Packet Fair Queuing Algorithms for Wireless Networks

You-Chiun Wang and Yu-Chee Tseng

Abstract—Due to the rapid growth of wireless data services, the issues of providing different quality of services and fair channel access have become more and more crucial. In wireline networks, there are many mature fair queuing algorithms that can provide fairness and bounded delay properties. However, they cannot be directly applied to wireless networks because of the bursty and location-dependent error characteristics of wireless channels. So many fair queuing algorithms designed for wireless networks are proposed. In this chapter, we introduce the concept of fair queuing in wireline networks, discuss the problems and difficulties to apply them to wireless networks, and provide a comprehensive survey of recent research on wireless fair queuing algorithms.

Index Terms—Fair queuing, quality of service (QoS), scheduling, wireless networks.

1 Introduction

In recent years, there has been huge growth in the wireless networking industry. With the rapid growth and usage of wireless data services and the increasing demand for real-time applications, such as VoIP and video conferencing, the issues related to providing quality of service (QoS) and fair channel accessing among multiple packet flows over a shared and bandwidth-limit wireless channel have become more and more important.

In wireline networks, fair queuing has long been used to provide fairness and bounded delay access [1]. The earliest representative fair queuing algorithm is *Weight Fair Queuing (WFQ)* [2]. WFQ is an approximation algorithm designed for providing fairness and delay bound properties to emulate the *Generalized Processor Sharing (GPS)* scheme [3], which is based on a fluid model but cannot be practiced in the real world, at the packet level. Many other packet fair queuing (PFQ) algorithms, such as *Start-time Fair Queuing (SFQ)* [4], *Self-Clocked Fair Queuing (SCFQ)* [5], and *Worst-case Fair Weighted Fair Queuing (WF^2Q)* [6], are also proposed to approximate the services of GPS.

Many researchers have observed that it is not a trivial work to directly apply these PFQ algorithms to the wireless domain. This is because of the bursty and location-dependent channel error characters of wireless networks [7]–[9]. Bursty errors may break a flow's continuous services, while location-dependent channel errors may cause error-free flows to receive more services than they should, thus violating the fairness and delay bound properties.

To solve these problems, many wireless PFQ algorithms have been proposed to take into account the wireless environment's special characteristics [10]–[17]. The basic idea behind these algorithms is to provide a compensation mechanism, which gives additional services to those flows who do not receive their normal share due to channel errors. The major difference between these algorithms is their compensation model.

This chapter presents a survey of fair queuing algorithms for wireless networks. Section 2 introduces the network architecture and system model. Section 3 gives a review of fair queuing algorithms for wireline networks. Section 4 discusses several fair queuing algorithms for wireless networks. A summary of these algorithms is in Section 5. Finally, conclusions are given in Section 6.

The authors are with the Department of Computer Science, National Chiao-Tung University, Hsin-Chu, 30010, Taiwan.
E-mail: {wangyc, yctseng}@cs.nctu.edu.tw

2 System Model

The concept of fair queuing (FQ) is originally developed for routers [18]. Traditionally, routers usually maintain a single queue for each output port, and apply a first-in-first-out (FIFO) queuing discipline to serve each queue (refer to Fig. 1(a)). When a new packet arrives, it is placed at the end of the queue. As long as the queue is not empty, the router transmits the head-of-line (HOL) packet from the queue.

However, the main drawback of the FIFO queuing discipline is unfairness. A greedy flow can generate many packets to occupy the queue, and thus block other flows' transmission opportunity. In other words, most services are grabbed by these greedy flows. To overcome this problem, most FQ solutions try to maintain one queue for each source (refer to Fig. 1(b)), and each incoming packet is placed in an appropriate queue. The system serves these queues in a round-robin-like fashion by taking one packet from each nonempty queue in turn. Empty queues are skipped over. This is basically fair because every busy flow can send exactly one packet in each round. If a greedy flow generates too many packets, these packets will be kept in its queue, thus causing longer delays. Since every flow has its own queue, greedy flows will no longer affect other flows.

