS system is based on a fluid model where the packets are assumed to be infinitely divisible and multiple sessions may transmit traffic through the outgoing link simultaneously at different rates. A packet-by-packet version of the algorithm, known as PGPS or Weighted Fair Queueing (WFQ), is defined in terms of the GPS system [1, 2]. That is, a GPS system is simulated in parallel with the packet-by-packet system in order to identify the set of connections that are backlogged at each instant. This information is used to compute a timestamp for each arriving packet, indicating the time at which it would depart the system under GPS. Packets are then transmitted in increasing order of their timestamps. A serious problem with this approach is its computational complexity: A maximum of V events may be triggered in the GPS simulator during the transmission of one packet. Thus, the process overhead for completing a scheduling decision is $O(V)$.

In order to reduce its complexity, an approximate implementation of GPS multiplexing was proposed by Davin and Heybey [12] and later analyzed by Golestani [13] under the name *Self-Clocked Fair Queueing* (SCFQ). In this implementation, the timestamp of an arriving packet is computed based on the timestamp of the packet currently in service. This approach reduces the complexity of the algorithm greatly. However, the price paid is the reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of sessions that share the outgoing link [10]. Thus, the worst-case delay of a session can no longer be controlled just by controlling its reservation, as is possible in Weighted Fair Queueing (WFQ). The higher end-to-end delay also affects the burstiness of sessions within the network, increasing the buffer requirements. The VirtualClock scheduling algorithm, on the other hand provides the same end-to-end delay and burstiness bounds as WFQ with a simple timestamp computation algorithm, but the price paid is in terms of fairness.

An algorithm that combines the delay and fairness bounds of Weighted Fair Queueing with O(1) timestamp computations had remained elusive so far. In this paper, we present two novel scheduling algorithms that have O(1) complexity for timestamp computations, and provide the same bounds on end-to-end delay and buffer requirements as those of Weighted Fair Queueing. These algorithms are based on the analytical framework of *rate-proportional servers* (RPS) presented in [11].

Schedulers in the RPS class use the concept of *potential* to track the state of the system. Each connection is associated with a a *connection potential* that keeps track of the amount of

normalized service actually received by the connection during the current system-busy period,* plus any normalized service it missed during the period when it was not backlogged. The connection potential is a non-decreasing function of time during a system-busy period. The basic system is defined in terms of a fluid model, and the corresponding packet-by-packet server is obtained by computing a timestamp for each arriving packet that represents the value of the connection potential at the instant the last bit of the packet leaves the the fluid system, and scheduling the packets in the order of increasing timestamps.

We assume that V connections share the outgoing link of the scheduler, a rate ρ_i is allocated to each connection i, and that the total bandwidth assigned to the connections does not exceed the link capacity r. That is,

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

When connection i is backlogged, its potential increases exactly by the normalized service it receives. That is, if $P_i(t)$ denotes the potential of connection i at time t, then, during any interval $(\tau, t]$ within a backlogged period for session i,

$$P_i(t) - P_i(\tau) = \frac{W_i(\tau, t)}{\rho_i},$$

where $W_i(\tau, t)$ denotes the amount of service received by session i during the interval $(\tau, t]$.

The basic objective of a rate-proportional server is to *equalize* the potential of all backlogged connections at each instant. This is achieved in a fluid server as follows: At any instant t, the scheduler services only the subset of connections with the minimum potential, and each connection in this subset receives service in proportion to its reserved rate ρ_i. Thus, the scheduler can be seen to increase the potentials of the connections in this subset at the same rate. At the time that a connection becomes backlogged, its potential is updated based on a *system potential* function that keeps track of the progress of the total work done by the scheduler. The system potential $P(t)$ is a non-decreasing function of time. When an idle session i becomes backlogged at time t, its potential $P_i(t)$ is set as

$$P_i(t) = \max(P_i(t-), P(t)),$$

to account for the service it missed. Schedulers use different functions to maintain the system potential, giving rise to widely different delay- and fairness-behaviors. In general, the system potential at time t can be defined as a non-decreasing function of the potentials of the individual connections before time t, and the real time t.

$$P(t) = \mathcal{F}(P_1(t-), P_2(t-), \ldots, P_V(t-), t). \tag{1.1}$$

Ideally, the rate of increase of the system potential at each instant should match the rate of increase of the potential of a connection currently being serviced by the scheduler. In practice, however, a much more relaxed definition of the system potential function is adequate.

The system potential function in a rate-proportional server must satisfy two fundamental properties to provide performance bounds comparable to that of a WFQ scheduler. First, during any interval $(t_1, t_2]$ within a system-busy period, the system potential function must be increased with a rate of at least one, that is,

$$P(t_2) - P(t_1) \geq (t_2 - t_1). \tag{1.2}$$

Second, the system potential function must never exceed the potential of any backlogged connection. In addition, if the difference between the system potential and the potential of every backlogged connection is bounded, then the server is fair and its fairness can be estimated in terms of this difference [11].

*A system-busy period is defined as a period during which the server is continuously transmitting packets.

The definition of rate-proportional servers does not specify the exact method of maintaining the system potential function. This enables a wide range of algorithms to be defined, all with the same delay bound as that of WFQ, but with different fairness characteristics. For example, both GPS and the fluid-model equivalent of VirtualClock are rate-proportional servers, but their fairness bounds and implementation complexities occupy two extremes in the RPS framework.

The fundamental difficulty in designing a practical rate-proportional server is the need to maintain the system potential function. Tracking the global state of the system precisely requires simulating the corresponding fluid-model RPS in parallel with the packet-by-packet system. However, the definition of the system potential function allows considerable flexibility in approximating the global state of the system. This flexibility is exploited in this paper in the design of two practical scheduling algorithms — *Frame-based Fair Queueing* (FFQ) and *Starting Potential-based Fair Queueing* (SPFQ). Both algorithms maintain the system potential function only as an approximation of the actual global state in the fluid model, but re-calibrate the system potential periodically to correct any discrepancies. This re-calibration is key to providing bounded fairness, where fairness is defined as the maximum difference in normalized service received by any two backlogged sessions during any arbitrary interval. In the Frame-based Fair Queueing (FFQ) algorithm this re-calibration is done at frame boundaries, while in Starting Potential-based Fair Queueing (SPFQ) the re-calibration occurs at packet boundaries. This gives rise to two algorithms with the same delay bound, but with slightly different fairness properties. Both algorithms, however, provide bounded unfairness and $O(1)$ timestamp computations. It is interesting to note that the maximum short-term unfairness of SPFQ is actually no worse than that of Weighted Fair Queueing.

Both FFQ and SPFQ are timestamp-based algorithms. However, FFQ uses a framing approach similar to that used in frame-based schedulers to re-calibrate the system potential periodically. This makes the fairness of the algorithm depend on the frame size chosen by the implementation. SPFQ avoids this sensitivity to the frame size by re-calibrating the system potential at the end of transmission of every packet. In comparison to FFQ, SPFQ requires more state information to be maintained, resulting in a more complex hardware implementation; however, this increased hardware complexity does not affect its asymptotic time-complexity. Thus, SPFQ is attractive over FFQ in applications where its improved fairness properties justify the additional hardware cost.

The rest of this paper is organized as follows: In Section 2, we define a general methodology for estimating and updating the system potential. In Section 3 we present Frame-based Fair Queueing in terms of a hypothetical fluid-model, and subsequently extend to a packet-by-packet model. We also analyze the fairness properties of the algorithm. In Section 4 we develop and analyze the Starting Potential-based Fair Queueing (SPFQ) algorithm. In Section 5 we provide simulation results on the end-to-end delays seen by sessions in various network configurations with FFQ and SPFQ, and compare the performance of these algorithms with both Weighted Fair Queueing and Self-Clocked Fair Queueing. Some concluding remarks are presented in Section 6. In Appendix A, we discuss illustrative implementations of the algorithms for ATM networks. Finally, the proofs of some of the lemmas and theorems can be found in Appendix B.

2 Methodology for Maintaining System Potential

The basic difficulty in the design of a rate-proportional server is in maintaining the system potential function. Since the degree of short-term unfairness of the algorithm depends on the difference between the system potential and the potentials of backlogged connections at any time, the fairness of the algorithm is determined by the choice of the system potential function. In this section we will present a general methodology for updating the system potential function. The resulting algorithms will be referred as *Fair Rate-Proportional Schedulers* (FRPS). Formally, we can define a FRPS as a rate-proportional server as follows:

Definition 1: *Let $P(t)$ denote the system potential in an RPS and $P_i(t)$ the potential of connection i at time t. The scheduling algorithm is a Fair Rate-Proportional Server if and only if a finite constant $\Delta P \geq 0$ can be found such that*

$$P(t) \geq P_i(t) - \Delta P, \quad \textit{for any } i \in B(t);$$

where $B(t)$ is the set of backlogged connections at time t.

The above constraint can be satisfied by the use of a *re-calibration mechanism* periodically to bound the maximum difference of the system potential function from the potential of a backlogged connection.

Let us first introduce some notations. We assume that a rate ρ_i is allocated to connection i. Let r be the bandwidth capacity of the outgoing link; then, $r_i = \rho_i/r$ is the fraction of the link rate allocated to connection i. As in the previous section, let $P_i(t)$ represent the potential of connection i at time t and $P(t)$ the corresponding value of system potential.

The fluid version of a Fair Rate-Proportional Server follows all the conditions in Definition 5 of [11]. That is, at each instant, the scheduler services only the set of backlogged connections with the minimum potential and connections in this set are serviced at rates proportional to their reservations.

In an idealized fluid server it is possible to update the system potential at any instant of time. However, in a packet-by-packet server it is desirable to update the system potential only when a packet departs from the system. In order to simplify the implementation of the algorithm we will define a general method for updating the system potential that will be based on only information extracted from the packet-by-packet implementation of the algorithm.

We first define a function $S^P(t)$ that we will call the *base potential*. $S^P(t)$ is a non-decreasing function with the following properties:

1. Let $B(t)$ represent the set of connections that are backlogged at time t in the system. Then,

$$S^P(t) \leq P_i(t), \quad \forall i \in B(t). \tag{2.1}$$

2. A finite constant $\Delta P \geq 0$ can be found such that

$$S^P(t) \geq P_i(t) - \Delta P, \quad \forall i \in B(t). \tag{2.2}$$

That is, the base potential function can be any non-decreasing function whose value is never higher than the potential of any backlogged connection at that instant. It is easy to see that such a function can be used as the reference for updating the system potential periodically, since it satisfies condition 3 of Definition 5 of [11]. Since the above definition of the base potential function does not specify how to construct such a function, considerable flexibility exists in its choice. Assuming the base potential is used to re-calibrate the system potential periodically, and that the interval between re-calibrations is bounded, the condition in Eq. (2.2) is sufficient to achieve bounded fairness, that is, for the algorithm to belong to the FRPS class. Different choices of the function $S^P(t)$ result in algorithms with different implementation complexities, but all with bounded fairness. In the later sections, we will show two distinct ways to construct the base-potential function $S^P(t)$, resulting in the FFQ and SPFQ algorithms.

Any rate-proportional server can achieve bounded fairness by periodically re-calibrating the system potential using the base potential function. Thus, in general, a Fair Rate-Proportional Server can be constructed by maintaining the system potential function $P(t)$ as follows.

Definition 2: *Let the system-potential function in an RPS be defined as follows: When the system is not busy the system potential function is equal to zero. During a system-busy period, the function $P(t)$ is a piecewise linear function of time t. Let τ_0 be the beginning of the current system-busy period. Then,*

1. *At times $\tau_1, \tau_2, \ldots, \tau_k$, with $\tau_0 < \tau_1 < \ldots < \tau_k$, a re-calibration is performed by updating $P(t)$ to the base potential at that instant, if the system potential is lower than the base potential. That is,*

$$P(\tau_j) = \max(P(\tau_j-), S^P(\tau_j)), \tag{2.3}$$

   where τ_j- denotes the instant of time just before the update.

