Packet Fair Queueing Algorithms for Wireless Networks with Location-Dependent Errors

T.S. Eugene Ng   Ion Stoica   Hui Zhang

Feburary 2000

CMU-CS-00-112

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

An earlier version of this paper appeared in *Proceedings of IEEE INFOCOM'98*.

This research was sponsored by DARPA under contract numbers N66001-96-C-8528 and N00174-96-K-0002, and by a NSF Career Award under grant number NCR-9624979. Additional support was provided by Intel Corp., MCI, and Sun Microsystems.

Keywords: Wireless network, channel error, resource management, scheduling, delay guarantee, fairness.

Abstract

While Packet Fair Queueing (PFQ) algorithms provide both bounded delay and fairness in wired networks, they cannot be applied directly to wireless networks. The key difficulty is that in wireless networks sessions can experience *location-dependent channel errors*. This may lead to situations in which a session receives significantly less service than it is supposed to, while another receives more. This results in large discrepancies between the sessions' virtual times, making it difficult to provide both delay-guarantees and fairness simultaneously.

Our contribution is twofold. First, we identify a set of properties, called *Channel-condition Independent Fair* (CIF), that a Packet Fair Queueing algorithm should have in a wireless environment: (1) delay and throughput guarantees for error-free sessions, (2) long term fairness for error sessions, (3) short term fairness for error-free sessions, and (4) graceful degradation for sessions that have received excess service. Second, we present a methodology for adapting PFQ algorithms for wireless networks and we apply this methodology to derive a novel algorithm based on Start-time Fair Queueing, called *Channel-condition Independent packet Fair Queueing* (CIF-Q), that achieves all the above properties. To evaluate the algorithm we provide both theoretical analysis and simulation results.

1 Introduction

As the Internet becomes a global communication infrastructure, new Quality of Service (QoS) service models and algorithms are developed to evolve the Internet into a true integrated services network. At the same time, wireless data networks are becoming an integral part of the Internet, especially as an access networking technology. An important research issue is then to extend the QoS service models and algorithms developed for wired networks to wireless networks. In this paper, we study how to implement Packet Fair Queueing (PFQ) algorithms in wireless networks.

PFQ algorithms are first proposed in the context of wired networks to approximate the idealized Generalized Processor Sharing (GPS) policy [2, 7]. GPS has been proven to have two important properties: (a) it can provide an end-to-end bounded-delay service to a leaky-bucket constrained session; (b) it can ensure fair allocation of bandwidth among all backlogged sessions regardless of whether or not their traffic is constrained. The former property is the basis for supporting guaranteed services while the later property is important for supporting best-effort and link-sharing services. While GPS is a fluid model that cannot be implemented, various packet approximation algorithms are designed to provide services that are almost identical to that of GPS.

Unfortunately, the GPS model and existing PFQ algorithms are not directly applicable to a wireless network environment. The key difficulty is that there are *location-dependent channel errors* in a wireless environment. In GPS, at any given time, all backlogged sessions send data at their fair rates. However, in a wireless environment, some mobile hosts may not be able to transmit data due to channel errors, while other hosts may have error-free channels and can transmit data. To be work-conserving, it is impossible to achieve the instantaneous fairness property defined by the GPS model because only a subset of backlogged sessions are eligible for scheduling. That is, a session with an error-free channel may receive more normalized amount of service than that by a session with an error channel. However, it is conceivable to achieve long term fairness by giving more service to a previously error session so that it can be compensated. Of course this compensation can only be achieved by degrading the services of other sessions, which may affect the QoS guarantees and fairness property for these sessions. It is unclear what is the right model and algorithm to provide QoS guarantee and ensure fairness in a wireless network.

In this paper, we identify a set of properties, called *Channel-condition Independent Fair*

(CIF), desirable for any PFQ algorithm in a wireless network: (1) delay and throughput guarantees for error-free sessions, (2) long term fairness guarantee among error sessions, (3) short term fairness guarantee among error-free sessions, and (4) graceful degradation in quality of service for sessions that have received excess service. We then present a methodology for adapting PFQ algorithms for wireless networks and we apply this methodology to derive a new scheduling algorithm called the *Channel-condition Independent packet Fair Queueing* (CIF-Q) algorithm that achieves the CIF properties. New algorithmic techniques are introduced in the CIF-Q algorithm. We prove that CIF-Q achieves all the properties of the CIF and show that it has low implementation complexity. Finally, we use simulation to evaluate the performance of our algorithm.

The rest of this paper is organized as follows. In Section 2 we describe the network model that we are assuming and in Section 3, we discuss in detail the problems involved in applying existing PFQ algorithms in wireless networks. We present the CIF properties in Section 4 and the CIF-Q algorithm in Section 5. We then show that the CIF-Q algorithm achieves all the properties of CIF in Section 6. Finally, we present simulation results in Section 7 and conclude the paper in Section 8.

2 Network Model

In this paper, we consider a simplified shared-channel wireless cellular network (e.g. WaveLAN [9]) model in which each cell is served by a base station. Centralized scheduling of packet transmissions for a cell is performed at the base station, and media access control is integrated with packet scheduling. Mobile hosts may experience location-dependent channel errors in the sense that they cannot receive or transmit data error-free. Error periods are assumed to be short and sporadic relative to the lifetimes of the sessions so long term fairness is possible. Instantaneous knowledge of channel conditions (error or error-free) and packet queue status of all sessions is assumed at the base station. Under these assumptions, the difference between a PFQ algorithm in a wired and wireless environment is that in the latter a backlogged session may not be able to receive service due to location independent errors. Lu et al have given this broad problem a good initial formulation in [6], and have effectively addressed many practical issues. Therefore, in this paper, we focus on the algorithmic aspects of the problem.

3 GPS and PFQ

In wired networks, Packet Fair Queueing (PFQ) is based on the GPS model [7]. In a GPS each session i is characterized by its allocated rate, r_i. During any time interval when there are exactly M non-empty queues, the server serves the M packets at the head of the queues simultaneously, in proportion to their rates.

Each PFQ algorithm maintains a system virtual time $V(\cdot)$. In addition, it associates to each session i a virtual start time $S_i(\cdot)$, and a virtual finish time $F_i(\cdot)$. Intuitively, $V(t)$ represents the normalized fair amount of service that each session should have received by time t, $S_i(t)$ represents the normalized amount of service that session i has received by time t, and $F_i(t)$ represents the sum between $S_i(t)$ and the normalized service that session i should receive for serving the packet at the head of its queue. Since $S_i(t)$ keeps track of the normalized service received by session i by time t, $S_i(t)$ is also called the virtual time of session i, and alternatively denoted $V_i(t)$. The goal of all PFQ algorithms is then to minimize the discrepancies among $V_i(t)$'s and $V(t)$. This is usually achieved by selecting for service the packet with the smallest $S_i(t)$ or $F_i(t)$. Notice that the role of the system virtual time is to reset $S_i(\cdot)$ (or $V_i(\cdot)$) whenever an unbacklogged session i becomes backlogged again. More precisely,

$$S_i(t) = \begin{cases} \max(V(t), S_i(t-)) & i \text{ becomes active} \\ S_i(t-) + \frac{l_i^k}{r_i} & p_i^k \text{ finishes} \end{cases} \quad (1)$$

$$F_i(t) = S_i(t) + \frac{l_i^{k+1}}{r_i} \quad (2)$$

where p_i^k represents the k-th packet of session i, and l_i^k represents its length.

While GPS and PFQ algorithms provide both guaranteed and fairness services in a wired network, they cannot achieve both properties in a wireless network. The key difference is that there are *location-dependent channel errors* in a wireless environment. That is, some mobile hosts may not be able to transmit data due to channel errors even when there are backlogged sessions on those hosts while others may have error-free channels and can transmit data in that time. Since GPS is work-conserving, during such a period with location-dependent channel errors, error-free sessions will receive more service than their fair share, while a session with errors will receive no service. Since the virtual time of a session increases only when it receives service, this may result in a large difference between the virtual time of an error session i and that of an error-free session. There are two problems with this large discrepancy between session virtual times:

1. If session i exits from errors, and is allowed to retain its virtual time, then it will have the smallest virtual time among all sessions. The server will select session i *exclusively* for service until its virtual time catches up with those of other sessions. In the meantime, all other sessions will receive *no* service. Since a session can be in error indefinitely, the length of such zero-service period for the error-free sessions can be arbitrarily long.

2. If session i exits from errors, and its virtual time is updated to the system virtual time $V(\cdot)$, then the error-free sessions will not be penalized. However, session i's history of lost service is now completely erased and session i will never be able to regain the service. This results in unfair behaviors.

To address these problems, in [6], Lu et al augmented the GPS model and proposed the Wireless Fluid Fair Queueing (WFFQ) service model and the Idealized Wireless Fair Queueing (IWFQ) algorithm for packet systems. Their observation is that, to ensure fairness, it is desirable to let sessions that fall behind to "catch-up" with the other sessions. However, allowing an unbounded amount of "catch-up" can result in denial of service to error-free sessions. Therefore, in WFFQ, only bounded amount of "catch-up" B is allowed. As a result, delay and throughput guarantees to error-free sessions become possible.