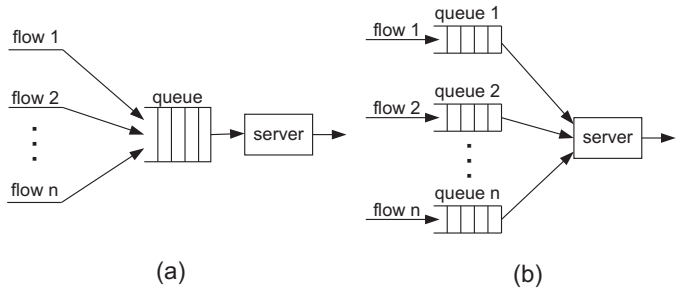

Fig. 1: The FIFO queuing and fair queuing models. (a) FIFO queuing. (b) Fair queuing.

To consider the FQ problem in wireless environments, we will deal with a base station which is responsible for scheduling both uplink and downlink traffic between mobile stations and itself. Typically, the base station should have full knowledge of all downlink queues. We assume that mobile stations will send their queue status to the base station, so scheduling of uplink traffic is also possible.

In a wireless environment, stations may perceive different fading patterns and interference, so errors are location-dependent [7]. Wireless channels are subject to bursty errors too. A two-state discrete Markov channel model [19] is widely used to model a wireless channel (refer to Fig. 2). A station can be in two states: *good* (*error-free*) and *bad* (*error*). The transition probability p_g is the probability that the next time slot is good given that the current slot is bad, and p_b is the probability that the next time slot is bad given that the current slot is good. Then the steady-state probabilities P_G and P_B of being in the good and bad states, respectively, are given by [10]

$$P_G = \frac{p_g}{p_g + p_b}, \quad P_B = \frac{p_b}{p_g + p_b}.$$

Note that if a station is in a bad state, packets transmitted to it will be corrupted with a high probability.

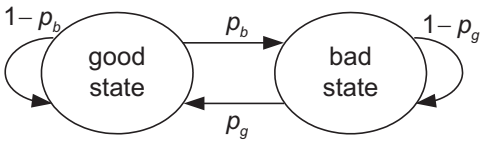

Fig. 2: Two-state discrete Markov channel model.

3 A Review of Wireline Fair Queuing Algorithms

In this section, we review several important fair queuing algorithms designed for wireline networks. We first discuss the Generalized Processor Sharing (GPS) scheme [3], which is designed based on an ideal model, followed by the Weighted Fair Queuing (WFQ) scheme [2], whose goal is to emulate the service of GPS as closely as possible.

3.1 Generalized Processor Sharing (GPS)

A round-robin scheduling may seemingly guarantee fariness. Unfortunately, it does not satisfy the criteria of fair queuing because shorter packets are penalized if there exist longer packets. If every queue transmits one packet in each round, shorter packets will have to wait longer. To overcome this problem, the *bit-round fair queuing (BRFQ)* [20] is proposed, which takes packet lengths into account to schedule packets. BRFQ is designed based on *processor sharing (PS)*, where only one bit from each queue is transmitted in each round. So longer packets no longer take advantage over shorter packets, and each busy flow can receive exactly the same amount of services.