2. *At any time time t between updates, the system potential increases linearly with time. That is,*

$$P(t) = P(\tau_j) + (t - \tau_j), \quad \tau_j \leq t < \tau_{j+1}. \tag{2.4}$$

3. The interval between successive re-calibrations is bounded, that is,

$$\tau_{j+1} - \tau_j \leq \Delta T, \quad \textit{for some finite } \Delta T.$$

The update in Eq. (2.3) enables us to bound the difference between the system potential and the potentials of backlogged connections. Without such an update mechanism, the system potential may diverge from the connection potentials by an arbitrary amount, causing the unfairness of the algorithm to be unbounded. The updates at time instants $\tau_1, \tau_2, \ldots \tau_k$ are designed to bring the system potential to a value closer to the connection potentials. This value is estimated through the base potential function $S^P(t)$.

Before proceeding further, it is important to note that the fairness of any rate-proportional server as defined above depends on two factors:

1. The choice of the base potential function $S^P(t)$.
2. The frequency of re-calibrations, that is, the choice of the update instants $\tau_1, \tau_2, \ldots \tau_k$.

In order to design an efficient packet-by-packet version of the algorithm, we can only perform these re-calibration steps at the times that a packet finishes its service. Thus, the frequency of re-calibration is upper-bounded by the departure rate of packets from the server.

We now proceed to show that the system-potential function in Definition 2 results in a Fair Rate-Proportional Server. We first need to show that the system-potential function satisfies the two key properties in the definition of a rate-proportional server [11]. The following two lemmas prove that the system potential increases at least at the rate of real time, and that it never increases above the potential of a backlogged connection.

Lemma 1: *If the system-potential function is maintained as described by Definition 2, then, for any interval $(t_1, t_2]$ during a system-busy period,*

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

Proof: Assume that the system-busy period under observation started at time 0. If no re-calibrations occurred during the interval $(t_1, t_2]$, then the lemma is true by Eq. (2.4). Now consider the case when one or more re-calibrations occurred during the interval $(t_1, t_2]$. Let $\tau_1, \tau_2, \ldots, \tau_k$ be the instants in this interval just after an update to $P(t)$, and $\tau_1-, \tau_2-, \ldots, \tau_k-$ the corresponding instants just before the update, with $\tau_1 < \tau_2 < \ldots < \tau_k$. Then, by equations (2.3) and (2.4),

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

Proceeding similarly,

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

This concludes the proof of Lemma 1. □

Lemma 2: *If the system potential function is maintained as in Definition 2, then, at any time t,*

$$P(t) \leq P_i(t), \quad \forall i \in B(t). \tag{2.5}$$

Proof: The proof is by contradiction. Since $P(0) = P_i(0)$, Eq. (2.5) is satisfied trivially at time 0. Let t be the earliest time during a system-busy period at which $P(t) > P_i(t)$ for some i. Then, let Δt be the smallest interval such that $P(t - \Delta t) \leq P_i(t - \Delta t)$. Session i must be continuously backlogged in the server during the interval $(t - \Delta t, t]$. We need to consider two cases:

Case 1: No re-calibration of the system potential occurred during the interval $(t - \Delta t, t]$. Then i is a session with minimum potential during the interval $(t - \Delta t, t]$. Therefore, it is serviced with rate at least ρ_i during this interval. Thus, the potential of session i at t must be at least

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

Thus, the result is true by contradiction.

Case 2: A re-calibration occurred at time t. Let $t-$ denote the instant just before the update to $P(t)$ and t the instant just after the update. Then,

$$P_i(t-) \geq S^P(t-). \tag{2.6}$$

The new system potential after the update is given by

$$P(t) = \max(P(t-), S^P(t-)). \tag{2.7}$$

Using equations (2.6) and (2.7), as well as the fact that $P_i(t-) \geq P(t-)$,

$$P_i(t) = P_i(t-) \geq P(t). \tag{2.8}$$

□

Theorem 1: *A rate-proportional server with its system-potential function $P(t)$ defined as per Definition 2 is a Fair Rate-Proportional Server.*

Proof: Lemmas 1 and 2 prove that the system potential function satisfies the two main conditions imposed by the definition of a rate-proportional server. In addition, if the re-calibrations are performed at finite intervals, by Eq. (2.2), the difference between the system potential and the potential of any backlogged connection will be bounded. Thus, the algorithm is a Fair Rate-Proportional Server. □

3 Frame-based Fair Queueing

Using the methodology we described in the previous section, we can define several algorithms by choosing different base-potential functions and re-calibration intervals. A simple approach is to perform the re-calibration periodically, with a maximum period equal to an internal parameter of the algorithm that we call the *frame size* F. This approach results in the definition of Frame-based Fair Queueing (FFQ). In this section, we will describe the FFQ algorithm and analyze its properties.

We will first define the parameters of the algorithm with respect to a fluid system and subsequently extend them to the packet-by-packet system. We define the frame size parameter such that exactly F bits can be transmitted during a *frame period* T. That is,

$$T = \frac{F}{r}.$$

We define ϕ_i as

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

ϕ_i denotes the maximum amount of session i traffic that can be serviced during one frame. When a connection remains backlogged, its potential increases by the normalized service offered to it. Thus, when ϕ_i bits are serviced from connection i, its potential will increase by

$$\frac{\phi_i}{\rho_i} = T.$$

We impose one more restriction on the value of ϕ_i, that the largest packet of a connection can be transmitted during a frame period. That is, if L_i is the maximum packet size for connection i, then

$$L_i \leq \phi_i. \tag{3.1}$$

We will refer to the process of re-calibrating the system-potential in a FFQ server as a *frame update* operation. Each frame-update operation marks the beginning of a new frame in the system. If all the connections are continuously backlogged, frame updates can be performed in a fluid server exactly at intervals of the frame period T. The updates will occur earlier, however, if the arrivals from some of the sessions are below their respective reservations, causing the potentials of backlogged connections to rise faster. Thus, in the fluid server, the kth frame update can be performed when the potentials of backlogged connections reach the value kT. Note that all connections reach the potential of kT at the same time in the fluid server. In a packet-by-packet server, however, this is not the case. Therefore, in order to avoid simulation of the fluid system to determine the frame-update instants, we define the frame-update instants in a more relaxed manner as follows: Let τ_{k-1} denote the last time a frame update occurred. The next update is performed when both of the following conditions hold:

1. The potentials of all backlogged connections in the fluid server belong in the next frame. That is,
$$P_i(t) \geq kT, \quad \forall i \in B(t), \tag{3.2}$$
where $B(t)$ is the set of connections currently backlogged.
2. $P_i(t) < (k+1)T, \quad i = 1, 2, \ldots, V.$

Note that the above conditions may be satisfied during a *window* of of time. Performing the next frame update at any time during this window will result in a valid algorithm. Let us assume that we decide to update the frame at time τ_k. Then, at time τ_k we set

$$P(\tau_k) \leftarrow \max(P(\tau_k), kT). \tag{3.3}$$

Since analysis of the packet-by-packet FFQ server requires reference to the corresponding fluid server, we define the frame update instants τ_k to be identical in both servers. These update instants can then be determined from only information available in the packet-by-packet server, so as to fall in the window defined by conditions 1 and 2 above. This relaxed definition allows re-calibrations to occur only when a packet finishes or starts service in the packet-by-packet server. Note that the duration of a frame (that is the interval between successive frame-updates), never exceeds $2T$.

We can now define the base-potential function $S^P(t)$ for FFQ as follows: $S^P(t)$ is a step function whose value is zero when the server is idle and increases by T at every frame-update instant. Thus, at the kth frame-update instant τ_k, $S^P(t)$ assumes a value of $k \cdot T$.

Figure 3.1 illustrates the base-potential and system-potential functions in a fluid FFQ server, where τ_1 – τ_4 denote the instants at which the frame updates occur. The system potential grows linearly with time between the update instants. Note that the kth frame update can be performed any time during a window when the potential of a backlogged connection is between kT and $(k+1)T$.

In a packet-by-packet server, the frame updates can only be performed at packet boundaries. We now show that an update instant can be found in a packet-by-packet server based only on the timestamps of the queued packets. Recall that the timestamp of a packet denotes the potential of the corresponding connection at the instant the packet completes its service in the fluid system. We make use of the following lemma from [11] to establish a relationship between the potentials of a connection in the fluid and packet servers.

Lemma 3: *Let $(0,t]$ be a server-busy period in the fluid server. Let i be a session backlogged in the fluid server at time t such that i received more service in the packet-by-packet server in the interval $(0,t]$. Then there is another session j, with $P_j(t) \leq P_i(t)$ that received more service in the fluid server than in the packet-by-packet server during the interval $(0,t]$.*

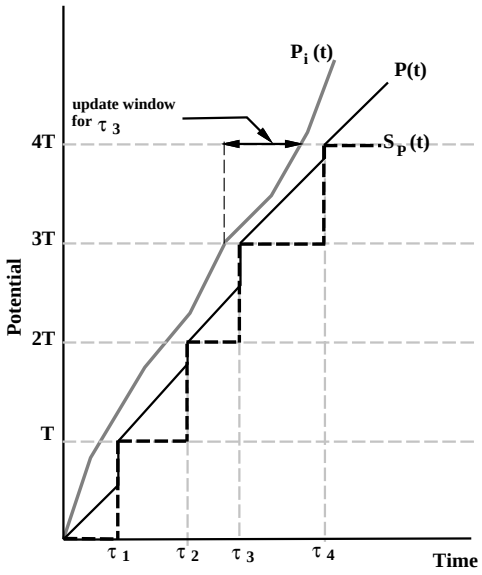

Figure 3.1: Behavior of the base-potential and system-potential functions in a fluid FFQ server. $P_i(t)$ represents the potential of all backlogged connections, $P(t)$ the system potential, and $S_P(t)$ the base potential. Re-calibration of the system potential occurs at frame-update points τ_1, τ_2, τ_3, and τ_4. The frame update points can fall anywhere within a window where the individual connection ptentials lie between kT and $(k+1)T$.

This lemma enables us to find a relationship between the potentials of the backlogged connections in the fluid server and the timestamps of the backlogged connections in the packet-by-packet server. We can now prove the following lemma, that will allow us to perform frame updates in FFQ by using only information extracted from the packet-by-packet system. Let us first define the starting potential s_i^j of a packet j of connection i as the potential of the connection when packet j starts being serviced in the corresponding fluid server, and let $S_i(t)$ denote the starting potential of the first packet in the queue of connection i at time t. Let $\hat{B}(t)$ denote the set of backlogged sessions at time t in the packet-by-packet server.

Lemma 4: *Assume that at time t, for each backlogged session in the packet-by-packet system, the starting potential of its first packet belongs in the next frame. That is, if τ_k was the last instant at which a frame update occurred,*

$$S_i(t) \geq (k+1)T, \quad \forall i \in \hat{B}(t).$$

Then, the potential of each backlogged session in the fluid server at time t is also greater than or equal to $(k+1)T$.