The WFFQ model and the IWFQ algorithm, while provide limited fairness and bounded throughput and delay guarantees for error-free sessions, has several limitations. First, there is a coupling between the delay and fairness properties. To achieve long term fairness, a lagging session should be allowed to catch-up as much as possible, which requires a large B. However, a large B also means that an error-free session can face a large "denial of service" period and experience a large delay. Thus, one cannot have perfect fairness while at the same time achieve a low delay bound for an error-free session using the WFFQ model. In this paper, we will show that these two properties are in fact orthogonal and both can be achieved.

In addition, the service selection policy used in WFFQ and IWFQ gives absolute priority to the session with the minimum virtual time. Consequently, as long as there exists a lagging session in the system, all other leading or non-leading sessions in the system cannot receive service. Under this selection policy, compensation for all lagging sessions will take the same amount of time regardless of their guaranteed rate, contradicting the semantics that a larger guaranteed rate implies better quality of service.

We believe the root of the problems lies in the fact that the virtual time parameter in GPS is not adequate for performing both scheduling functions and fairness enforcement in a wireless environment. In the next section we present the desirable properties of a PFQ algorithm for wireless networks.

4 The CIF Properties

To implement PFQ algorithm in an environment with location-dependent errors, we need to address two main questions: (1) How is the service of an error session distributed among the error-free sessions? (2) How does a session that was in error and becomes error-free receive back the "lost" service? Although the answers to the above questions may depend on the specifics of a particular algorithm, in this section we give four generic properties, collectively call *Channel-condition Independent Fair* (CIF), that we believe any such algorithm should have. The first two are:

1 *Delay bound and throughput guarantees.* Delay bound and throughput for error-free sessions are guaranteed, and are not affected by other sessions being in error.

2 *Long term fairness.* During a large enough busy period, if a session becomes error-free, then, as long as it has enough service demand, it should get back all the service "lost" while it was in error.

Thus, a session which becomes error-free will eventually get back its entire "lost" service. However, as implied by the first property, this compensation should *not* affect the service guarantees for error-free sessions.

Next, we classify sessions as *leading*, *lagging*, and *satisfied*. A session is leading when it has received more service than it would have received in an ideal error-free system, lagging if it has received less, and satisfied if it has received exactly the same amount of service. Then, the last two properties are:

3 *Short term fairness.* The difference between the normalized services received by any two error-free sessions that are continuously backlogged and are in the same state (i.e., leading, lagging, or satisfied) during a time interval should be bounded.

4 *Graceful degradation.* During any time interval while it is error-free, a leading backlogged session should be guaranteed to receive at least a minimum fraction of its service in an error-free system.

The third property is a generalization of the well-known *fairness* property in classical PFQ algorithms. The requirement that sessions in the same state receive the same amount of normalized service implies that (1) leading sessions should be penalized by the same normalized amount during compensation, (2) compensation services should be distributed in proportion to the lagging sessions' rates, and (3) when services from error sessions are available, lagging sessions receive these services at the same normalized rate, so do leading sessions and satisfied sessions. Finally, the last property says that in the worst case a leading session gives up only a percentage of its service. This way, an adaptive application may continue to run.

5 The CIF-Q Algorithm

In this section we present our *Channel-condition Independent Packet Fair Queueing* (CIF-Q) algorithm for systems with location-dependent channel errors.

In order to account for the service lost or gained by a session due to errors, we associate to each system S a reference error-free system S^r. Then, a session is classified as *leading*, *lagging*, or *satisfied* with respect to S^r, i.e., a session is leading if it has received more service in S than it would have received in S^r, lagging if it has received less, and satisfied if it has received the same amount. The precise definition of S^r depends on the corresponding PFQ algorithm we choose to extend for the error system. Although theoretically we can choose any of the well-known algorithms, such as WFQ [2, 7], SCFQ [4], WF^{2}Q+ [1], EEVDF [8], for simplicity, in this paper we use Start-time Fair Queueing (SFQ) [5]. The reason for this choice is that in a system with location-dependent channel errors, it is harder to do scheduling based on the finishing times than on the starting times. This is because finishing times are computed based on the length of the packets at the head of sessions' queues, and finishing times scheduling assumes implicitly that once a session is selected, that packet can be sent. Unfortunately, this is not true in an error system; a session can enter in error just before the packet is transmitted. In this case the service should be given to another session, whose packet may have a different length. Since, as we shall see, in our algorithm this service is charged to the session which is selected in the first place, this might create service inversions. More precisely, if the packet that is actually transmitted is longer than the packet that is supposed to be sent, the resulting finishing time can be larger than the finishing time of another error-free session that has packets to send. Since SFQ does not make use of finishing times in scheduling decisions, it does not exhibit this problem.

Thus, to every error system S we associate an error-free reference system S^r_{SFQ} with the following properties:

1. S^r_{SFQ} employs an SFQ algorithm, i.e., packets are served in the increasing order of their virtual starting times,

2. The same session is selected at the same time in both systems.

3. Whenever a session is selected in S^r_{SFQ}, the packet at the head of its queue is transmitted. In contrast, whenever a session is selected in S, it is possible that the packet of another session is transmitted. This happens when the selected session is in error, or when it is leading and has to give back its lead.

4. A session is active during the same time intervals in both systems. In S a session is said to be active if it is backlogged, or as long as it is leading. In S^r_{SFQ} a session is active only as long as it is backlogged.

There are two things worth noting. First, the scheduling decisions are made in S^r_{SFQ}, and not in S. More precisely, the session that has the smallest virtual time in S^r_{SFQ} is selected to be served in S. Second, no matter what session is actually served[1] in S, in S^r_{SFQ} the transmitted packet is assumed to be belonging to the *selected* session, and therefore its virtual time is updated accordingly.

Below we give some of the key techniques introduced by our CIF-Q algorithm.

[1] As implied by 3, the selected session may not be served if it is in error or has to give up some of its lead.

- Unlike other PFQ algorithms, in CIF-Q, a session's virtual time does *not* keep track of the normalized service received by that session in the real system S, but in the *reference* error-free system S^r_{SFQ}.

- To provide fairness, we use an additional parameter (called *lag*) that keeps track of the difference between the service that the session should receive in S^r_{SFQ} and the service it has received in S. Then, to achieve perfect fairness, the lag of every session should be zero.

- A leading session is not allowed to leave until it has given up its lead. Otherwise, as we will show later, this translates into an aggregate loss for the other active sessions.

- To deal with the case when all active sessions are in error, we introduce the concept of forced compensation. We force a session to receive service and we charge it for this service, even if it cannot send any packet. This makes it possible to ensure delay and throughput guarantees for error-free sessions.

Finally, we note that our algorithm is self-clocking in the sense that there is no need for emulating a fluid flow system for scheduling or keeping track of lead and lag. As a result, our algorithm has lower implementation complexity than IWFQ [6] which requires the emulation of a fluid system.

For clarity, we first describe a simple version of CIF-Q that achieves the two most important properties of CIF: (1) delay and throughput guarantees for error-free sessions, and (2) long term fairness for error sessions. Definitions of some key terms appearing in this section are shown in Table 1.

5.1 CIF-Q: Simple Version

Besides a virtual time v_i, each session i in CIF-Q is associated with an additional parameter lag_i that represents the difference between the service that session i should receive in a reference error-free packet system and the service it has received in the real system. An active session i is said to be *lagging* if its lag_i is positive, *leading* if its lag_i is negative, and *satisfied* otherwise. In the absence of errors, lag_i of all active sessions are zero. Since the system is work-conserving, the algorithm maintains at all time the following invariant:

$$\sum_{i \in \mathcal{A}} lag_i = 0, \tag{3}$$

Term	Definition
Leading session	A session i that has a negative lag_i
Lagging session	A session i that has a positive lag_i
Satisfied session	A session i that has a zero lag_i
Lead	The absolute value of a negative lag_i
Lag	The value of lag_i
Backlogged session	A session that has a queue length > 0
Active session	A session that is either backlogged or unbacklogged with a negative lag
Can send	A session can send if it is backlogged and experiences no error at the moment
Excess service	Service made available due to errors
Compensation service	Service made available due to a leading session giving up its lead
Additional service	Excess or compensation service
Lost service	Service lost due to errors that is received by another session
Forgone service	Service lost due to errors that is *not* received by another session

Table 1: *Definitions of terms used in the description of the CIF-Q algorithm.*

where $\mathcal{A}$ is the set of active sessions. The simple version of CIF-Q is shown in Figure 1.

When a session i becomes backlogged and active, its lag is initialized to zero. Its virtual time is initialized to the maximum of its virtual time and the minimum virtual time among other active sessions to ensure the virtual times of all active sessions are bounded. The algorithm selects the active session i with the minimum virtual time for service. If that session is not leading and can send, then the packet at the head of its queue is transmitted; this ensures error-free non-leading sessions get their fair share. Its virtual time is advanced as follows to record the amount of normalized work:

$$v_i = v_i + \frac{l_i^k}{r_i} \tag{4}$$

on session i **receiving** packet p:
    enqueue($queue_i, p$)
    if ($i \notin \mathcal{A}$)
        $v_i = \max(v_i, \min_{k \in \mathcal{A}}\{v_k\})$;
        $lag_i = 0$;
        $\mathcal{A} = \mathcal{A} \cup \{i\}$; /∗ *mark session active* ∗/

on sending current packet: /∗ *get next packet to send* ∗/
    $i = \min_{v_i}\{i \in \mathcal{A}\}$; /∗ *select session with min. virtual time* ∗/
    if ($lag_i \geq 0$ **and** (i can send)) /∗ *session i non-leading, can send* ∗/
        $p =$**dequeue**($queue_i$);
        $v_i = v_i + p.length/r_i$;
    else
        $j = \max_{lag_k/r_k}\{k \in \mathcal{A} \mid k \text{ can send}\}$;
        if (j **exists**)
            $p =$**dequeue**($queue_j$);
            $v_i = v_i + p.length/r_i$; /∗ *charge session i* ∗/
            $lag_i = lag_i + p.length$;
            $lag_j = lag_j - p.length$;
            if ($i \neq j$ **and empty**($queue_j$) **and** $lag_j \geq 0$)
                leave(j);
        else /∗ *there is no active session ready to send* ∗/
            $v_i = v_i + \delta/r_i$;
            if ($lag_i < 0$ **and empty**($queue_i$)) /∗ *i is leading, unbacklogged* ∗/
                $j = \max_{lag_k/r_k}\{k \in \mathcal{A}\}$;
                $lag_i = lag_i + \delta$;
                $lag_j = lag_j - \delta$; /∗ *forced compensation* ∗/
                set_time_out(on sending, δ/R);
    if (**empty**($queue_i$) **and** $lag_i \geq 0$)
        leave(i);

leave(i) /∗ *session i leaves* ∗/
    $\mathcal{A} = \mathcal{A} \setminus \{i\}$;
    for ($j \in \mathcal{A}$) /∗ *update lags of all active sessions* ∗/
        $lag_j = lag_j + lag_i \times r_j/(\sum_{k \in \mathcal{A}} r_k)$;
    if ($\exists j \in \mathcal{A}\ s.t.\ \mathbf{empty}(queue_j) \wedge lag_j \geq 0$)
        leave(j);

Figure 1: *Simple version of the CIF-Q algorithm.*

where l_i^k is the length of the k^{th} packet of session i and r_i is the rate of session i. However, if the session is leading or cannot send, we search for the session j with the largest normalized lag that can send a packet. If there is such a session j, the packet at the head of its queue is transmitted. That is, when additional service is available, we first try to compensate the session that is normalized lagging the most. Note that session i's virtual time (*not* sessions j's virtual time) is advanced and lag_i and lag_j are adjusted. The key is that by doing so we charge the packet transmission to session i (not j), and we keep track of this by adjusting the lags of the two sessions accordingly. The lags adjustments indicate that session i has now given up l_j^k amount of service, while session j has now received l_j^k amount of additional service. This selection policy reduces to SFQ in an error-free system.

To achieve long term fairness, in addition to compensating lagging sessions, we need to address the following question: What should happen if a session i with a non-zero lag becomes unbacklogged and wants to leave the active set? Clearly, if session i is allowed to leave, we need to modify the lag of at least one other active session in order to maintain the invariant (3) of the algorithm. Our solution is that when a lagging session i wants to leave, its positive lag_i is proportionally distributed among all the remaining active sessions j such that each lag_j is updated according to the following equation:

$$lag_j = lag_j + lag_i \frac{r_j}{\sum_{k \in \mathcal{A}} r_k}, \tag{5}$$

where $\mathcal{A}$ represents the set of the remaining active sessions. In contrast, a leading session is *not* allowed to leave the active set until it has given up all its lead.

Intuitively, when a lagging session becomes unbacklogged and wants to leave, its positive lag is "unjustified" because it does not have enough service demand to attain such lag. In addition, the leaving of a lagging session translates into gains in services for the remaining active sessions. By updating their lags according to equation (5), we practically distribute this gain in proportion to their rates. Therefore, such lag can be safely redistributed back into the system. In contrast, if a leading session is allowed to leave, and its lead (negative lag) is redistributed back into the system, then the remaining active sessions are penalized. If the leading session's lead is not redistributed back into the system and its lead history is erased (reset to zero), the aggregate sum over the lags of the remaining sessions becomes negative. Consequently, even if none of the remaining sessions experiences any errors in the future, they cannot get back their lost services unless the leading session that left the system becomes active again and gives back its lead. On the other hand, if the lead history is retained, then the leading session may be unnecessarily penalized in the future when it becomes active again. Therefore, a leading session is not allowed to leave.

With the mechanisms discussed so far, as long as there exists an active session that can send, lost services by a session are always reflected as leads in other active sending sessions. Therefore, if all the error sessions exit from error and remain error-free for a long enough period of time, the normalized lag of all active sessions approaches zero and the long term fairness property of CIF is achieved. There is however a special case where no active sending sessions are left in the system to receive the excess service from an error session. Such service is said to be *forgone* and active error sessions are not allowed to reclaim such forgone services. In this case, the algorithm advances the active error session's virtual time using a dummy packet of length δ so that all active sessions can be chosen by the server[2] in the correct order even when none of them can send.

A similar special case exists for distributing compensation service. Recall that a leading unbacklogged session i is not allowed to leave until it has given up all its lead. However, if all other active sessions are in error and cannot receive compensation service from this leading session, this leading session may be stuck in the active set indefinitely. Using the dummy packet, we allow a leading unbacklogged session i to gradually give up its lead by *forcing* an active error lagging session j to "receive" δ amount of compensation service. In effect, we force session j to *forgo* δ amount of service. If the leading unbacklogged session is not allowed to give up its lead by forcing the compensation, the allocated share of this leading session can be violated at a later time. Thus, the algorithm ensures that, given enough service demand from an error-free session, it always receives no less than its guaranteed share of service. As a result, the algorithm is capable of providing a delay bound to an error-free session whose source is constrained by a leaky-bucket regardless of the behavior of other sessions in the system.

In summary, in this simple version of the CIF-Q algorithm, we have achieved two properties of CIF. First, long term fairness is ensured. Second, an error-free session is always guaranteed its fair share, thus there is a delay bound for an error-free session whose source is constrained by a leaky-bucket that is independent of the behavior of any other sessions in the system. As a result, real-time guarantee and long term fairness are decoupled. These properties are shown in Section 6.

[2]Recall the server always chooses the session with the minimum virtual time.

Parameter	Definition
α	Minimal fraction of service retained by any leading session
s_i	Normalized amount of service actually received by a leading session i through virtual time (v_i) selection since it became leading
c_i	Normalized amount of additional service received by a lagging session i
f_i	Normalized amount of additional service received by a non-lagging session i

Table 2: *Definitions of new parameters used in the full version of CIF-Q.*

5.2 CIF-Q: Full version

The simple version of the CIF-Q algorithm has two major drawbacks. First, the service received by a leading session does not degrade gracefully when it is necessary for it to give up its lead. In fact, a leading session receives *no* service at all until it has given up all its lead. The second drawback is that only the session with the largest normalized lag receives additional services. That is, short term fairness is not ensured. Consequently, during certain periods of time, a session with a smaller guaranteed rate can actually receive better normalized service than a session with a larger guaranteed rate. This contradicts the semantics that a larger guaranteed rate implies better quality of service.

The full version of the CIF-Q algorithm which addresses both of these problems is shown in Figure 2 and 3. Several new parameters are introduced and their definitions can be found in Table 2. For clarity, we have separated out some groups of operations into new functions. Function **send_pkt**(j,i) now contains the operations performed when the server serves a packet from session j but charge the service to session i. Because of the changes in lags resulting from the charging technique, sessions' states may change. Therefore, several cases are listed to check for state changes to update each parameter accordingly. Operations related to sending a dummy packet, which are identical to those in the simple version, are now in the **send_dummy_pkt**(i) function. In addition, parameters are also updated when a session exits from error state as shown in the processing of the **on exiting** from error-mode event, and when a session leaves the active set as shown in the **leave**(i) function.

```
on session i receiving packet p:
   enqueue(queue_i, p)
   if (i ∉ A)
      v_i = max(v_i, min_{k∈A}{v_k});
      lag_i = 0;
      f_i = max(f_i, min_{k∈A}{f_k | lag_k ≤ 0 ∧ k can send});
      A = A ∪ {i}; /* mark session active */


on sending current packet: /* get next packet to send */
   i = min_{v_i}{i ∈ A};
   if ((i can send) and (lag_i ≥ 0 or (lag_i < 0 and s_i ≤ αv_i)))
      send_pkt(i, i); /* session i served through v_i selection */
   else /* i cannot send or i is leading and not allowed to send */
      /* select lagging session j to compensate */
      j = min_{c_k}{k ∈ A | lag_k > 0 ∧ k can send};
      if (i can send)
         if (j exists)
            send_pkt(j, i); /* serve session j but charge to i */
         else /* there is no lagging session that can send */
            send_pkt(i, i); /* service given back to session i */
      else /* i cannot send */
         if (∀k ∈ A  k cannot send)
            send_dummy_packet(i);
         else /* there is at least one session that can send */
            if (j exists)
               send_pkt(j, i); /* serve session j but charge to i */
            else /* no active lagging session, and i cannot send */
               /* select session j to receive excess service */
               j = min_{f_k}{k ∈ A | session k can send};
               send_pkt(j, i); /* serve session j but charge to i */
      if (i ≠ j and empty(queue_j) and lag_j ≥ 0)
         leave(j); /* j becomes inactive */
   if (empty(queue_i) and lag_i ≥ 0)
      leave(i); /* i becomes inactive */
```