Proof: We will prove the lemma by contradiction. Let us denote with i the connection with the minimum potential in the fluid server and let us assume that the potential of connection i is less than $(k+1)T$. Connection i has received until time t more service in the packet-by-packet server than in the fluid-server. By Lemma 3, there is another connection k with potential $P_k(t) \leq P_i(t)$ that has received less service in the packet-by-packet server than in the fluid-server. Let s'_k be the starting potential of the packet that is being serviced in the fluid-server at time t from connection k. Then $s'_k \leq P_k(t)$ and thus $s'_k \leq (k+1)T$. Notice also that this packet has not yet been serviced in the packet-by-packet server. This is a contradiction. □

The significance of Lemma 4 is that we can determine a valid update time for the frame by using only information extracted from the packet-by-packet server. The scheduler can keep track of all the connections that are backlogged and have packets with starting potential in the next frame. When the starting potentials of the packets at the head of the queues of all backlogged sessions have

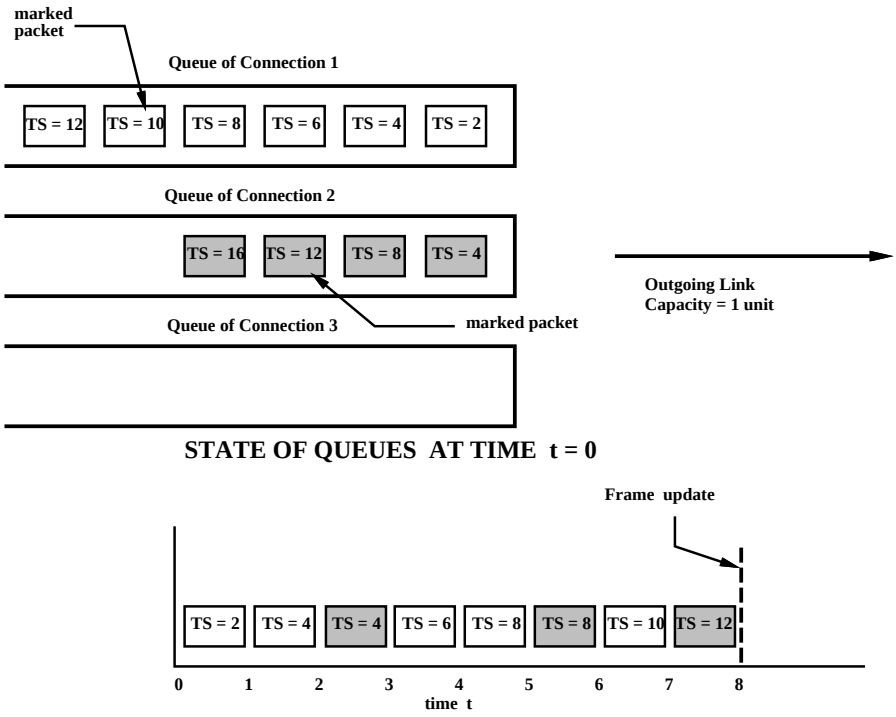

Figure 3.2: Example of operation of the packet-by-packet FFQ server. The frame can be updated at time 8 when the starting potentials of the packets of all backlogged connections in the packet-by-packet server have crossed the frame boundary.

crossed the frame boundary, we know that the potentials of the connections in the fluid-system have also crossed the frame boundary. Therefore, the crossing time of the last connection is a valid time to update the frame and the system potential function.

This can be seen better by an example. Let us assume that the frame size F is set to 10 cells and that rate of the server is set equal to 1. Assume that connection 1 has reserved half of the output link bandwidth and each of connections 2 and 3 has reserved 25% of the output link bandwidth. Let us also assume that the system was idle before time 0. At time 0, connections 1 and 2 send a large number of packets to the scheduler, while the queue of connection 3 remains empty. At time 0, the system potential and the individual connection potentials are zero. In a fluid server, the packets of connections 1 and 2 will be serviced in proportion to their reservations. Thus, the rate of service for the connection 1 and 2 will be $0.5/(0.5+0.25) = 2/3$ and $0.25/0.75 = 1/3$, respectively. Assuming connection 3 remains idle, the potentials of connections 1 and 2 will reach the value of $T = \frac{F}{1} = 10$ at the same time. Let τ be this time. Since both connections are backlogged, their potentials are being increased by the normalized service offered to them. Thus,

$$\frac{W_1(0,\tau)}{0.5} = \frac{W_2(0,\tau)}{0.25} = 10. \tag{3.4}$$

Thus, we can determine τ from

$$(2/3)(\tau - 0)/0.5 = 10, \tag{3.5}$$

which yields $\tau_1 = 7.57$. This is the beginning of the window in which the first frame-update can occur.

Now considering the packet-by-packet server, the packets are timestamped with the potential of the connection at the time they finish service in the fluid server. Let us assume that all packets have a size of 1. In Figure 3.2 we show the timestamps of the packets and the sequence with which

```
    Calculate current value of system potential.
    Let t be the current time and t_s the time when
    the packet currently in service started its transmission.
1.  temp ← P + (t − t_s)/F

    Calculate the starting potential of the new packet
2.  SP(i,k) ← max(TS(i,k−1), temp)

    Calculate timestamp of packet
3.  TS(i,k) ← SP(i,k) + length(i,k)/ρ_i

    Check if packet crosses a frame boundary
4.  n1 ← int(SP(i,k)); n2 ← int(TS(i,k))
5.  if (n1 < n2) then (if finishing potential is in next frame)
6.      B[n1] ← B[n1] + 1 (increment counter);
7.      mark packet
8.  endif
```

Figure 3.3: Algorithm executed on the arrival of a packet.

they are serviced. Note that the starting potential of a packet in this example is identical to the timestamp of the packet ahead of it. At time 8, all the packets with starting potentials in the first frame have already been transmitted. At this time, the potentials of all backlogged connections have a value greater than or equal to 10 in the fluid server. Thus, at this time we can update the system potential to 10 without violating the properties of the system potential function.

We can now describe the packet-by-packet version of the frame-based fair queueing algorithm. Without loss of generality we can assume that the service rate of the server is 1. Thus, the time to transmit F bits is also equal to F. A fraction r_i of the output link bandwidth is allocated to connection i and therefore $\phi_i = F \times r_i$ bits can be sent from connection i during a frame. As in the fluid version, we require that the maximum packet size be less than ϕ_i, so that a single packet can be transmitted within one frame.

On the arrival of a packet, the algorithm in Figure 3.3 is executed to calculate the timestamp associated with the packet. The variable P keeps track of the system potential. P is a floating-point number with two parts — the integer part representing the current frame number and the fractional part representing the elapsed real time since the last frame update. On arrival of a packet, the current system potential is estimated. Since the variable P is updated only at the end of transmission of each packet, the current system potential is obtained by adding to P the elapsed real-time since the current packet in service started transmission. The starting potential of the newly-arrived packet is then computed as the maximum of the finishing potential of the previous packet from the same session and the system potential. The packet is then timestamped with its finishing potential, computed from knowledge of its length and the reserved rate. If the starting and finishing potentials of the packet belong to different frames, the current packet is one that crosses over to the next frame. Therefore the packet is marked to indicate that this is the first packet of the session to cross over to the next frame. In addition, a counter is incremented to keep track of the number of connections that have crossed over into the new frame. The algorithm maintains one counter per frame to keep track of the number of sessions whose packets cross into the next frame. Later, when a marked packet is scheduled for transmission, the corresponding counter is decremented; when the counter reaches zero, the potentials of all the backlogged connections have crossed over to the next frame, and a frame update can be performed.

The array of counters B is used to count the number of connections that have packets with a starting potential in each frame. Although an infinite number of frames may need to be serviced, in practice the number of distinct frames in which the potentials of queued packets can fall into is

```
     Increase system potential by the transmission time of the packet just completed, say j.
1.   P ← P + length(j)/F

     Find timestamp of next packet for transmission
2.   TS_min ← min_{i∈B}(TS(i))

     Determine the corresponding frame number.
3.   F_min ← int(TS_min)

     Perform frame update operation if required
4.   if (packet j was marked) then
5.        B[current-frame] ← B[current-frame] − 1
6.   end if
7.   if (B[current-frame] = 0 and F_min > current-frame) then
8.        current-frame ← current-frame + 1
9.        P ← max(current-frame, P)
10.  end if

     Store starting time of transmission of next packet in t_s
11.  t_s ← current time
12.  Retrieve packet from head of queue and transmit
```

Figure 3.4: Algorithm executed on the departure of a packet.

limited by the buffer size allocated to the connections. Thus, if b_i denotes the buffer space allocated to connection i, the size of the array B can be limited to

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

If M is rounded up to the nearest power of 2, then the array can be addressed with the $\lceil \log_2 M \rceil$ least significant bits of the current frame number. The number of counters can further be reduced to *three* if steps 4–8 of the algorithm are executed only when a packet reaches the head of the queue of the corresponding session.

When a packet finishes transmission, the algorithm in Figure 3.4 is executed to update the state of the system. The system potential is first increased by the transmission time of the packet just serviced. The packet with the minimum timestamp is then selected for transmission. This selection can be performed efficiently by maintaining the packets in a priority list structure, such as a heap. The variable *current-frame* keeps track of the index of the frame currently in progress. If the transmitted packet was marked, the counter corresponding to the current frame is decremented. If the counter becomes zero, the session that was serviced is the last to cross the current frame. However, a second condition must be tested before performing a frame update, since it is possible for a packet to arrive with its finishing potential in the current frame during the transmission of this last marked packet. This could result in the system potential temporarily assuming a value higher than the potential of a backlogged connection, thus violating the definition of a rate-proportional server. This problem is avoided by ensuring that the timestamps of none of the queued packets fall in the current frame, just before performing the frame update. If both conditions in step 7 are satisfied, a frame update is performed by incrementing the frame number and re-calibrating the system potential to the corresponding base potential.

It is possible to avoid testing the second condition in step 7 of the algorithm by modifying the algorithm slightly. The modification consists in updating the variable P and performing the frame update when a packet is *selected* for transmission, rather than when it completes transmission. In this case, a packet arriving after the last marked packet started its service will always receive a timestamp value in the next frame. To show that this modified system remains a rate-proportional

server with the same latency, consider the following equivalent system: Assume that the traffic scheduling system consists of a regulator followed by a FFQ scheduler. The regulator holds all packets that arrive while the transmitter is busy, and delivers them to the scheduler in batches at the end of transmission of each packet. It is easy to verify that this new system consisting of the regulator and the scheduler is work-conserving.

Since packets arrive in the FFQ scheduler only at times when a packet finishes service, it is easy to verify that the a packet will never finish transmission in the packet-by-packet server later than in the corresponding fluid server.(The proof can be easily derived by extending Lemma 3 of [11].) An arriving packet may see a maximum delay of L_{max}/r in the regulator, equal to the maximum time needed for the current packet to complete service in the transmitter. Thus, the new system, consisting of the regulator and the scheduler, is still an $\mathcal{LR}$-server with the same latency as a simple rate-proportional server. However, we must note here that updating the variable P and performing the frame update when a packet is selected for transmission, rather than when it completes transmission, alters the system potential function and therefore may change the transmission sequence of packets considerably.

3.1 Correctness of Frame-based Fair Queueing

In order to be complete, it is necessary to verify that all conditions imposed in the definition of the FFQ algorithm for updating the frame are satisfied when the above algorithm is executed. We have already proved in Lemma 4 that when the frame is updated at time τ_k, the potentials of all backlogged connections in the fluid-server are at least equal to kT. We also have to prove that, at this time, the potential of any connection in the fluid-server is also less than $(k+1)T$. We will use the following sequence of three lemmas to prove this result. The proofs of these lemmas can be found in Appendix B.

Lemma 5: *Let $(0,t]$ be a server-busy period in an RPS fluid server. Let i be a session that received more service in the fluid server compared to the packet-by-packet server in the interval $(0,t]$. Then, there is another session j with $P_j(t) \geq P_i(t)$ that received more service in the packet-by-packet server than in the fluid-server during the interval $(0,t]$.*

Lemma 6: *At time τ_k when the frame is updated as described in the packet FFQ algorithm, the server has not yet transmitted any packet with potential greater than or equal to $(k+1)T$.*

Lemma 7: *Let τ_k be the time at which the kth frame-update occurs in the packet FFQ server. Then, the potential of all the connections in the fluid server at time τ_k is less than $(k+1)T$.*

3.2 Fairness of Frame-based Fair Queueing

Since frame-based fair queueing is a rate-proportional server, in order to analyze its fairness it is sufficient to prove that the difference between the system potential and the potential of any backlogged connection is always bounded. We can state the following lemma for the fluid FFQ server.

Lemma 8: *For every connection i backlogged in the fluid FFQ server at time t,*

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

A detailed proof can be found in Appendix B. The above bound applies only to the fluid server. We can now use this result and Theorem 4 of [11], which defines a general fairness bound for rate-proportional servers, to provide a bound for the packet FFQ server. This bound is given by

$$\left| \frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i} \right| \leq \max(\Delta P + C_j + \frac{L_{max}}{\rho_i} + \frac{L_j}{\rho_j}, \frac{2F-\phi_i}{r} + C_i + \frac{L_{max}}{\rho_j} + \frac{L_i}{\rho_i}) \qquad (3.6)$$

However, this bound only considers the maximum difference between the system potential and the potential of a backlogged connection. The fact that the system potential function in the FFQ algorithm is increased as a linear function of time between frame updates can be used to derive a much tighter bound:

Lemma 9: *For any two connections i, j that are continuously backlogged in the interval $(t_1, t_2]$ in the packet FFQ server,*

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

The rather long proof of the above lemma can be found in Appendix B. Thus, the fairness of the algorithm depends on the selection of the frame size. The latter, in turn, depends on the maximum packet size of each connection and its minimum bandwidth allocation. Thus, the algorithm is especially suited to application in ATM networks where the traffic consists of small fixed-size cells and the frame size can be kept small. Note, however, that the frame size does not affect the latency of the server as is the case in frame-based schedulers such as weighted-round-robin and deficit-round-robin. In addition, some short-term unfairness is unavoidable in any packet-level scheduler. The difference in normalized service received by two connections can be proportional to the number of backlogged connections even in a WFQ server. Most applications can tolerate a small amount of short-term unfairness as long as the unfairness is bounded.

4 Starting Potential-based Fair Queueing

The highest frequency at which re-calibration of the system potential can be performed in a packet server is determined by the transmission rate of packets on the outgoing link. Thus, we can attempt to perform a re-calibration each time a packet finishes service, thus improving on the fairness properties of Frame-based Fair Queueing. This approach is used in the definition of the *Starting Potential-based Fair Queueing* (SPFQ) algorithm in this section.

We define the *starting potential* of a packet of connection i as the potential of connection i when the first bit of the packet starts service in the fluid server. Let $S_i(t)$ denote the starting potential of the first packet in the queue of a backlogged connection i in the packet server. That is, $S_i(t)$ is a step function that is increased every time a new packet is placed at the head of the queue of connection i. Then, we define the base potential function $S^P(t)$ as

$$S^P(t) = \min_{i \in B^P(t)} S_i(t), \tag{4.1}$$

where $B^P(t)$ denotes the set of backlogged connections in the packet server at time t. That is, the base potential at any time t is defined as the *minimum* of the starting potentials of the backlogged connections. This allows $S^P(t)$ to be calculated in an efficient manner: Its value needs to be updated only when a packet is moved to the head of a session's queue.

To complete the specification of the algorithm, we must also define the time instants τ_k at which the re-calibration of system potential, as defined by Eq. (2.3), is performed. We define these instants to be the times at which a packet completes its service in the packet server.

Before proceeding to describe the algorithm further, we first show that the above definition of the base potential function satisfies the property in Eq. (2.2), that its value never exceeds the minimum potential of a backlogged connection in the corresponding fluid server.

Lemma 10: *If the starting potential of every backlogged session in the packet-by-packet server is greater than or equal to $S^P(t)$ at time t, then the potential of each backlogged session in the corresponding fluid server at time t is also greater than or equal to $S^P(t)$.*

Proof: We will prove the lemma by contradiction. Let us denote with i the connection with the minimum potential in the fluid server at time t, and let us assume that its potential $P_i(t)$ in the fluid server is less than $S^P(t)$. Connection i has received until time t more service in the packet-by-packet server than in the fluid server. By Lemma 3, there is another connection k with potential $P_k(t) \leq P_i(t)$ that has received less service in the packet-by-packet server than in the fluid-server. This means that the packet most recently serviced from connection k in the fluid-server has not yet finished service in the packet-by-packet server. Let s_k be the starting potential of this packet. Then,

$$s_k \leq P_k(t) \leq P_i(t).$$

By hypothesis, $P_i(t) < S^P(t)$. Therefore, we must have $s_k < S^P(t)$, which contradicts with the definition of $S^P(t)$. □

The above result enables the re-calibrations to be performed using only information extracted from the packet-by-packet server. The scheduler can keep track of all the connections that are backlogged in the packet server, and determine the minimum starting potential among their packets. When the system potential is lower than the starting potentials of the packets at the head of the queues of all the backlogged sessions, an update is performed to increase the system potential to the minimum among the starting potentials.

Let us refer again to the example of Figure 3.2. Consider an SPFQ server used to schedule traffic from three connections on an outgoing link. As in the case of FFQ, since both connections initially have the same potential they will be serviced in proportion to their reservations. As connection 3 is idle, the rates of service for connections 1 and 2 will be $0.5/(0.5+0.25) = 2/3$ and and $0.25/0.75 = 1/3$, respectively. Thus, after time 0, their potentials increase by the normalized service offered to them. That is,

$$\frac{W_1(0,\tau)}{0.5} = \frac{W_2(0,\tau)}{0.25}, \tag{4.2}$$

at any time τ when both connections remain backlogged. Notice that at time 3, the minimum starting potential of all backlogged connections is 4. At this time the potential of connection 1 in the fluid server is also $3 \cdot (2/3)/0.5 = 4$. Similarly, the potential of connection 2 is also 4 in the fluid server. However, if no re-calibration is done, the system potential will have a value of 3, equal to the real time. Thus, at the end of transmission of the third packet, we can update the system potential to 4. A similar re-calibration can be done at time 6, which brings the system potential up to 8.

The example illustrates the superior fairness of SPFQ in comparison with FFQ. In the case of the latter, when the frame size is set to 10, a frame update can be performed only at time 8, when the potentials of all backlogged connections have crossed the boundary between the first and second frames. At this time, the system potential is updated from 8 to 10. Now, if connection 3 were to send a packet just after time 6, it would see a system potential of 8 in the SPFQ algorithm, which is identical to the potentials of the backlogged connections in the corresponding fluid server. In the case of FFQ, however, the system potential at time 6 would be equal to 6, resulting in connection 3 being set to a lower potential than that of connections 1 and 2. Thus, connections 1 and 2 would be penalized for bandwidth they received while connection 3 was absent. It is easy to see that this discrepancy can increase with the frame size.

We can now describe the packet-by-packet version of SPFQ more precisely. On the arrival of a packet from connection i, an algorithm similar to FFQ is used to compute the timestamp associated with that packet. However, no special operations are required for marking packets. The steps executed on arrival of a new packet are outlined in Figure 4.1. Comparing with Figure 3.3, the only additional step is the addition of the starting potential of the new packet to a separate priority queue, so as to facilitate the re-calibration operation.

Figure 4.2 summarizes the operations performed when a packet completes transmission in the SPFQ server. The difference from Figure 3.4 lies in the re-calibration procedure in step 3 of the algorithm. This step maintains the system potential at or above the minimum starting potential of backlogged connections.

Algorithm executed on arrival of a packet from connection i:

Calculate current value of system potential.
Let t be the current time and t_s the time when
the packet currently in transmission started its service.

1. $temp \leftarrow P + (t - t_s)$

Calculate the starting potential of the new packet

2. $SP(i,k) \leftarrow \max(TS(i,k-1), temp)$

Calculate timestamp of packet

3. $TS(i,k) \leftarrow SP(i,k) + length(i,k)/\rho_i$

4. Add packet to priority queue ordered by timestamp.

5. Add starting potential of packet to separate priority queue, ordered by starting potential.

Figure 4.1: Algorithm executed on the arrival of a packet in the SPFQ server.

Algorithm executed when a packet completes service:

Increase system potential by the transmission time of the packet just completed

1. $P \leftarrow P + length(j)$

2. Delete entry corresponding to transmitted packet from the priority queue of starting potentials.

Re-calibrate system potential

3. $P \leftarrow \max(P, \min_{i \in B^P(t)} (SP(i)))$

4. Retrieve packet from head of priority queue and transmit.

Figure 4.2: Algorithm executed on completion of service of a packet in the SPFQ server.

It is easy to see that the price paid for the improved fairness of the SPFQ algorithm is in step 3 above. This re-calibration step requires knowledge of the minimum among the starting potentials of the packets at the head of the queues of all the backlogged sessions. This operation can be implemented efficiently by maintaining the starting potentials of the backlogged connections in a separate priority queue, so that the minimum value can be retrieved in $O(1)$ time. An entry is added to this priority queue when a packet is moved to the head of the queue of a connection. Likewise, when a packet completes its service, the corresponding entry is removed. If the maximum number of connections sharing the link is V, these operations can be performed in $O(\log V)$ time, the same complexity incurred in maintaining the priority queue of packets. Thus, the re-calibration step does not affect the asymptotic time-complexity of the algorithm, although it requires an additional data structure for the starting potentials.

4.1 Fairness of SPFQ

In this section we derive bounds on the short-term unfairness of the packet-by-packet SPFQ algorithm and show that it is comparable to that of Weighted Fair Queueing. In order to calculate tight bounds on the fairness of SPFQ, we will need to take into account the potentials of connections in both the packet-by-packet server and the corresponding fluid server. Let us denote with $a_i(t)$ the potential of connection i at time t in the packet-by-packet server, calculated as follows: When a new packet is placed at the head of the queue of connection i, the function $a_i(t)$ is set equal to the starting potential of that packet. While the packet is waiting for transmission, the potential remains

unchanged. When the packet starts transmission, the potential $a_i(t)$ is increased by a step equal to the normalized service offered to connection i.

As before, we will use $P_i(t)$ to denote the potential of connection i at time t in the corresponding fluid server. Note that the system potential function is identical for the two systems and will be denoted as $P(t)$. Since the packet-by-packet system is based on the fluid system, the service missed by a connection while it is absent is the same in both servers. Similarly the total service received by a connection over a system-busy period is also the same in both servers. However, at a certain instant of time t, the packet-by-packet server may be ahead of or behind the fluid server in the amount of service offered to a connection. Therefore, the potential of the connection in the packet-by-packet server may be different from its potential in the fluid server. This discrepancy, however, is always bounded.

We will first prove a lemma that establishes a correspondence between the amount of service received by a backlogged connection in the packet server during an interval $(t_1, t_2]$ and its gain in potential during the same period.

Lemma 11: *Let i be a connection in the packet server with an infinite supply of packets after time τ. For any interval of time $(t_1, t_2]$, with $t_1 \geq \tau$,*

$$\frac{\hat{W}_i(t_1, t_2)}{\rho_i} \geq a_i(t_2) - a_i(t_1) - \frac{L_{max}}{r}. \tag{4.3}$$

The proof of this lemma can be found in Appendix B. Using the above lemma, we can prove the following theorem that will provide a fairness bound for the SPFQ algorithm:

Theorem 2: *Let connections i, j have an infinite supply packets at time τ. During any interval $(t_1, t_2]$, with $\tau \leq t_1 < t_2$,*

$$\left| \frac{\hat{W}_i(t_1, t_2)}{\rho_i} - \frac{\hat{W}_j(t_1, t_2)}{\rho_j} \right| \leq \max_{1 \leq n \leq V} \left(\frac{L_n}{\rho_n}\right) + \max\left(\frac{L_i}{\rho_i}, \frac{L_j}{\rho_j}\right) + \frac{L_{max}}{r}.$$

The rather long proof of this theorem is also given in Appendix B. Note that, disregarding the term L_{max}/r, this fairness bound is nearly identical to that of WFQ [10] and SCFQ [13]. Thus, we can conclude that the maximum degree of short-term unfairness in SPFQ is very close to the best-known bound for *any* traffic scheduling algorithm.

5 Simulation Results

In this section we present some simulation results to verify our analytical bounds and study the average performance of the algorithms in comparison to Weighted Fair Queueing. Although we have shown that the upper bound on the short-term unfairness of the SPFQ algorithm is comparable to that of WFQ, it is important to compare the actual delays seen by sessions in realistic network topologies. We also compare the proposed algorithms with Self-Clocked Fair Queueing. The performance metrics we use for this comparison are the average and maximum delays experienced by the traffic sessions during the simulation. We first present results from simulating the algorithms in a single switch, followed by those from a multi-hop network configuration.

First, we simulated the four scheduling algorithms as applied to a single output port of an ATM switch. Our model consists of eight sessions sharing the same outgoing link. The reservation of each of the sessions is shown in Table 5.1. An ON-OFF traffic model was used to generate traffic within each session. Both the ON and OFF periods of the traffic model were drawn from a Poisson distribution; the mean duration of the ON period of session i was set to $100 \cdot \rho_i$ cell times, and the mean OFF time to $100 \cdot (1 - \rho_i)$ cell times, where ρ_i is the reservation of session i given in column 2 of Table 5.1. The traffic was then shaped through a leaky bucket.