Figure 2: The full version of the CIF-Q algorithm (Part I).

```
send_pkt(j, i) /* serve session j but charge to i */
   p = dequeue(queue_j);
   v_i = v_i + p.length/r_i; /* charge session i */
   if (i == j and lag_i < 0 and s_i ≤ α v_i)
      /* session i is leading and served through v_i selection */
      s_i = s_i + p.length/r_i;
   if (i ≠ j)
      lag_j = lag_j − p.length; /* session j has gain extra service */
      if (lag_j > 0)
         /* case 1: j continues to be lagging */
         c_j = c_j + p.length/r_j;
      if (lag_j + p.length ≤ 0 and lag_j ≤ 0)
         /* case 2: j continues to be non-lagging */
         f_j = f_j + p.length/r_j;
      if (lag_j + p.length > 0 and lag_j ≤ 0)
         /* case 3: j just becomes non-lagging */
         f_j = max(f_j, min_{k∈A}{f_k | lag_k ≤ 0 ∧ k can send});
      if (lag_j + p.length ≥ 0 and lag_j < 0)
         s_j = α v_j; /* case 4: j just becomes leading */
      lag_i = lag_i + p.length; /* session i has lost service */
      if (lag_i − p.length ≤ 0 and lag_i > 0)
         /* case 5: i just becomes lagging */
         c_i = max(c_i, min_{k∈A}{c_k | lag_k > 0 ∧ k can send});

send_dummy_pkt(i) /* i was selected, but no session can send */
   v_i = v_i + δ/r_i; /* send an infinitesimally small dummy packet */
   if (lag_i < 0 and empty(queue_i))
      j = max_{lag_k/r_k}{k ∈ A};
      lag_i = lag_i + δ;
      lag_j = lag_j − δ; /* forced compensation */
      set_time_out(on sending packet, δ/R);

on session i exiting from error-mode:
   if (lag_i > 0)
      c_i = max(c_i, min_{k∈A}{c_k | lag_k > 0 ∧ k can send});
   else
      f_i = max(f_i, min_{k∈A}{f_k | lag_k ≤ 0 ∧ k can send});
   if (lag_i < 0)
      s_i = α v_i;

leave(i) /* session i leaves */
   A = A \ {i};
   for (j ∈ A) /* update lags of all active sessions */
      lag'_j = lag_j;
      lag_j = lag_j + lag_i × r_j/(Σ_{k∈A} r_k);
      if (lag'_j ≤ 0 and lag_j > 0 and j can send)
         /* j just becomes lagging */
         c_j = max(c_j, min_{k∈A}{c_k | lag_k > 0 ∧ k can send});
   if (∃j ∈ A s.t. empty(queue_j) ∧ lag_j ≥ 0)
      leave(j);
```

Figure 3: The full version of the CIF-Q algorithm (Part II).

To achieve graceful degradation in service for leading sessions, we use a system parameter α ($0 \leq \alpha \leq 1$) to control the minimal fraction of service retained by a leading session. That is, a leading session has to give up at most $(1 - \alpha)$ amount of its service share to compensate for lagging sessions. To implement this policy, we associate to each leading session i a parameter s_i, which keeps track of the normalized service actually received by such leading session through virtual time (v_i) selection. When a session i becomes leading, s_i is initialized to αv_i (see case 4 in **send_pkt** and **on exiting** from error-mode). Thereafter, s_i is updated whenever a leading session is served through virtual time selection (see **send_pkt**). When selected based on v_i, a leading session is assured service only if the normalized service it has received through virtual time selection since it became leading is no larger than α of the normalized service it should have received based on its share. That is, a leading session is assured service only if $s_i \leq \alpha v_i$. Intuitively, the larger the value of α, the more graceful the degradation experienced by leading sessions. At the limit, when α is set to one, no compensation is given to lagging sessions.

To provide short term fairness, we distinguish the two types of additional service in the algorithm: *excess service* and *compensation service*. Excess service is made available due to a session's error, while compensation service is made available due to a leading session giving up its lead.

First of all, lagging sessions have higher priority to receive additional services to expedite their compensation. But we now distribute these additional services among lagging sessions in proportion to the lagging sessions' rates, instead of giving all of it to the session with the largest normalized lag. This way a lagging session is guaranteed to catch up, no matter what the lags of the other sessions are, and the short term fairness property is ensured among lagging sessions during compensation. This policy is implemented by keeping a new virtual time c_i that keeps track of the normalized amount of additional services received by session i while it is lagging. When a session i becomes both lagging and can send, c_i is initialized according to (see case 5 in **send_pkt**, **on exiting** from error-mode and **leave**):

$$c_i = \max(c_i, \min_{k \in \mathcal{A}} \{c_k \mid lag_k > 0 \wedge k \text{ can send}\}). \tag{6}$$

When additional service is available, the lagging session j with the minimum c_j that can send is chosen to receive it. Session j's c_j is then updated accordingly (see case 1 in **send_pkt**). However, if such session j does not exist, then there are two scenarios. First, if the additional service is a compensation service, then this service is given back to the original chosen session i. Otherwise, it must be an excess service. If none of the active sessions can send at the moment, then **send_dummy_packet**(i) is called to advance the virtual time v_i and perform any applicable forced compensation. But if there are active sessions that can send left in the system, then this excess service is distributed among all non-lagging sending sessions in proportion to their rates. This way, short term fairness is ensured among non-lagging sessions when excess services are available. This policy is implemented by keeping a virtual time f_i that keeps track of the normalized amount of excess services received by session i while it is non-lagging. When a session i becomes non-lagging and sending, f_i is initialized according to (see **on receiving** packet, case 3 in **send_pkt** and **on exiting** from error-mode):

$$f_i = \max(f_i, \min_{k \in \mathcal{A}}\{f_k \mid lag_k \leq 0 \wedge k \text{ can send}\}). \tag{7}$$

To distribute the excess service, the non-lagging session j with the minimum f_j that can send is chosen to receive it. Session j's f_j is then updated accordingly (see case 2 in **send_pkt**).

In summary, using the four new parameters (α, s_i, c_i, and f_i) and the associated mechanisms presented above, the full version of the CIF-Q algorithm now achieves (a) graceful degradation in service for leading sessions and (b) short term fairness guarantee (these properties are shown in Section 6) in addition to (c) long term fairness guarantee and (d) error-free sessions delay bound/throughput guarantee that are achieved by the simple version of the algorithm. Thus, all the properties of CIF are satisfied.

5.3 Algorithm Complexity

In this section we discuss the algorithm complexity. We are interested in the complexity of each of the following five operations: (1) a session becoming active, (2) a session becoming inactive, (3) a session being selected to receive service, (4) an active session entering error mode, and (5) an active session becoming error-free. It can be deduced from Figure 2 that these operations ultimately reduce to the following basic set operations: adding, deleting, and querying the element with the minimum key from the set. Since these operations can be efficiently implemented in $O(\log n)$ by using a heap data structure, a straightforward implementation of our algorithm would be to maintain three heaps based on v_i, f_i, and c_i, respectively. More precisely, the first heap will maintain all *active* sessions based on v_i, the second one will maintain all *non-lagging* *error-free* sessions based on f_i, and the last one will maintain all *lagging error-free* sessions based on c_i. Since with the exception of the leaving operation, all the other four operations involve only a constant number of heap operations, it follows that they can be implemented in $O(\log n)$, where n represents the number of active sessions.

Regarding the leaving operation, when the lag is non-zero, this operation requires updating of the lags of all other active sessions. However, when a session's lag changes, that session might change its state from leading to lagging, which eventually requires moving it from one heap to another. Thus, in the worst case the leaving operation takes $O(n \log n)$.

Although the leaving operation takes significantly longer than that in an error-free Packet Fair Queueing algorithm, we note that in wireless networks, algorithm efficiency is not as critical as in wired networks. The main reason for this is that wireless networks are mainly used as access technology, they have significantly lower bandwidth, and support a significantly lower number of hosts compared to wired networks. As an example, the current WaveLAN technology provides 2 Mbps theoretical throughput and supports on the order of 100 hosts [9]. These figures are several orders of magnitude smaller than the ones for a high speed communication switch.

6 Fairness and Delay Results

In this section we show that our algorithm meets the properties presented in Section 4. Specifically, Theorem 1 says that the difference between the normalized services received by two error-free active sessions during any time interval in which they are in the same state (i.e., leading, satisfied, or lagging) is bounded (Property 3), Theorem 2 says that the time it takes a lagging session that no longer experiences errors to catch up is bounded (Property 2), and finally, Theorem 3 gives the delay bound for an error-free session (Property 1). Note that Property 4 is explicitly enforced by the algorithm via the parameter α. The complete proofs can be found in the Appendix.

Theorem 1 *The difference between the normalized service received by any two sessions i and j during an interval $[t_1, t_2)$ in which both sessions are continuously backlogged, error-free, and their status does not change is bounded as follows:*

$$\left| \frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j} \right| \leq \beta \left(\frac{L_{max}}{r_i} + \frac{L_{max}}{r_j} \right), \tag{8}$$

where $W_i(t_1, t_2)$ represents the service received by session i during $[t_1, t_2)$, L_{max} is the maximum packet length, and $\beta = 3$ if both sessions are non-leading, $\beta = 3 + \alpha$ otherwise.

Theorem 2 *Consider an active lagging session i that becomes error-free after time t. If session i is continuously backlogged after time t, then it is guaranteed to catch up after at most Δ units of time,*

$$\Delta = \frac{\widehat{R}^2}{r_i r_{min}(1-\alpha)R} lag_i(t) + \left(\frac{\widehat{R}(\widehat{R}/r_i + n + 2)}{r_{min}(1-\alpha)} + n + 1 + \frac{\widehat{R}}{r_{min}}\right)\frac{L_{max}}{R}, \tag{9}$$

where n is the number of sessions that are active at any time in $[t, t')$, R is the channel capacity, L_{max} is the maximum length of a packet, $\widehat{R}$ is the aggregate rate of all sessions in the system, and r_{min} is the minimum rate of any session.

Theorem 3 *The delay experienced by a packet of an error-free session i with rate r_i in an error system S is bounded by*

$$(n-1)\frac{L_{max}}{R} + \frac{l_i^k}{R} + \frac{L_{max}}{r_i}, \tag{10}$$

where n is the number of active sessions, l_i^k is the length of the k^{th} packet of session i, and R is the channel capacity.

7 Simulation Experiments

In this section, we present results from simulation experiments to demonstrate the delay bound guarantees and the fairness properties of CIF-Q. All the simulations last for 200 seconds and there are seven sessions: a real-time audio session, a real-time video session, four FTP sessions, and a cross traffic session to model the rest of the traffic in the system. The properties of each session are shown in Table 3. The audio and video sessions are constant-bit-rate (CBR) sources such that their packets are evenly spaced at 50 ms apart[3] and their throughputs

[3]To be more realistic and to avoid the worst case behavior of SFQ, the packet spacing has a small probability of drifting slightly

	Pkt size	Guaranteed rate	Src model	Error
Audio	1 KB	160 Kbps	CBR	None
Video	8 KB	1.25 Mbps	CBR	None
FTP-1	3 KB	2 Mbps	Greedy	None
FTP-2	3 KB	2 Mbps	Greedy	Pattern 1
FTP-3	8 KB	2 Mbps	Greedy	Pattern 2
FTP-4	8 KB	2 Mbps	Greedy	Pattern 1
Cross	4 KB	10 Mbps	Poisson	None

Table 3: *Properties of the 7 sessions used in the simulations.*

	Max	Min	Mean	Std Dev
Audio	46 ms	0.40 ms	4.1 ms	4.4 ms
Video	49 ms	3.2 ms	6.9 ms	4.3 ms

Table 4: *Packet delay statistics for the audio and video sessions when α is 0.9.*

are 160 Kbps and 1.25 Mbps respectively. The four 2 Mbps FTP sessions are all continuously backlogged. Finally, the cross traffic session is a Poisson source with an average rate of 10 Mbps.

For clarity in showing the effects of channel errors and for ease of interpretation, we choose to model errors as simple periodic error bursts rather than using a more realistic model [3]. During the 200 second periods of our simulation experiments, channel errors occur only during the first 45 seconds, leaving enough error-free time to demonstrate the long term fairness property of our algorithm. Error pattern 1 represents a periodic error burst of 1.6 second with 3.2 seconds of intermediate error-free time. Error pattern 2, a less severe error pattern, represents a periodic error burst of 0.5 seconds with 5.5 seconds of intermediate error-free time. Notice session FTP-2 and session FTP-4 experience identical error pattern but have different packet sizes, while session FTP-1 experiences no error at all. In the following, we present two sets of simulation results using different values as the the system parameter α.

7.1 $\alpha = 0.9$

An α value of 0.9 intuitively means that leading sessions will give up up-to 10 percents of their service rates to compensate for lagging sessions. Table 4 shows the packet delays statistics

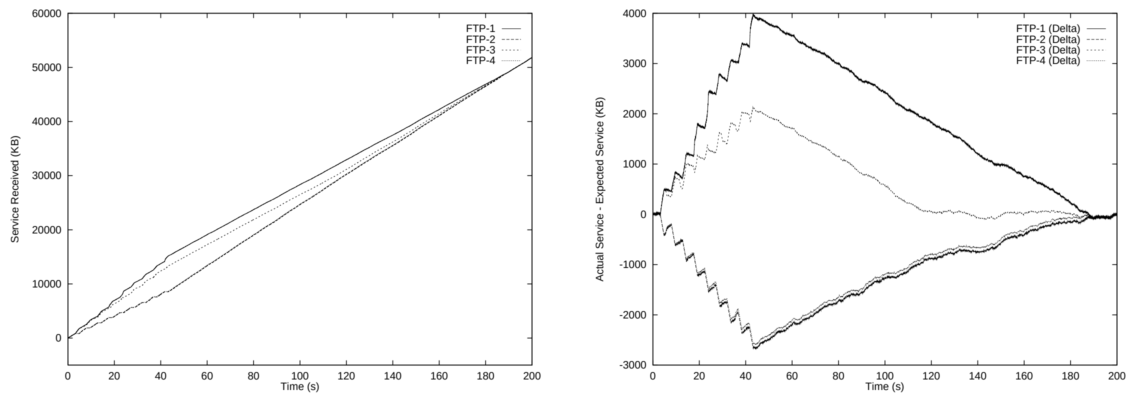


Figure 4: *Behavior of the FTP sessions when α is 0.9. (a) Service received by each FTP session. Note that FTP-2,4 are the bottom two lines that virtually overlap each other. (b) Difference between the actual service received by the FTP sessions and the corresponding expected amount of service. Note this is not the same as the lead defined in the CIF-Q algorithm*

for the two real-time sessions under this compensation policy. For comparison purpose, if the audio and video sessions were served by an error-free fluid GPS system, their packets would have a delay bound of 50 ms. Clearly, the delays experienced by the audio and video packets under our algorithm compare favorably against the GPS delay bound and are well below the worst case delay bound of our algorithm. The worst case delay bound is much larger than 50 ms due to the SFQ discipline used. However, a packet experiences the worst case delay only when the starting virtual time of all sessions are perfectly synchronized. This is avoided in the simulation by introducing small infrequent drifts into the packet spacing to portrait a more realistic situation.

In addition to providing delay bound guarantees, an equally important aspect of our algorithm is on fairness. To demonstrate the fairness properties, consider the behavior of the four FTP sessions as shown in Figure 4. Figure 4(a) shows the amount of service received by each FTP session over the period of the simulation. Recall that sessions FTP-2,3,4 experience errors during the first 45 seconds of the simulation as evidenced by the flat periods in their service progressions. Sessions FTP-2,4 experience identical errors and session FTP-3 experiences slighter errors. Session FTP-1 is error-free during the simulation.

The most notable feature in Figure 4(a) is the fact that the service received by all four FTP sessions, regardless of the amount of errors they have experienced, converges gradually when the system becomes error-free. This demonstrates the perfect long term fairness guarantee over

	Max	Min	Mean	Std Dev
Audio	43 ms	0.40 ms	4.1 ms	4.4 ms
Video	51 ms	3.2 ms	7.0 ms	4.5 ms

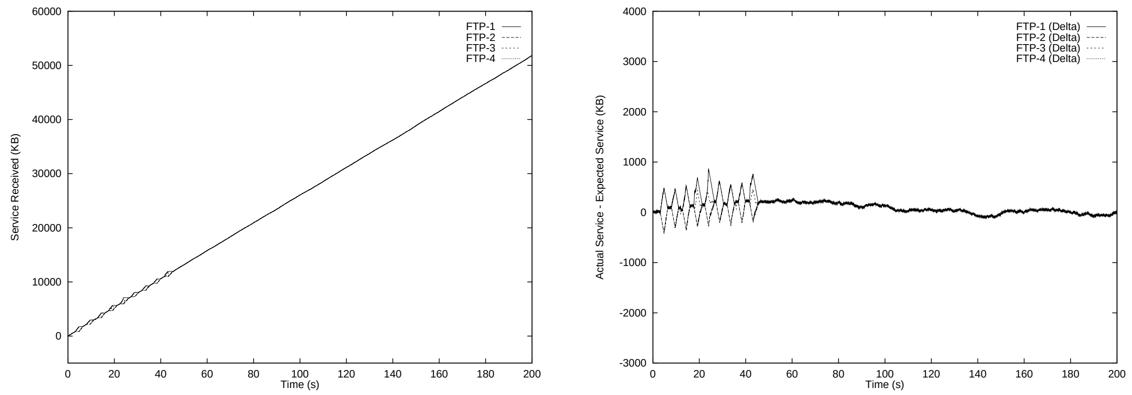
Table 5: *Packet delay statistics for the audio and video sessions when α is 0.0.*

a busy period provided by our algorithm. To see the changes in leads and lags more easily, we show in Figure 4(b) the difference between the actual service received by the FTP sessions and the corresponding expected amount of service. The expected amount of service is computed as the product of the overall throughput and time. A leading session gives up its lead to lagging sessions at a rate of $1 - \alpha$ that of its actual service rate. Notice the give-up rates and compensation rates varies slightly since the Poisson traffic of the cross traffic session affects the actual service rates.