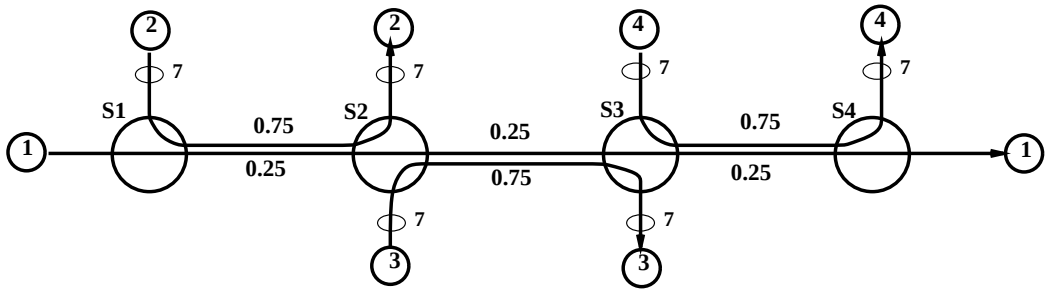

Figure 5.1: Network configuration used for multi-hop simulations.

Since our interest is in evaluating the delay in the scheduler rather than the effect of input burstiness, we selected a σ_i of 2 for each connection. We also assumed that one session (session 1) is misbehaving, attempting to transmit more than its reservation. We assumed an infinite number of buffers, causing session 1 to remain backlogged throughout the simulations. For simulations of Frame-based Fair Queueing, we set the frame size as 1000 cell times. With this model, we measured the delays and bandwidth allocations seen by all the sessions. A summary of our results is presented in Tables 5.1 and 5.2. Delays are shown in the tables in terms of cell-transmission times. The upper bounds for delay for each session in the servers, computed using Theorem 1, are as shown in Table 5.3. Note that the delay bounds are identical for WFQ, FFQ and SPFQ.

The maximum delay seen by session 0, which has reserved 50% of the link bandwidth, is substantially lower in the SPFQ server as compared to the SCFQ server. Both SPFQ and FFQ provide the same maximum end-to-end delay of 2 for this session. From Theorem 1, the upper bound on end-to-end delay for session 0 can be computed as $(2/0.5) + 1 = 5$. Note that the maximum delay observed under SCFQ is even higher than this upper bound. A large value of maximum delay may lead to increased burstiness and buffer requirements within the network if the session is going through multiple hops. This is consistent with the result in [10, 14] where it was shown that the maximum end-to-end delay for SCFQ increases with the number of connections sharing the outgoing link.

On studying the average delays seen by the individual traffic sessions, it is easy to verify that the SPFQ algorithm gives results very similar to that of WFQ. In most cases, the average delays are almost identical. Frame-based Fair Queueing also provides nearly identical delays, but SCFQ causes a substantial increase in the average delay seen by some of the sessions. Thus, while these simulation results do not bring out the superior fairness properties of SPFQ in comparison with FFQ, they clearly illustrate its superior isolation properties in comparison with SCFQ.

To compare the performance of the scheduling algorithms in a more complex network configuration, we also simulated the algorithms in a 4-hop network model shown in Figure 5.1, consisting of four ATM switches. The network model chosen was a "parking lot" configuration where one connection passes through the four switches in series and shares the outgoing link at each hop with local cross-traffic transmitted from one switch to the next. The 4-hop connection was given an allocation of 25% of the link bandwidth, and shares the outgoing link at each hop with seven other cross-traffic connections sharing the same outgoing link. One of these connections at each hop was made to misbehave just as in the case of the single-hop simulations. Traffic is shaped through a leaky bucket at the source and the burstiness σ_i was selected as 2 for all the connections. Table 5.4 provides the average and maximum delays seen by the 4-hop session at each hop for all four algorithms simulated. Again, it is easy to verify that SPFQ, FFQ and WFQ all achieve nearly identical maximum delays; these are much lower than the analytical upper-bound of $2 \cdot (1/0.25) + 3 \cdot (1/0.25) + 4 = 24$. In contrast, the delays are much higher with SCFQ. In addition, the average delays of the connection when SCFQ schedulers are used throughout the network are much different than the average delays of WFQ. On the other hand, both FFQ and SPFQ show an average behavior very similar to WFQ.

The difference in the fairness behavior of SPFQ over FFQ is not apparent in the above simulation results. This is because of the choice of a small bucket-size σ_i for the traffic shapers. To investigate

the effect of fairness, we increased the value of σ_i for session 0 to 21, maintaining σ_i for others as 2. We also reduced the arrival rate for session 0 approximately 5 percent below its reservation, thus increasing the burstiness of its traffic at the output of the leaky bucket. Other parameters of the simulation were not changed.

Table 5.5 shows the results of simulating the three algorithms in a single ATM switch with the modified traffic parameters. It is interesting to observe that the average delays of sessions 1–7 are slightly higher in SPFQ as compared to Frame-based Fair Queueing. This behavior is a direct result of the inferior fairness of FFQ. Since the potential of the bursty session is likely to diverge more from the system potential as compared to that of other sessions, FFQ often starves the bursty session temporarily to provide service for the less bursty ones, resulting in lower average delays for the latter. This behavior is evident from the larger average delay seen by session 0 under FFQ.

From Table 5.5, it is easy to see that the average delays of SPFQ fall between those of WFQ and FFQ. This brings out the fact that the maximum short-term unfairness of SPFQ lies between those of FFQ and WFQ. Indeed, the average delays of SPFQ in Table 5.5 are very close to those under WFQ. Thus, we can conclude that, in practice SPFQ may provide average performance close to that of WFQ in almost all cases.

6 Conclusions

In this paper we introduced and analyzed two novel scheduling algorithms — Frame-based Fair Queueing (FFQ) and Starting Potential-based Fair Queueing (SPFQ). Both algorithms provide the worst-case service guarantees of a Weighted Fair Queueing (WFQ) server and comparable fairness. We analyzed the fairness properties of the algorithms, and showed that the difference in normalized service offered to any two connections that are continuously backlogged is always bounded and this bound for SPFQ is comparable to that of WFQ. The main advantage of the algorithms compared to WFQ is that they do not require simulation of a fluid server in parallel, enabling them to be implemented in a simple and efficient manner. All the information needed for the algorithm can be extracted from the packet-by-packet server itself.

Compared to FFQ, SPFQ provides the same end-to-end delay bounds, but superior fairness properties. However, although SPFQ and FFQ have asymptotically the same implementation complexity, the former requires the use of two priority lists as opposed to one in the latter. Thus, SPFQ is attractive in applications where its improved fairness justifies the additional cost of implementation.

In Appendix A, we have presented a hardware implementation of the two algorithms for ATM networks. A working prototype of Frame-based Fair Queueing has been implemented in our FPGA-based Simulation Testbed for ATM Networks (FAST) [16]. The algorithm is incorporated in a shared-memory ATM switch architecture, using a set of parallel priority lists. A central controller arbitrates the sharing of the output link by the distributed shared-memory modules. The prototype works at a 16 MHz clock rate, supporting a link speed of approximately 80 Mbits/sec and up to 1024 virtual channels. Given the routing, density, and speed limitations of the FPGA devices, the implementation of the algorithm using ASIC technology is projected to support 622 Mbits/sec links easily. Experimental tests using this prototype are currently being carried out to study the average behavior of several of these algorithms.

Since timestamp computations are performed in $O(1)$ time in both the FFQ and SPFQ algorithms, the asymptotic time-complexity of the algorithms is determined by the priority-list operations. Traditional heap algorithms for insertion and deletion have a complexity of $O(\log_2 V)$ for V virtual channels. There are a number of ways for reducing this complexity for ATM networks where timestamps take integer values in a finite range. A recursive algorithm was proposed in [17, 18, 19] for implementing add and delete operations in such a priority queue with $O(\log\log V)$ time complexity, where V is the number of elements in the queue. These algorithms were further refined by Johnson [20] who presented a non-recursive algorithm with $O(\log\log D)$ complexity for the add and delete operations. In this algorithm D denotes the smallest interval between successive elements in the priority queue. Applying this algorithm to Frame-based Fair Queueing results in a complexity of $O(\log\log F)$, where F is the frame size. Furthermore, Dixon presented a method for pipelining

such an algorithm [21]. Note that, in an $N \times N$ output-buffered ATM switch, a maximum of N cells may be added to the priority list in one cell cycle while only one cell is selected for transmission. Thus, in practice, the time-complexity imposed by performing multiple insertions into the list may dominate the overall complexity of the algorithm.

Application of the above algorithms for optimizing the cost of implementing the priority lists is beyond the scope of this paper. We must point out, however, that in a high-speed ATM network environment, a more complex approach than that presented in Appendix A is unlikely to be used. Many of the algorithms mentioned in the previous paragraph are inherently recursive in nature, thus making the constant factor of a software implementation prohibitively expensive. For a reasonable number of connections, a simple approach like a k-ary heap, with some hardware support for priority encoding, as presented in Appendix A, is much more attractive.

Apart from the scheduling algorithms, a major contribution of this paper is in illustrating the power of the framework of rate-proportional servers and the definition of a general methodology for designing fair rate-proportional servers. The novelty of our approach is that instead of just designing a scheduling algorithm and analyzing its properties, we have isolated the important properties of a fair scheduling algorithm and showed how a switch- or router-designer can balance its cost and performance. It is hoped that this framework will lead to the development of other scheduling algorithms in the future.

Recently, a new algorithm was proposed for approximating Generalized-Processor Sharing, called Worst-Case Fair Weighted Fair Queueing [22]. This approach improves the discrepancy of the packet-by-packet algorithm from the fluid version. The algorithm, however, still requires parallel simulation of the fluid-model. A similar approach can be applied to the packet-by-packet version of any rate-proportional server, and to SPFQ in particular. We have found that the fairness properties of SPFQ can be further improved by using a more intelligent selection of the packets that are candidates for transmission. This analysis is beyond the scope of this paper, and will be presented in a later publication.

References

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

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

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

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

[5] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1265–79, October 1991.

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

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

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

[9] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, pp. 1374–96, October 1995.

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

[11] D. Stiliadis and A. Varma, "Rate-proportional servers: A design methodology for fair queueing algorithms," submitted to *IEEE/ACM Transactions on Networking*, April 1996.

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

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

[14] S. Golestani, "Network delay analysis of a class of fair queueing algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1057–70, August 1995.

[15] J. L. Rexford, A. Greenberg, and F. Bonomi, "Hardware efficient fair queueing architectures for high-speed networks," in *Proc. IEEE INFOCOM 96*, March 1996.

[16] A. Varma and D. Stiliadis, "FAST: an FPGA-based simulation testbed for ATM switching systems," in *Proc. ICC '96*, June 1996, to appear.

[17] P. V. E. Boas, "Preserving order in a forest in less than logarithmic time," in *Proc. 16th IEEE Conference on Foundations of Computer Science*, pp. 75–84, 1975.

[18] P. V. E. Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, vol. 10, pp. 99–127, 1977.

[19] K. Mehlhorn, *Data structures and algorithms*. Springer-Verlag, 1984.

[20] D. Johnson, "A priority queue in which initialization and queue operations take $O(\log \log d)$ time," *Mathematical Systems Theory*, vol. 15, pp. 295–309, 1982.

[21] B. Dixon, "Concurrency in an $O(\log \log n)$ priority queue," in *Proc. Parallel and Distributed Computing, Theory and Practice, First Canada-France Conference*, pp. 59–71, 1994.

[22] J. Bennett and H. Zhang, "WF^{2}Q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM 96*, March 1996.

Session	Reserved Bandwidth	Arrival Rate	SPFQ	FFQ	WFQ	SCFQ
0	0.500000	0.498	1.995	1.995	1.995	4.2353
1	0.062500	0.100	N/A	N/A	N/A	N/A
2	0.062500	0.062	7.263	6.424	7.259	15.101
3	0.062500	0.061	9.176	8.391	9.166	15.828
4	0.078125	0.076	4.040	3.685	4.152	10.284
5	0.078125	0.076	5.608	5.264	5.759	11.060
6	0.078125	0.076	6.625	6.283	6.795	11.618
7	0.078125	0.076	7.368	7.023	7.550	11.943

Table 5.1: Comparison of *average* delays from a simulation of SPFQ, FFQ, WFQ and SCFQ algorithms. The eight sessions shown share the same outgoing link. Delays are measured in terms of cell transmission times. Session 1 is misbehaving while others are transmitting within their reservations.