Finally, notice in both Figure 4(a) and (b), the lines for sessions FTP-2 and FTP-4 almost overlap each other and the lines for sessions FTP-1 and FTP-3 parallel each other while they are both leading. This shows the short term fairness guarantee provided by our algorithm which states that the difference in normalized services received by two sessions during a period in which they are in the same state (leading or lagging, error or error-free) is bounded. This ensures that all leading sessions in the same error state give up their leads at approximately the same normalized speed and that all lagging sessions in the same error state get compensated at about the same normalized speed. One might incorrectly assume that the lines for sessions FTP-2 and FTP-4 should completely overlap each other since they experience the same errors. The reason they do not is that the difference in the amount of normalized services received may drift apart when the sessions change states as can be seen in Figure 4(b). Nonetheless, it is important to note that the two lines are parallel during periods where the two sessions do not change state.

7.2 $\alpha = 0.0$

In this experiment, the value of α is zero. This means that a leading session i will receive *no* service as long as there exists a lagging error-free session in the system. This absolute priority compensation behavior is similar to the behavior of the algorithm proposed in [6], except that we have not put an artificial upper bound on this zero-service period and that real-time require-



(a) (b)

Figure 5: *Behavior of the FTP sessions when α is 0.0. (a) Service received by each FTP session. (b) Difference between the actual service received by the FTP sessions and the corresponding expected amount of service.*

ments are still guaranteed. Although we believe such aggressive compensation is not desirable, it is worthwhile to demonstrate the behavior of our algorithm under this policy. Even though such an aggressive compensation policy is used, the delays experienced by real-time packets are unaffected under our algorithm (See Table 5). Thus, delay bounds for real-time sessions are guaranteed independent of the value of α or whether compensation is bounded. The value of α only affects the fairness properties of the system. That is, real-time delay bound guarantee and fairness guarantees are decoupled under our algorithm.

In Figure 5, we show the behavior of the four FTP sessions Clearly, the services received by the four FTP sessions converge very rapidly after each error period. However, the price to pay for such absolute priority compensation is the abrupt changes in the available bandwidth experienced even by error-free sessions (e.g. FTP-1). Despite the abruptness, it is clear from Figure 5 that the long term and short term fairness guarantees provided by our algorithm still hold. One thing worth explaining is that in Figure 5(b), the lines converge to a value above zero and then slowly drop to zero together. This is due to the changing actual service rates caused by the Poisson traffic of the cross traffic session in the system. Nevertheless, the convergence of the services sufficiently shows the fairness properties of our algorithm.

8 Conclusion

In this paper, we make two main contributions. First, we identified four key properties (CIF) that any PFQ algorithm should have in order to work well in a wireless network where channel errors are location-dependent. Specifically, the properties are (1) delay guarantees and throughput guarantees for error-free sessions, (2) long term fairness guarantee for error sessions, (3) short term fairness guarantee for error-free sessions, and (4) graceful degradation in quality of service for sessions that have received excess service. As a second contribution, we present a methodology for adapting PFQ algorithms for wireless networks and we apply this methodology to derive a new scheduling algorithm called CIF-Q that provably achieves all the properties of CIF. Four novel algorithmic techniques are introduced in CIF-Q to make achieving the CIF properties possible. We demonstrate the performance of CIF-Q in simulation and show how compensation rate can be tuned to suit specific needs. As possible further work, the CIF-Q algorithm may be extended to support hierarchical link-sharing service.

References

[1] J.C.R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of the ACM-SIGCOMM 96*, pages 143–156, Palo Alto, CA, August 1996.

[2] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[3] D. Eckhardt and P. Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. In *Proceedings of ACM SIGCOMM'96*, Standford University, CA, August 1996.

[4] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, CA, April 1994.

[5] P. Goyal, H.M. Vin, and H. Chen. Start-time Fair Queuing: A scheduling algorithm for integrated services. In *Proceedings of the ACM-SIGCOMM 96*, pages 157–168, Palo Alto, CA, August 1996.

[6] S. Lu, V. Bharghavan, and R. Srikant. Fair scheduling in wireless packet networks. In *Proceedings of ACM SIGCOMM'97*, Cannes, France, September 1997.

[7] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. *ACM/IEEE Transactions on Networking*, 1(3):344–357, June 1993.

[8] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the IEEE RTSS 96*, pages 288 – 289, December 1996.

[9] B. Tuch. Development of WaveLAN, an ISM band wireless LAN. *AT&T Technical Journal*, 72(4):27–37, July 1993.

Appendix

In this section we prove the main fairness and delay properties of our algorithm. First, we start with several preliminary results. Lemma 1 gives a upper bound for the lag of an error-free session, while the next three lemmas give bounds for the difference between the virtual times (v_i's), the virtual compensation times (c_i's), and the virtual excess times (f_i's) between any two active sessions.

Lemma 1 *The lag of an error free session is never greater than L_{max}, where L_{max} represents the maximum size of a message.*

Proof. The proof is by induction. From the algorithm in Figures 2 and 3, the lag of an *error-free* session i changes in one of the following three cases: (a) session i becomes active, (b) session i is selected based on its virtual time but since it is leading another session j is selected to receive service, and (c) session i receives service from another session j.

Basic step. When an error-free session i becomes active, its lag is set to zero, and therefore the lemma is trivially true.

Induction step. Assume $lag_i \leq L_{max}$. We consider two cases: (1) $lag_i < 0$, and (2) $0 \leq lag_i \leq L_{max}$. Since in case (1) session i is leading, its lag can increase only when its service is given to another session j (see case (b) above). In this case, we have

$$lag_i = lag_i + l_j^k \leq l_j^k \leq L_{max}, \tag{11}$$

where l_j^k represents the length of the packet at the head of the queue of session j. In case (2), session i is non-leading, and so its lag can only decrease (case (c) above). Thus, the bound holds. □

Lemma 2 *The difference between the virtual times of any two active sessions i and j is bounded as follows:*

$$-\frac{L_{max}}{r_j} \leq v_i - v_j \leq \frac{L_{max}}{r_i}. \tag{12}$$

Proof. The virtual time of a session is updated in one of the following cases: (1) the session becomes active, (2) the session is selected. Again, the proof is by induction.

Basic step. When there is only one active session, the lemma is trivially true.

Induction step. Consider a session i that becomes active at time t, and assume that the lemma is true at any time before t. Then the virtual time of session i is either initialized to the minimum virtual time among all active sessions, or remains the same if it is larger than this minimum. Since virtual times are non-decreasing, it is easy to see that the difference between v_i and the virtual time of any other active session remains in the same bounds. This concludes the argument for case (1).

For case (2), assume again that before session i is selected, the lemma holds. When selected, the virtual time of session i changes as follows

$$v_i = v_i + \frac{l^k}{r_i}, \tag{13}$$

where l^k represents the length of the packet that is served (not necessary a packet of session i) when session i is selected, if any. (If there is no such packet, we assume a dummy packet of length $\delta \ll L_{max}$ is served, the proof proceeds identically.) Since v_i represents the *minimum* virtual time among all currently active sessions, we have

$$v_i \leq v_j, \ \ \forall j \in \mathcal{A}. \tag{14}$$

Since v_i is the only virtual time that changes at time t, it is enough to show that the difference between v_i and any other v_j is bounded. Recall that by hypothesis we have

$$-\frac{L_{max}}{r_j} \leq v_i - v_j \leq \frac{L_{max}}{r_i}, \ \ \forall j \in \mathcal{A}. \tag{15}$$

From this and from Eq. (13) and Ineq. (14) it follows that

$$v_i + \frac{l^k}{r_i} - v_j \leq \frac{l^k}{r_i} \leq \frac{L_{max}}{r_i}, \ \ \forall j \in \mathcal{A}, \tag{16}$$

Similarly, if we assume that j is selected (instead of i), we have

$$v_i - v_j - \frac{l^k}{r_j} \geq -\frac{l^k}{r_j} \geq -\frac{L_{max}}{r_j}, \ \ \forall i \in \mathcal{A}, \tag{17}$$

which concludes the proof. □

Since the proofs of the next two lemmas are similar to that of Lemma 2, we give the results without the proofs.

Lemma 3 *The difference between the virtual compensation times of any two active error-free sessions i and j that are both lagging is bounded as follows:*

$$-\frac{L_{max}}{r_j} \leq c_i - c_j \leq \frac{L_{max}}{r_i} \tag{18}$$

Lemma 4 *The difference between the virtual excess times of any two active error-free sessions i and j that are both non-lagging is bounded as follows:*

$$-\frac{L_{max}}{r_j} \leq f_i - f_j \leq \frac{L_{max}}{r_i} \tag{19}$$

The next lemma gives bounds on the difference between the normalized service received by a leading session i (s_i) and the amount it should have received (αv_i).

Lemma 5 *For any leading error-free session i,*

$$(\alpha - 1)\frac{L_{max}}{r_i} \leq \alpha v_i - s_i \leq \alpha \frac{L_{max}}{r_i}. \tag{20}$$

Proof. The proof is by induction.

Basic step. Initially, when session i becomes leading s_i is initialized to αv_i, and therefore the bounds hold.

Induction step. Assume the bounds hold before v_i and/or s_i are updated. Since v_i and/or s_i change only when session i is selected, we consider two cases: (1) session i is actually served, and (2) another session j is served. According to the algorithm, the first case occurs only when $s_i \leq \alpha v_i$. Therefore, we have,

$$\alpha(v_i + \frac{l_i^k}{r_i}) - s_i - \frac{l_i^k}{r_i} = (\alpha - 1)\frac{l_i^k}{r_i} + \alpha v_i - s_i \geq (\alpha - 1)\frac{L_{max}}{r_i}, \tag{21}$$

where l_i^k represents the length of the packet being transmitted.