Session	Reserved Bandwidth	Arrival Rate	SPFQ	FFQ	WFQ	SCFQ
0	0.500000	0.498	2.00	2.00	3.00	8.00
1	0.062500	0.100	N/A	N/A	N/A	N/A
2	0.062500	0.062	25.0	21.0	25.0	29.0
3	0.062500	0.061	25.0	21.0	23.0	25.0
4	0.078125	0.076	14.0	12.0	13.0	14.0
5	0.078125	0.076	16.0	13.0	17.0	18.0
6	0.078125	0.076	17.0	14.0	16.0	18.0
7	0.078125	0.076	16.0	14.0	16.0	20.0

Table 5.2: Comparison of *maximum* delays from a simulation of SPFQ, FFQ, WFQ and SCFQ algorithms. The eight sessions shown share the same outgoing link. Delays are measured in terms of cell transmission times. Session 1 is misbehaving while others are transmitting within their reservations.

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

Table 5.3: Analytical delay bounds for the sessions in the simulation.

Hop	SPFQ		FFQ		WFQ		SCFQ	
	Avg. Delay	Max Delay	Avg. Delay	Max Delay	Avg. Delay	Max Delay	Avg. Delay	Max Delay
1	2.1203	5.00	2.1308	5.00	2.1199	6.00	4.4246	8.00
2	4.1253	7.00	4.1533	7.00	4.1580	8.00	9.0875	15.00
3	5.1804	8.00	5.2347	8.00	5.6670	9.00	12.7108	21.00
4	7.2890	11.00	7.4011	11.00	7.9301	12.00	17.7870	26.00

Table 5.4: Comparison of maximum and average delays from a simulation of SPFQ, FFQ, WFQ and SCFQ algorithms in a 4-hop network. The delays shown are for a session going through all the four switches, which has a reservation of 25%. This session shares the link at each hop with seven other sessions, one of which is misbehaving.

Session	σ	Reserved Bandwidth	Arrival Rate	SPFQ	FFQ	WFQ
0	21	0.250000	0.200	32.029	34.900	31.142
1	2	0.250000	0.249	1.6547	1.6431	1.6945
2	2	0.125000	0.121	2.9275	2.6533	3.2396
3	2	0.062500	0.059	7.0999	5.6325	7.0579
4	2	0.078125	0.070	4.1154	3.0062	4.4594
5	2	0.078125	0.071	5.0889	3.9650	5.4199
6	2	0.078125	0.071	5.6883	4.6016	6.0031
7	2	0.078125	0.071	6.1980	5.1188	6.4709

Table 5.5: Comparison of *average* delays from a simulation of SPFQ, FFQ, WFQ algorithms in a single ATM switch. The eight sessions shown share the same outgoing link. Delays are measured in terms of cell-transmission times.

Appendix A: Hardware Implementation for ATM Networks

Although the general algorithms of Sections 3 and 4 can be used in an Asynchronous Transfer Mode (ATM) network, the fixed (and small) cell-size of ATM can be exploited to simplify the algorithms considerably. In this section we will present simplified versions of the FFQ and SPFQ algorithms that will allow implementation using entirely hardware elements. The fixed size of the ATM cell allows potentials to be represented as integers, instead of the floating-point numbers required in the general implementation. This, in turn, allows an efficient hardware implementation of the priority queues. The idea of using integer timestamps was first proposed in [15].

Since the ATM cell has a fixed size, the unit of time is now chosen as $1/K$ times the transmission time of an ATM cell through the outgoing link, where $K \geq 1$ is a suitable integer scaling constant. Bandwidth reservations of sessions are assumed to be in terms of number of ATM cells per second. That is, each session i reserves a bandwidth equal to ρ_i cells/second. Integer representations of timestamps is achieved by imposing the restriction that the values of K/ρ_i for each session i is an integer. The choice of the scaling constant is based on the granularity required in the bandwidth reservations of the individual sessions. Choosing a large value of K increases the granularity with which reservations can be made. However, increasing K also increases the hardware complexity of the implementation, so a tradeoff must be made between the two.

The system potential function in these implementations is maintained as an integer, increasing it by the quantity K after the transmission of each ATM cell. As in the general implementation, a re-calibration step is used to update the system potential close to the potentials of connections with ATM cells queued in the system.

As before, the processing performed by the algorithm can be divided into two parts: (i) a part that is performed when a new cell arrives, and (ii) a part that is executed when the transmission of a cell has been completed. Since the transmission time of an ATM cell is very short, the cells arriving at the scheduler need to be processed only at the boundaries of cell transmissions, so that calculation of the system potential need not take into account the partial service received by the cell currently being transmitted. As seen in Section 3, this is equivalent to a regulator followed by a scheduler, and does not affect its delay bound.

The processing performed on a cell arrival is identical in both the FFQ and SPFQ algorithms, except for the mechanism used to bound the unfairness. The starting potential of an arriving cell is determined as the maximum of the finishing potential (timestamp) of the previous cell that arrived from the session and the system potential. The timestamp of the cell is then calculated by adding the quantity K/ρ_i to the starting potential. Note that, since the quantity K/ρ_i is an integer, this operation involves only integer arithmetic. The cell is inserted into the priority queue of cells according to its computed timestamp. In the case of FFQ, cells crossing a frame boundary are marked and the corresponding counters incremented, as in the general version of the algorithm. This step is replaced in the SPFQ algorithm by an insertion of the starting potential of the arriving cell into a second priority list.

The processing performed on the departure of a cell is also similar in both algorithms, except for the re-calibration step. The system potential value is increased by the constant K to account for the transmission time of the cell, followed by a re-calibration step. The re-calibration step in FFQ consists in decrementing the counter corresponding to the current frame if the transmitted cell was marked, and performing a frame update if the counter becomes zero. Note that the second condition in step 7 of Figure 3.4 need not be tested. In the case of SPFQ, the re-calibration step involves simply setting the system potential to the value at the head of the priority list of starting potentials, if the latter is larger. Both algorithms then select the next cell with minimum timestamp value for transmission. Note that SPFQ requires the additional step of removing the transmitted cell from the priority list of starting potentials.

We now describe a hardware implementation of the priority queue used to order cell transmissions. This implementation is attractive for high-speed ATM switches, and is based on integer representation of potentials. In addition, if we consider only the cells at the head of each session's

queue currently in the system, the difference in the values of the minimum and maximum timestamps among them must be bounded. The following two lemmas establish upper bounds for this difference for both Frame-based Fair Queueing and Starting Potential-based Fair Queueing.

Lemma 12: *Let i and j be two backlogged sessions in a packet-by-packet FFQ server. Let S_i and S_j denote the starting potentials of the first packet in the queues of connections i and j, respectively, and let F_i and F_j be the respective timestamp values. Then,*

$$F_j - S_i \leq 4\frac{F}{r}. \tag{A.1}$$

Proof: It is easy to prove the above lemma by contradiction. Assume, without loss of generality, that at time τ, $S_i < F_j$ and $F_j - S_i > 4\frac{F}{r}$. Then, at time τ, assume that both connections receive an infinite supply of packets. Note that the service offered to the two connections until time τ does not depend on packets that arrive after time τ. Then, during an interval $(\tau, t]$, it is easy to verify that connection i may receive normalized service equal to

$$\frac{\hat{W}_i^P(\tau,t)}{\rho_i} > 4\frac{F}{r},$$

while connection j is receiving no service. This is a contradiction with Lemma 9. □

Lemma 13: *Let i and j be two backlogged sessions in a packet-by-packet SPFQ server. Let S_i and S_j denote the starting potentials of the first packet in the queues of connections i and j, respectively, and let F_i and F_j be the corresponding timestamp values. Then,*

$$F_j - S_i \leq 2 \cdot \max_{1\leq n\leq V}(\frac{L_n}{\rho_n}) + \frac{L_{max}}{r}, \tag{A.2}$$

where L_n is the maximum packet size of session n and L_{max} the maximum packet size among all sessions.

Proof: The proof is again by contradiction. Assume, without loss of generality, that at time τ, $S_i < F_j$ and $F_j - S_i > 2 \cdot \max_{1\leq n\leq V}(L_n/\rho_n) + L_{max}/r$. Then, at time τ, assume that both connections receive an infinite supply of packets. It is easy to verify that, during an interval $(\tau, t]$, connection i may receive normalized service equal to

$$F_j - S_i > \max_{1\leq n\leq V}(\frac{L_n}{\rho_n}) + \max(\frac{L_i}{\rho_i}, \frac{L_j}{\rho_j}) + \frac{L_{max}}{r}.$$

This is a contradiction with Theorem 2. □

For the special case of the ATM implementation where the packet size is taken as 1 unit, and with a scaling constant of K, Eq. (A.2) simplifies to

$$F_j - S_i \leq 2 \cdot \max_{1\leq n\leq V}\left(\frac{K}{\rho_n}\right) + K. \tag{A.3}$$

Similarly, the bound for FFQ in Eq. (A.1) becomes

$$F_j - S_i \leq 4K\frac{F}{r}. \tag{A.4}$$

Note that, in SPFQ, the maximum value of K/ρ_n occurs when a session has the minimum possible reservation in the system. For example, if the minimum allowable reservation for a session is $1/1000$ of the link bandwidth, then the above difference will be $(2 \cdot 1000 + 1)K = 2001 \cdot K$. In the case of FFQ, the maximum difference is determined by the choice of the frame size, which, in turn, depends on the granularity of bandwidth reservation. In both cases, let W denote this maximum difference. Since the method involves modulo-W arithmetic operations, it is advantageous to choose W as a power of 2. Thus, we assume that W is chosen as the smallest power of 2 satisfying Eq. (A.4) or Eq. (A.3) for FFQ and SPFQ, respectively.

Thus, if we consider only the packets at the head of each backlogged session's queue, their timestamp values must fall into a window of size W. The following hardware implementation of the priority queue is based on this property. We refer to each distinct integer value within this window as a "slot." Thus, each slot corresponds to a unique value taken by the timestamp representation, modulo W. A given slot, say j, may be in one of two states:

1. There is no cell currently queued in the system with a timestamp value of j (modulo W). In this case, we say that the slot j is *empty*.
2. There is at least one cell currently in the system with a timestamp value of j (modulo W). We designate this state as *full*.

Thus, to implement a priority queue with integer timestamps, it is sufficient to maintain a separate list for each slot. That is, the list corresponding to slot j includes cells whose timestamp value is j. Selecting cells in the order of timestamp values can be accomplished simply by scanning the slots in order and transmitting the cell associated with the first slot in the full state. Slots in empty state are skipped during the scanning. In addition, the cells associated with a given slot can be transmitted in any arbitrary order, since they all have the same timestamp value. Thus, the list of cells associated with each slot can be maintained in any order that facilitates a simple implementation.

The basic system for implementing the priority queue is shown in Figure A.1. The system maintains W flip-flops, each representing the state of a slot in the current window. The flip-flops are shown around a circle in the figure, as they are conceptually organized as a circular queue and scanned in the clockwise direction. A pointer, referred to as *first-slot*, points to the beginning of the current window in the scheduling system, and therefore provides the starting point for scanning the state bits. The state bits are labeled as 0 through $(W-1)$ in the figure. The first-slot pointer is initially set to point to bit 0, and is moved cyclically as the window advances. In FFQ, the window advances by an amount of $K \cdot F$ with each frame update operation, while in SPFQ, the window advances more gradually as the system potential is increased.

The ATM cells buffered in the system reside in the cell memory. Since there may be more than one such cell in the system with a timestamp value corresponding to the slot, a list of such cells needs to be maintained. This is accomplished with an array of pointers, designated as *head-pointer array* in Figure A.1. This array consists of a total of W pointers, and each has a one-to-one correspondence with one of the state bits. The pointer at location j of the array points to a location in cell in memory where a cell with timestamp value j (modulo W) is stored. Thus, when the state of a particular slot j is determined to be full, the corresponding pointer from the head-pointer array provides access to the list of cells with timestamp j (modulo W).

The array of pointers labeled as *tail-pointer array* is used to identify the locations of the last cell received from each session. The pointer at location i of the array points to the location in cell memory where the last cell received from session i is stored. When a new cell is received, this pointer is used to add the cell to the session's queue.

Although scanning for a full slot in such an implementation takes linear time, the operation can be performed in logarithmic time by organizing the state storage in a tree structure. A similar structure can be used to implement the second priority list of starting potentials required in the SPFQ implementation. However, the list of cells maintained for each slot can be replaced in this second priority queue by a simple counter that keeps a count of cells with a given starting potential. This results in a much simpler implementation than the cell queue.

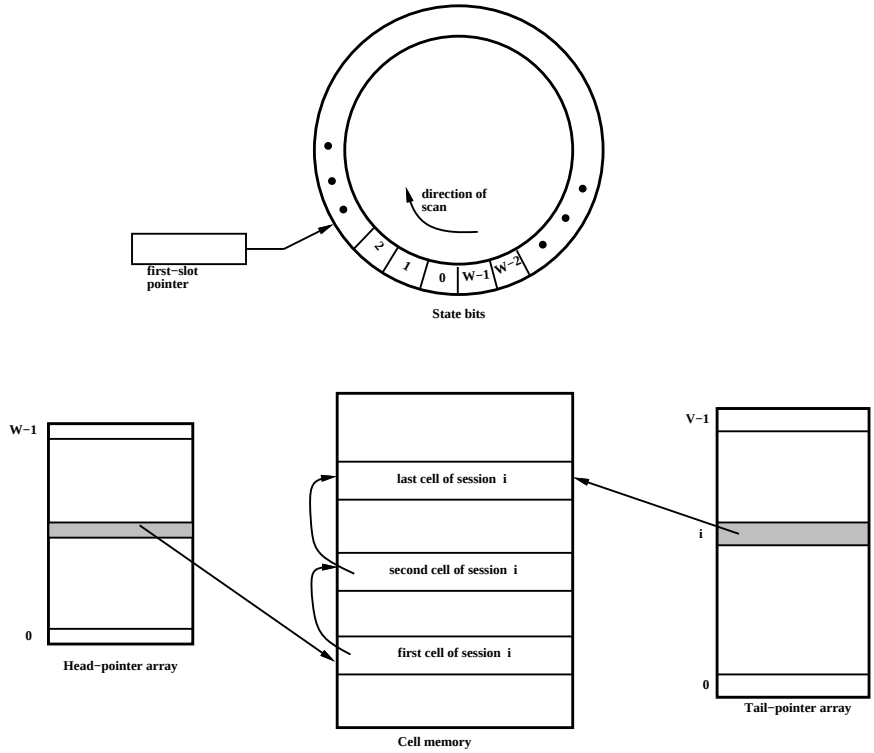

Figure A.1: Block diagram of priority queue implementation for ATM networks.

Appendix B: Proofs of Lemmas and Theorems

Proof of Lemma 5: First note that session i is backlogged in the packet-by-packet server. Since both servers are work-conserving, it is clear that if session i received more service in the fluid-server, there is another session j that has received less service in the fluid-server.

We will prove the lemma by contradiction: Let us denote with S the set of connections with potential at least equal to that of connection i and let us assume that all these connections have received more or equal service in the fluid server compared to the packet-by-packet server. That is, for all $k \in S$, $P_k(t) \geq P_i(t)$ and $W_k^F(t) \geq W_k^P(t)$. We will distinguish two cases:

Case 1: Some connection $k \in S$ is being serviced at time t. All other backlogged connections in the fluid server at time t have potential at least equal to $P_k(t)$ and thus they belong in the set S. However, we also know that there exists a connection j that has received more service in the packet-by-packet server than in the fluid-server until time t. This connection can only have potential less than $P_i(t)$. That is,

$$P_j(t) < P_i(t). \tag{B.1}$$

Since this connection received more service in the packet-by-packet server, it must still be backlogged in the fluid-server. Thus connection j should be serviced at time t instead of connection i. This is a contradiction.

Case 2: A connection m that does not belong in the set S is being serviced at time t, and thus $P_m(t) < P_i(t)$. Let τ denote the last time that a connection $k \in S$ was in service in the fluid server. Then, in the interval $(\tau, t]$ all connections $k \in S$ have not received any service in the fluid server and thus

$$W_k^F(\tau, t) = 0, \quad \forall\ k \in S. \tag{B.2}$$

Since the service function is non-decreasing, we can write

$$W_k^F(\tau, t) \leq W_k^P(\tau, t), \quad \forall\ k \in S. \tag{B.3}$$

We know that every connection $k \in S$ has received until time t more service in the fluid server than in the packet-by-packet server. Therefore,

$$W_k^F(0, t) \geq W_k^P(0, t), \quad \forall\ k \in S. \tag{B.4}$$

By subtracting Eq. (B.3) from (B.4).

$$W_k^F(0, \tau) \geq W_k^P(0, \tau), \quad \forall\ k \in S. \tag{B.5}$$

But at this time, there must exist at least one connection $j \notin S$, that received less service in the fluid server compared to the packet-by-packet server. This means that connection j is still backlogged at time τ in the fluid-server. The potential of connection j can not be lower than that of connection i, because then a connection from the set S would not be serviced just before time τ. Thus, the potential of connection j is at least equal to $P_i(\tau)$. This is a contradiction. □

Proof of Lemma 6: We will prove the lemma by contradiction. It is easy to verify from the definition of the algorithm, that the frame will be updated the first time the starting potentials of all backlogged connections in the packet-by-packet server are greater than or equal to kT. Let us assume that at some time $t < \tau_k$ the server transmitted a packet with a timestamp greater than or equal to $(k+1)T$. Then, at time t this packet would have the minimum timestamp. Since we assumed that the frame size is selected such that the largest packet can be transmitted within a frame period, the potential of all backlogged connections in the packet-by-packet server at time t would be greater than or equal to kT. Therefore, the kth frame-update would have occurred at or before t, a contradiction. □

Proof of Lemma 7: The proof is again by contradiction. Let us assume that a connection i exists with $P_i(\tau_k) \geq (k+1)T$. Then connection i has received more service in the fluid server

compared to the packet-by-packet server until time τ_k. By Lemma 5, there is another connection j with $P_j(\tau_k) \geq P_i(\tau_k) \geq (k+1)T$, that has received more service in the packet-by-packet server until time τ_k compared to the fluid-server. Let F_j denote the timestamp of the packet under service in the fluid-server for connection j. Then, $F_j \geq P_j(\tau_k) \geq (k+1)T$ and this packet has already been serviced in the packet-by-packet server. This is a contradiction to Lemma 6. □

Proof of Lemma 8: While a connection is backlogged in the FFQ server, its potential is increasing by the normalized service offered to it. The system potential, on the other hand, is increased in two cases. While the frame is not changing it is increased by the real time, and when the frame changes it becomes at least equal to the starting potential of the current frame. Let us assume that the current time is t, and that the last frame update occurred at τ_{k-1}. The next frame-update will occur after the time when all backlogged connections have crossed the potential of kT in the packet server. As we showed, this will occur before the potential of any connection becomes greater than $(k+1)T$. The largest difference between the system potential and a connection potential will appear just before the frame update. At this time

$$P_i(t) \leq (k+1)T, \tag{B.6}$$

and

$$P(t) \geq (k-1)T + \frac{\phi_i}{r}. \tag{B.7}$$

Subtracting Eq. (B.7) from (B.6),

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

Note that the fastest way for the potential of a connection to reach the value $P_i(t)$ from the time that the frame was last updated is through its normalized service. However, by the time the next frame update occurs, the system potential function would have increased by at least the time to service ϕ_i bits of connection i. This bounds the difference in potentials to $2T - \frac{\phi_i}{r}$. □

Proof of Lemma 9: In order to calculate tight bounds on the fairness of the packet-by-packet version of FFQ, we will need to take into account the potentials of connections in both the packet-by-packet server and the corresponding fluid server. Let us denote with $a_i(t)$ the potential of connection i at time t in the packet-by-packet server, calculated as follows: When a new packet is placed at the head of the queue of connection i, the function $a_i(t)$ is set equal to the starting potential of that packet. While the packet is waiting for transmission, the potential remains unchanged. When the packet starts transmission, the potential $a_i(t)$ is increased by a step equal to the normalized service offered to connection i.

As before, we will use $P_i(t)$ to denote the potential of connection i at time t in the corresponding fluid server. Note that the system potential function is identical for the two systems and will be denoted as $P(t)$. Since the packet-by-packet system is based on the fluid system, the service missed by a connection while it is absent is the same in both servers. Similarly the total service received by a connection over a system-busy period is also the same in both servers. However, as shown in Lemmas 3 and 4 of [11], at a certain instant of time t, the packet-by-packet server may be ahead of or behind the fluid server in the amount of service offered to a connection. Therefore, the potential of the connection in the packet-by-packet server may be different from its potential in the fluid server. This discrepancy, however, is always bounded.

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

$$\frac{\hat{W}_i(t_1,t_2)}{\rho_i} \geq \frac{\hat{W}_j(t_1,t_2)}{\rho_j}, \tag{B.8}$$

for a time interval $(t_1, t_2]$ with $\tau \leq t_1 < t_2$. We know that after time τ, both connections have an infinite supply of packets. Thus, the potential of both connections in the fluid server is only

increased by the normalized service offered to them. For the service offered to connection i in the interval $(t_1, t_2]$, we can write

$$\frac{\hat{W}_i^P(t_1,t_2)}{\rho_i} \leq a_i(t_2) - P_i(t_1) + \frac{\hat{W}_i^F(0,t_1) - \hat{W}_j^P(0,t_1)}{\rho_i}. \tag{B.9}$$

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

$$\frac{\hat{W}_j^P(t_1,t_2)}{\rho_j} \geq a_j(t_2) - P_j(t_1) - \frac{\hat{W}_j^P(0,t_1) - \hat{W}_j^F(0,t_1)}{\rho_j}. \tag{B.10}$$

That is, the normalized service offered to connection j in the packet-by-packet server during the interval $(t_1, t_2]$ is equal to the increase in its potential minus the additional service that the packet-by-packet server may have offered to connection j until time t_1. By subtracting Eq. (B.10) from Eq. (B.9),

$$\begin{aligned}\frac{\hat{W}_i^P(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j^P(t_1,t_2)}{\rho_j} \quad \leq \quad & a_i(t_2) - a_j(t_2) + P_j(t_1) - P_i(t_1) \\ & + \frac{\hat{W}_i^F(0,t_1) - \hat{W}_j^P(0,t_1)}{\rho_i} + \frac{\hat{W}_j^P(0,t_1) - \hat{W}_j^F(0,t_1)}{\rho_j}. \end{aligned} \tag{B.11}$$

We have assumed that connection i has received more normalized service; thus, from the definition of the packet-by-packet rate-proportional servers,

$$a_i(t_2) \leq a_j(t_2) + \frac{L_j}{\rho_j}. \tag{B.12}$$

Thus, Eq. (B.11) can be written as

$$\begin{aligned}\frac{\hat{W}_i^P(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j^P(t_1,t_2)}{\rho_j} \quad \leq \quad & \frac{L_j}{\rho_j} + (P_j(t_1) - P_i(t_1)) \\ & + \left(\frac{\hat{W}_i^F(0,t_1) - \hat{W}_j^P(0,t_1)}{\rho_i} + \frac{\hat{W}_j^P(0,t_1) - \hat{W}_j^F(0,t_1)}{\rho_j} \right) \end{aligned} \tag{B.13}$$