In the second case ($s_i > \alpha v_i$), the service is allocated to another session j, if any, v_i is updated but s_i is not. Thus, we have

$$\alpha(v_i + \frac{l_j^k}{r_i}) - s_i < \alpha\frac{l_j^k}{r_i} \leq \alpha\frac{L_{max}}{r_i}, \tag{22}$$

where l_j^k represents the length of the transmiteed packet of session j. $\square$

Theorem 1 *The difference between the normalized service received by any two sessions i and j during an interval $[t_1, t_2)$ in which both sessions are continuously backlogged, error-free, and their status does not change is bounded as follows:*

$$\left|\frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j}\right| \leq \beta\left(\frac{L_{max}}{r_i} + \frac{L_{max}}{r_j}\right), \tag{23}$$

where $W_i(t_1, t_2)$ represents the service received by session i during $[t_1, t_2)$, L_{max} is the maximum packet length, and $\beta = 3$ if both sessions are non-leading, $\beta = 3 + \alpha$ otherwise.

Proof. We consider three cases: both sessions are (1) lagging, (2) satisfied, or (3) leading during the entire interval $[t_1, t_2)$.

(1) (both sessions are lagging) In this case both sessions receives service each time they are selected, or when they receive compensation from a leading session. Since both the virtual time v_i and the compensation virtual time c_i are updated *before* a packet is send, it follows that the total service received by an error-free lagging session during $[t_1, t_2)$ is bounded by

$$\begin{aligned} v_i(t_2) - v_i(t_1) + c_i(t_2) - c_i(t_1) - \frac{L_{max}}{r_i} \quad &\leq \quad \frac{W_i(t_1,t_2)}{r_i} \\ &\leq \quad v_i(t_2) - v_i(t_1) + c_i(t_2) - c_i(t_1) + \frac{L_{max}}{r_i}. \end{aligned} \tag{24}$$

In the left-hand inequality, the term $-\frac{L_{max}}{r_i}$ accounts for the worst case in which t_2 occurs exactly after a packet is selected, while in the right-hand inequality the term $\frac{L_{max}}{r_i}$ accounts for the worst case when t_1 occurs exactly after a virtual time is updated but before the corresponding packet is transmitted. Thus, from the above inequality and by using Lemmas 2 and 3, it is easy to see that

$$\left|\frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j}\right| \leq 3\left(\frac{L_{max}}{r_i} + \frac{L_{max}}{r_j}\right). \tag{25}$$

(2) (both sessions are satisfied) In this case both sessions are served each time they are selected based on their virtual times, or when they receive excess service. Then, similar to the previous case we have

$$v_i(t_2) - v_i(t_1) + f_i(t_2) - f_i(t_1) - \frac{L_{max}}{r_i} \le \frac{W_i(t_1,t_2)}{r_i} \le v_i(t_2) - v_i(t_1) + f_i(t_2) - f_i(t_1) + \frac{L_{max}}{r_i}, \tag{26}$$

and consequently, similarly to the previous case, by using Lemmas 2 and 4, we obtain

$$\left|\frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j}\right| \le 3\left(\frac{L_{max}}{r_i} + \frac{L_{max}}{r_j}\right). \tag{27}$$

(3) (both sessions are leading) Similar to the previous case, the service received by a leading session i during $[t_1, t_2)$ is bounded by

$$s_i(t_2) - s_i(t_1) + f_i(t_2) - f_i(t_1) - \frac{L_{max}}{r_i} \le \frac{W_i(t_1,t_2)}{r_i} \le s_i(t_2) - s_i(t_1) + f_i(t_2) - f_i(t_1) + \frac{L_{max}}{r_i}. \tag{28}$$

Further, according to Lemma 5, for any leading error-free session i and any time t, while it is active, we have

$$\alpha v_i(t) - (\alpha - 1)\frac{L_{max}}{r_i} \ge s_i(t) \ge \alpha v_i(t) - \alpha\frac{L_{max}}{r_i}, \tag{29}$$

Consequently, for any two leading error-free sessions that are active at time t, we have

$$\alpha(v_i(t) - v_j(t)) + \alpha(\frac{L_{max}}{r_j} - \frac{L_{max}}{r_i}) - \frac{L_{max}}{r_j} \le s_i(t) - s_j(t) \le \alpha(v_i(t) - v_j(t)) + \alpha(\frac{L_{max}}{r_j} - \frac{L_{max}}{r_i}) + \frac{L_{max}}{r_i}. \tag{30}$$

From the above inequality and Lemma 2, we obtain

$$-\alpha\frac{L_{max}}{r_i} - \frac{L_{max}}{r_j} \le s_i(t) - s_j(t) \le \alpha\frac{L_{max}}{r_j} + \frac{L_{max}}{r_i}. \tag{31}$$

Finally, from this inequality and Ineq. (28), we have

$$\left|\frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j}\right| \le (3+\alpha)\left(\frac{L_{max}}{r_i} + \frac{L_{max}}{r_j}\right), \tag{32}$$

which concludes the proof of the theorem. □

Theorem 2 *Consider an active lagging session i that becomes error-free after time t. If session i is continuously backlogged after time t, then it is guaranteed to catch up after at most Δ units of time,*

$$\Delta = \frac{\widehat{R}^2}{r_i r_{min}(1-\alpha)R} lag_i(t) + \left(\frac{\widehat{R}(\widehat{R}/r_i + n + 2)}{r_{min}(1-\alpha)} + n + 1 + \frac{\widehat{R}}{r_{min}} \right) \frac{L_{max}}{R}, \tag{33}$$

where n is the number of sessions that are active at any time in $[t, t')$, R is the channel capacity, L_{max} is the maximum length of a packet, $\widehat{R}$ is the aggregate rate of all sessions in the system, and r_{min} is the minimum rate of any session.

Proof. After time t, as long as session i is lagging, its lag decreases each time it receives compensation. Since the total compensation received by session i during the interval $[t, t')$ is $r_i(c_i(t') - c_i(t))$, we have

$$lag_i(t') = lag_i(t) - r_i(c_i(t') - c_i(t)). \tag{34}$$

Let $C(t, t')$ be the total compensation received by all sessions during the interval $[t, t')$, and let $\mathcal{L}(t, t')$ denote the set of all lagging session that have received compensation at some point in the interval $[t, t')$. It is easy to see that during $[t, t')$ the compensation is *always* given to a lagging session. This is because there is at least one continuously lagging session, namely session i, that is error-free during this interval. Clearly, in the worst case, all sessions in $\mathcal{L}(t, t')$ are continuously lagging and error-free (so therefore they can accept compensation at any time) during the interval $[t, t')$. Thus, in this case, we have

$$C(t, t') \leq \sum_{j \in \mathcal{L}(t,t')} r_j(c_j(t') - c_j(t)) + L_{max}. \tag{35}$$

By using Lemma 3 for any two lagging error-free sessions i and j that are active during the interval $[t, t')$ we have

$$c_j(t') - c_j(t) \leq c_i(t') - c_i(t) + \frac{L_{max}}{r_i} + \frac{L_{max}}{r_j}, \tag{36}$$

and therefore

$$
\begin{aligned}
C(t,t') &\leq \sum_{j\in\mathcal{L}(t,t')} r_j(c_i(t') - c_i(t) + \frac{L_{max}}{r_j} + \frac{L_{max}}{r_i}) + L_{max} \\
&= (c_i(t') - c_i(t)) \sum_{j\in\mathcal{L}(t,t')} r_j + L_{max}|\mathcal{L}(t,t')| + \frac{L_{max}}{r_i} \sum_{j\in\mathcal{L}(t,t')} r_j + L_{max} \\
&< (c_i(t') - c_i(t))\widehat{R} + (n + \frac{\widehat{R}}{r_i})L_{max},
\end{aligned} \tag{37}
$$

where n represents the total number of sessions that are active at any time in $[t, t')$, which is at least $|\mathcal{L}(t,t')| + 1$. This is because as long as there is at least a lagging session, there is also at least a leading session. We denote this set of active sessions as A.