The difference in offered service will be maximized when the second part of the above equation is maximized. Notice that this will happen when the difference in potentials between the two connections is maximized, and the difference in offered service between the two servers for the two connections is maximized as well. Let us assume that the last frame update occurred at time τ_{k-1}. Then, from Lemma 6, we know that there is no packet serviced in the fluid-server with a finish potential higher than $(k+1)T$. Thus,

$$a_j(t_1) \leq (k+1)T. \tag{B.14}$$

At the same time, for connection i we can write that

$$a_i(t_1) \geq (k-1)T. \tag{B.15}$$

Notice also that the additional service offered by the packet-by-packet server compared to the fluid-server is never more than the difference in potentials between the two servers. Therefore,

$$P_j(t_1) + \frac{\hat{W}_j^P(0,t_1) - \hat{W}_j^F(0,t_1)}{\rho_j} \leq a_j(t_1) \leq f(t_1) + 2T. \tag{B.16}$$

Similarly, for connection i,

$$P_i(t_1) - \frac{\hat{W}_i^F(0,t_1) - \hat{W}_j^P(0,t_1)}{\rho_i} \geq a_i(t_1) \geq f(t_1). \tag{B.17}$$

From Eq. (B.13), and by subtracting Eq. (B.17) from (B.16), we can conclude that

$$\frac{\hat{W}_i^P(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j^P(t_1,t_2)}{\rho_j} \leq \frac{2F}{r} + \frac{L_j}{\rho_j}. \tag{B.18}$$

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

$$\frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i} \leq \frac{2F}{r} + \frac{L_i}{\rho_i}. \tag{B.19}$$

From Eq. (B.13) and (B.14) we can conclude that

$$\left| \frac{\hat{W}_j(t_1,t_2)}{\rho_j} - \frac{\hat{W}_i(t_1,t_2)}{\rho_i} \right| \leq \frac{2F}{r} + \max\left(\frac{L_i}{\rho_i}, \frac{L_j}{\rho_j} \right). \tag{B.20}$$

□

Proof of Lemma 11: The assumption of the lemma states that at time τ connection i has an infinite supply of packets. We will distinguish two cases: First, just before time τ connection i was not backlogged. Therefore, all packets in the queue of connection i can be considered as arriving at time τ. It is easy to verify that the starting potential of each packet arriving after τ will be calculated from the finishing potential (timestamp) of the previous packet. Thus, Eq. (4.3) is obvious.

In the second case, connection i was already backlogged at time τ. Let us denote with F_i the finishing potential of the first packet in the queue of connection i at time τ. If $P(t) \leq F_i$, then again the starting potential of all packets serviced during the interval $(t_1, t_2]$ is estimated by the finishing potential of the previous packet. However, it is possible that $F_i < P(\tau)$ and the starting potential of some of the packets in the queue of connection i after time τ was estimated from the system potential. From the definition of rate-proportional servers we know that at time τ,

$$P(\tau) \leq P_i(\tau). \tag{B.21}$$

However, $a_i(\tau) \leq F_i \leq P(\tau)$. This, implies that the fluid server has offered more service to connection i until time τ than the packet-by-packet server. Note also that the system potential has increased to a value greater than $a_i(\tau)$ through only the passage of real-time and not through a re-calibration, since the starting potential of the first packet of connection i is less than $P(\tau)$. Let us denote with t^*, the time at which the function $P_i(t)$ had a value equal to F_i. From Theorem 4 of [11] we know that,

$$\tau - t^* \leq \frac{L_{max}}{r}.$$

The system potential may have reached a value of F_i only at or after time t^*, and after that time it has increased only by the real time that elapsed. Thus, for the system potential we can write

$$P(\tau) \leq F_i + (\tau - t^*) \leq F_i + \frac{L_{max}}{r}. \tag{B.22}$$

Thus, some of the increase in potential of connection i in the packet server may be due to increase in the system potential. This total increase, however, happened before time τ and from the above equation can not be more than L_{max}/r. Thus,

$$W_i(t_1, t_2) \geq a_i(t_2) - a_i(t_1) - \frac{L_{max}}{r}. \tag{B.23}$$

□

Proof of Theorem 2: Let us assume two connections i and j. Without loss of generality, let us assume that at time t_1,

$$a_i(t_1) \leq a_j(t_1). \tag{B.24}$$

Let us also denote with p_i^k the first packet of connection i in the system at time t_1. Note that even if the packet is being serviced at time t_1, it will still be considered as being in the system. However, the partial service offered to a connection needs to be accounted for when we try to bound the difference in normalized service between two connections. Similarly, p_j^m is the first packet of connection j. Let us also assume that after time $\tau \leq t_1$ both connections have an infinite supply of packets. In order to be able to compute a tight bound we will have to consider several cases and subcases.

Case 1: When packet p_i^k arrived in the system at some time $t^* \leq t_1$, packet p_i^m is already in the system. We will separate the problem in two subcases:

Subcase 1: Packet p_j^m was the first packet in the queue of connection j at time t^*. Let F_j denote the timestamp of the last packet selected for transmission from connection j. Then, that packet was selected for transmission before time t^*. At this time it was the packet with the minimum timestamp. All other connections could only have packets with timestamp $\geq F_j$. However, this implies that, for every connection backlogged at time t^* in the packet-by-packet system,

$$a_k(t^*) \geq F_j - \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}), \forall k \in B(t). \tag{B.25}$$

Therefore, by the definition of a Fair Rate-Proportional Server,

$$P(t^*) \geq F_j - \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}). \tag{B.26}$$

Therefore, since the starting potential associated with packet p_i^k will be at least equal to $P(t^*)$,

$$\begin{aligned} a_i(t_1) &\geq S_i^k \\ &\geq P(t^*) \\ &\geq F_j - \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}), \quad \text{from Eq. (B.26)} \\ &\geq a_j(t) - \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}). \end{aligned} \tag{B.27}$$

The last inequality holds since F_j denotes the timestamp of the last packet selected from connection j. Thus, the potential $a_j(t)$ of connection j can not have increased to a value larger than F_j.

Subcase 2: Packet p_j^m was not the first packet of connection j at time t^*. Let us again denote with F_j the timestamp of the last packet selected from connection j before time t_1. Since both packet p_i^k and p_j^m are the only packets that may be serviced from the two connections at time t_1, the timestamp F_j of that packet can not be greater than $a_i(t_1) + \frac{L_i}{\rho_i}$. Therefore,

$$\begin{aligned} a_j(t_1) &\leq F_j \\ &\leq a_i(t_1) + \frac{L_i}{\rho_i} \\ &\leq a_i(t_1) + \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}). \end{aligned} \tag{B.28}$$

From Equations (B.27) and (B.28), we can conclude that in both subcases

$$a_j(t_1) \leq a_i(t_1) + \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}). \tag{B.29}$$

At time t_2, the potential of connection i can be greater or less than that of connection j. In either case, from the definition of the algorithm,

$$a_i(t_2) \leq a_j(t_2) + \frac{L_j}{\rho_j}. \tag{B.30}$$

For the service offered to connection i, we can write

$$\begin{aligned}
\frac{\hat{W}_i(t_1,t_2)}{\rho_i} &\leq a_i(t_2) - a_i(t_1) \\
&\leq a_j(t_2) - a_i(t_1) + \frac{L_j}{\rho_j}, \quad \text{from Eq. (B.30)} \\
&\leq a_j(t_2) - a_j(t_1) + \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}) + \frac{L_j}{\rho_j}, \quad \text{from Eq. (B.29)} \\
&\leq \frac{\hat{W}_j(t_1,t_2)}{\rho_j} + \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}) + \frac{L_j}{\rho_j} + \frac{L_{max}}{r}.
\end{aligned} \tag{B.31}$$

However, connection j may have received more normalized service than connection i. In this case, from the definition of the algorithm, $a_j(t_2) \leq a_i(t_2) + \frac{L_i}{\rho_i}$. Then,

$$\begin{aligned}
\frac{\hat{W}_j(t_1,t_2)}{\rho_i} &\leq a_j(t_2) - a_j(t_1) \\
&\leq a_i(t_2) - a_j(t_1) + \frac{L_i}{\rho_i} \\
&\leq a_i(t_2) - a_i(t_1) + \frac{L_i}{\rho_i} \\
&\leq \frac{\hat{W}_i(t_1,t_2)}{\rho_i} + + \frac{L_i}{\rho_i} + \frac{L_{max}}{r}.
\end{aligned} \tag{B.32}$$

Combining the two previous cases it is easy to verify that

$$\left| \frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j} \right| \leq \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}) + \frac{L_{max}}{r} + \max(\frac{L_i}{\rho_i}, \frac{L_j}{\rho_j}). \tag{B.33}$$

Case 2: Packet p_i^k arrived before packet p_j^m. Let us again denote with t^* the arrival time of packet p_j^m. We have to consider again several subcases:
Subcase 1: Let us assume that at least one other packet was selected for transmission from connection j after time t^* and before time t_1. Let F_j denote the timestamp of the last packet selected from connection j. Then $a_j(t_1) \leq F_j$ and $F_j \leq a_i(t^*) + \max_{1 \leq n \leq V}(L_n/\rho_n)$. Proceeding similarly as in the previous case, we can conclude that

$$a_j(t_1) \leq a_i(t_1) + \max_{1 \leq n \leq V}(\frac{L_n}{\rho_n}). \tag{B.34}$$

Subcase 2: No packet from connection j was selected after time t^* and the packet p_j^m received its starting potential from the finishing potential of the previous packet of connection j. Let us denote that packet with p_j^{m-1}. When that packet was selected for transmission, it had the smaller timestamp F_j. Therefore, all the other connections had packets with timestamps $\geq F_j$; but this implies that the starting potential of all packets from the other connections were at least $F_j - \max_{1 \leq n \leq V}(L_n/\rho_n)$.

If p_i^k was in the system at that time, its starting potential would also have been greater than this value. The system potential was also greater than $F_j - \max_{1\le n\le V}(L_n/\rho_n)$. If p_i^k arrived later, it would have received a starting potential that is at least equal to the system potential and thus it is also greater than $F_j - \max_{1\le n\le V}(L_n/\rho_n)$. Thus, in any case, the starting potential of p_i^k was greater than this value. Therefore,

$$a_j(t_1) - \max_{1\le n\le V}(\frac{L_n}{\rho_n}) \le F_j - \max_{1\le n\le V}(\frac{L_n}{\rho_n}) \le a_i(t^*) \le a_i(t_1). \tag{B.35}$$

From Eq. (B.34) and (B.35) we can conclude that $a_j(t_1) \le a_i(t_1) + \max_{1\le n\le V}(L_n/\rho_n)$. Following a similar procedure as in the previous case, it is easy to verify that for both of these subcases the theorem holds, and

$$\left|\frac{\hat{W}_i(t_1,t_2)}{\rho_i} - \frac{\hat{W}_j(t_1,t_2)}{\rho_j}\right| \le \max_{1\le n\le V}(\frac{L_n}{\rho_n}) + \frac{L_{max}}{r} + \max(\frac{L_i}{\rho_i}, \frac{L_j}{\rho_j}). \tag{B.36}$$

Subcase 3: The final subcase will require a more elaborate approach. No packet from connection j was selected after time t^* and packet p_j^m received its starting potential from the system potential. In that case we need to bound the difference between the potential of connection i in the packet-by-packet server and the system potential. As was described in the proof of Lemma 11, since the potential of connection i in the packet-by-packet server is less than the system potential $P(t)$, this implies that the packet-by-packet server has offered less service to connection i than the fluid-server. The system potential may be higher than the finishing potential of the first packet in the queue of connection i by $\Delta P_1 \le L_{max}/r$. Thus, we can write

$$a_j(t_1) = a_j(t^*) = P(t^*) \le a_i(t^*) + \frac{L_i}{\rho_i} + \Delta P_1. \tag{B.37}$$

In fact, there may be several packets from connection j that arrived after packet p_j^m and before time τ, and they all would have received a starting potential equal to the system potential. Note, however, that the total increase in the potential of connection j due to increases in the system potential, denoted by ΔP_2, together with ΔP_1 can only be less than L_{max}/r. Thus, we can write

$$\frac{\hat{W}_j(t_1,t_2)}{\rho_j} \ge a_j(t_2) - a_j(t_1) - \Delta P_2. \tag{B.38}$$

Following the same procedure as in the above cases, using Eq. (B.38) instead of Lemma 11 for estimating the service offered to connection j, and with the fact that $\Delta P_1 + \Delta P_2 \le L_{max}/r$, we can again prove the bound for the difference in normalized service offered to the two connections. □