Further, note that since the compensation $C(t,t')$ represents a fraction α of the work received by leading sessions, and since this work is proportional to the sessions' rates, it follows that the worst case occurs when there is only one leading session k and this session has rate $r_k = r_{min}$. Thus, in general, we have

$$
C(t,t') \geq r_k(v_k(t') - v_k(t)) - r_k(s_k(t') - s_k(t)) - L_{max}, \tag{38}
$$

Similar to Ineq. (37) we obtain

$$
\begin{aligned}
R(t'-t) &\leq \sum_{j\in\mathcal{A}} r_j(v_j(t') - v_j(t)) + L_{max} \\
&\leq \sum_{j\in\mathcal{A}} r_j(v_k(t') - v_k(t) + \frac{L_{max}}{r_j} + \frac{L_{max}}{r_k}) + L_{max} \\
&< (v_k(t') - v_k(t))\widehat{R} + (n + 1 + \frac{\widehat{R}}{r_k})L_{max}.
\end{aligned} \tag{39}
$$

By combining the above two inequalities, and by using Lemma 5 we get

$$
\begin{aligned}
C(t,t') &\geq r_k(v_k(t') - v_k(t)) - r_k(s_k(t') - s_k(t)) - L_{max} \\
&\geq r_k(v_k(t') - v_k(t)) - r_k(\alpha v_k(t') - \alpha v_k(t) + \frac{L_{max}}{r_k}) - L_{max} \\
&= r_k(1-\alpha)(v_k(t') - v_k(t)) - 2L_{max} \\
&> r_k(1-\alpha)\frac{R(t'-t) - (n + 1 + \widehat{R}/r_k)L_{max}}{\widehat{R}} - 2L_{max}.
\end{aligned} \tag{40}
$$

Now, from Ineqs. (37) and (40) we obtain

$$
\begin{aligned}
c_i(t') - c_i(t) &> \frac{C(t,t') - (n + \widehat{R}/r_i)L_{max}}{\widehat{R}} \\
&> r_k(1-\alpha)\frac{R(t'-t) - (n+1+\widehat{R}/r_k)L_{max}}{\widehat{R}^2} - \frac{(n+2+\widehat{R}/r_i)L_{max}}{\widehat{R}}.
\end{aligned} \tag{41}
$$

Finally, since $lag_i(t')$ is assumed to be no larger than zero, from the above inequality, Ineq. (34), and by taking $\Delta = t' - t$ the proof follows. □

Since during any busy period of a server there is no forced compensation, from the above theorem we have the following result:

Corollary 1 *Consider two sessions i and j backlogged during a service busy period $[t_1, t_2)$, and assume that at time t_1 both sessions have the same normalized lag, i.e., $lag_i(t_1)/r_i = lag_j(t_1)/r_j$. Then, irrespective of the errors experienced by these sessions during the interval $[t_1, t_2)$, if both sessions become error-free after t_2 and they have enough demand, then there exists a time $t_3 > t_2$ such that the difference between the normalized service received by the two sessions during the interval $[t_1, t_3)$ is bounded.*

In the following, we determine the delay bound for an error-free session. In the next two lemmas we give two preliminary results used in proving Theorem 3.

Lemma 6 *Let $W_i(t_1, t_2)$ be the service received by an error-free session during the interval $[t_1, t_2)$ (t_1 and t_2 are packet transmission finish times) while it is continuously active in S, and let $W_i^r(t_1, t_2)$ be the service received by the same session in S^r_{SFQ}. Then, we have*

$$
W_i^r(t_1, t_2) = W_i(t_1, t_2) + lag_i(t_2) - lag_i(t_1). \tag{42}
$$

Proof. The lag and/or the work received by session i in S during $[t_1, t_2)$ change when one of the following events occur: (1) session i is selected and the packet at the head of its queue is transmitted, (2) session i receives service from another session, and (3) session i is leading and its service is given to another session. On the other hand, the service received by session i in S^r_{SFQ} changes only when it is selected and the packet at the head of its queue is transmitted. In the following, we use induction on the events that change the lag and the work received by session i in S.

Basic step. At t_1, Eq. (42) reduces to $W_i^r(t_1, t_1) = W_i(t_1, t_1)$, which is obviously true.

Induction step. Assume that at time $t \in [t_1, t_2)$ one of the above three events occurs, and that for any time smaller than t Eq. (42) holds.

In case (1), when session i is selected and the packet at the head of its queue is served we have

$$\begin{aligned} W_i(t_1, t+) &= W_i(t_1, t) + l_i^k, \text{ and} \\ W_i^r(t_1, t+) &= W_i^r(t_1, t) + l_i^k, \end{aligned} \tag{43}$$

where $t+$ represents the time immediately after the packet has been transmitted. Since according to our algorithm, lag_i does not change in this case, it follows that if Eq. (42) holds at time t, then it will also hold at time $t+$.

In case (2), when session i receives service from another session in S, its lag and work change as follows

$$lag_i(t+) = lag_i(t) - l_i^k, \text{ and} \tag{44}$$

$$W_i(t_1, t+) = W_i(t_1, t) + l_i^k. \tag{45}$$

where again l_i^k represents the packet at the head of session i's queue. However, note that in this case W_i^r is *not* updated (because session i is not selected). Thus, we have

$$\begin{aligned} W_i^r(t_1, t+) &= W_i^r(t_1, t) = W_i(t_1, t) + lag_i(t) - lag_i(t_1) \\ &= W_i(t_1, t) + l_i^k + lag_i(t) - l_i^k - lag_i(t_1) = W_i(t_1, t+) + lag_i(t+) - lag_i(t_1). \end{aligned} \tag{46}$$

Finally, in case (3), session i is selected but its service is given to another session j. If there is no such session j that can send, then we simply assume a packet of a session j of length δ is served (forced compensation), and the proof proceeds identically. If there is such session, then let l_j^k be the length of the packet at the head of session j's queue. Then, we have

$$lag_i(t+) = lag_i(t) + l_j^k, \text{ and} \tag{47}$$

$$W_i^r(t_1, t+) = W_i^r(t_1, t) + l_j^k, \tag{48}$$

while W_i does not change. From this, it follows that

$$W_i^r(t_1,t+) = W_i^r(t_1,t) + l_j^k = W_i(t_1,t) + lag_i(t) + l_j^k - lag_i(t_1) \tag{49}$$
$$= W_i(t_1,t) + lag_i(t+) - lag_i(t_1) = W_i(t_1,t+) + lag_i(t+) - lag_i(t_1),$$

which completes the proof of the lemma. □

From Lemmas 1 and 6 it follows that the difference between the service received by an error-free session in the reference system and the service the session receive in the error system is bounded.

Lemma 7 *Assume an error-free session i becomes active at time t in an error system S. Then, the difference between the service received by i during any time interval $[t,t')$ (t' is a packet transmission finish time) while it remains active in S and the service the session would receive in the reference system S^r_{SFQ} is bounded as follows:*

$$W_i^r(t,t') - W_i(t,t') \leq L_{max}. \tag{50}$$

Proof. Since when session i becomes active at time t, $lag_i(t) = 0$, according to Lemma 6, we have

$$W_i^r(t,t') = W_i(t,t') + lag_i(t'), \tag{51}$$

Further, since session i is assumed to be error-free during the interval $[t,t')$, according to Lemma 1, we have $lag_i(t') \leq L_{max}$, which concludes the proof. □

Since in our case S^r_{SFQ} represents an error-free system where sessions are served by the SFQ policy, the above result suggests that we can use SFQ delay guarantees to bound the packet delay in S. In particular, it has been proved in [5] that the delay of any packet k of a session i under SFQ is bounded by

$$d_i^k \leq e_i^k + (n-1)\frac{L_{max}}{R} + \frac{l_i^k}{R}, \tag{52}$$

where R is the channel capacity, n represents the total number of active sessions, d_i^k represents the k-th packet of session i's departure time, and e_i^k represents the expected arrival time of the k-th packet of session i, and is computed as follows

$$e_i^k = \max\{a_i^k, e_i^{k-1} + \frac{l_i^{k-1}}{r_i}\}, \;\; k > 1, \tag{53}$$

where a_i^k represents the actual arrival time, and $e_i^1 = -\infty$.

Theorem 3 *The delay experienced by the k-th packet of an error-free session i in an error system S is bounded as follows:*

$$d_i^k \leq e_i^k + (n-1)\frac{L_{max}}{R} + \frac{l_i^k}{R} + \frac{L_{max}}{r_i}. \tag{54}$$

Proof. Since by time d_i^k the k-th packet of session i has been transmitted, we have

$$W_i(a_i^1, d_i^k) = \sum_{j=1}^{k} l_i^j, \tag{55}$$

But according to Lemma 7, in the reference error-free system S_{SFQ}^r, we have

$$W_i^r(a_i^1, d_i^k) \leq L_{max} + \sum_{j=1}^{k} l_i^j. \tag{56}$$

Thus, in the worst case the work that session i need to receive in the error-free reference system S_{SFQ}^r until the k-th packet of session i in the error system is transmitted is at most $L_{max} + \sum_{j=1}^{k} l_i^j$. Consequently, according to Eq. (53) the expected arrival time of the k-th packet of session i in S_{SFQ}^r, denoted $e_i^{k,r}$ is bounded by:

$$e_i^{k,r} \leq e_i^k + \frac{L_{max}}{r_i}. \tag{57}$$

From the above equation and Eq. (52) the proof follows. □

Finally, the next result gives the delay bound for an error-free session i, whose traffic conforms to the leaky-bucket constraints (σ_i, r_i) where σ_i is the bucket depth and r_i is the token rate. Since in this case, for any packet k of session i we have $e_i^k \leq a_i^k + \sigma_i/r_i$, from the above theorem the corollary below follows.

Corollary 2 *Consider an error-free session i with a reserved rate r_i and its traffic conforms to a leaky-bucket (σ, r_i). Then the deadline experienced by the k-th packet of session i is bounded as follows:*

$$d_i^k \leq a_i^k + (n-1)\frac{L_{max}}{R} + \frac{l_i^k}{R} + \frac{L_{max}}{r_i} + \frac{\sigma_i}{r_i}, \tag{58}$$

where a_i^k *represents the arrival time of that packet,* L_{max} *represents the maximum size of a packet,* R *represents the server's rate, and* n *represents the number of active sessions.*