However, BRFQ is still not satisfactory because to support quality of service (QoS), different requirements from different flows should be taken into account. Therefore, the concept of *generalized processor sharing (GPS)* [3] is proposed, which allows us to specify the different service demands of individual flows. A GPS server operates at a fixed rate r. It is *work-conserving* in the sense that it is never idle whenever there are packets to be transmitted. A GPS server serving N flows is characterized by N positive real numbers, $\phi_1, \phi_2, ..., \phi_N$, which can reflect their relative service demands. Let $W_i(t_1, t_2)$ be the amount of traffic served for flow i in the time interval $[t_1, t_2]$. Then for any flow i that is continuously backlogged (i.e., with a non-empty queue) during $[t_1, t_2]$, it is guaranteed that

$$\frac{W_i(t_1, t_2)}{W_j(t_1, t_2)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1, 2, ..., N. \tag{1}$$

Let $B_{GPS}(t)$ be the set of backlogged flows at time t. By Eq. (1), if $B_{GPS}(t)$ is unchanged during $[t_1, t_2]$, the service rate of flow i during this interval will be exactly

$$r_i^*(t_1, t_2) = \frac{\phi_i}{\sum_{j \in B_{GPS}(t_1)} \phi_j} r.$$

Since $B_{GPS}(t_1)$ is a subset of all flows at the server, it follows that $r_i^*(t_1, t_2) \geq r_i$, where

$$r_i = \frac{\phi_i}{\sum_{j=1}^{N} \phi_j} r.$$

Intuitively, flow i will be guaranteed a minimum service rate of r_i during any time interval when it is continuously backlogged. Besides, for a leaky-bucket-constrained source [21], with a suitable choice of parameters, a network with GPS servers can provide worst-case, end-to-end delay bounds [3].

3.2 Weighted Fair Queuing (WFQ)

The aforementioned GPS scheme is too idealized because it is a fluid model. It assumes that the server can serve all backlogged flows simultaneously and that the traffic is infinitely divisible. Most networks transmit in packet level. Therefore, many packet fair queuing (PFQ) algorithms have been proposed to approximate GPS. Weight fair queuing (WFQ) [2] is a representative scheme to approximate GPS. In WFQ, each packet is associated with a *start tag* $S_i(\cdot)$ and *finish tag* $F_i(\cdot)$, which correspond to the virtual times at which the first and the last bits of the packet, respectively, are served in GPS. At the time when the kth $(k \geq 1)$ packet of flow i, denoted by p_k^i, arrives at the queue, we mark its start tag and finish tag as follows:

$$\begin{aligned} S_i(p_k^i) &= \max\left\{V(A(p_k^i)), F_i(p_{k-1}^i)\right\}, \\ F_i(p_k^i) &= S_i(p_k^i) + \frac{l_k^i}{r_i}, \end{aligned}$$

where $F_i(p_0^i) = 0$, $A(p_k^i)$ is the physical arrival time of packet p_k^i, l_k^i is the length of p_k^i, and r_i is the weight of flow i. Here $V(t)$ is the virtual time at real time t to denote the current round of services in GPS. The progression of $V(t)$ is defined as

$$\frac{dV(t)}{dt} = \frac{C(t)}{\sum_{i \in B(t)} r_i},$$

where $C(t)$ is the link capacity at time t and $B(t)$ is the set of backlogged flows. WFQ then schedules packets in the increasing order of their finish tags.

By simulating the fluid model of GPS, WFQ has the property that the worst case packet delay of a flow compared to GPS is upper bounded by one packet [7].

4 WIRELESS FAIR QUEUING ALGORITHMS

Although GPS and WFQ can provide both guaranteed and fair services in wireline networks, they may fail in wireless networks. One of the main reasons is that wireless networks' channel condition is *location-dependent*. In wireline networks, if a channel is in error state, all backlogged flows cannot be served, and the virtual times of all flows will be frozen. Since no flow is served, the fairness property is still maintained. However, in wireless networks, some stations may experience bad channels, while others may not. Since the system is work-conserving, the scheduler will dispatch the services that should have been given to those error flows, to other error-free flows. During this period, error-free flows will receive more services than they deserve. This may result in large differences between the virtual times of error flows and error-free flows.

Let flow i be an error flow. The above discussion may lead to two dilemmas [11]:

- If packets of flow i are allowed to keep their original tags, they may have smaller start/finish tags compared to other error-free flows' packets. When flow i returns to normal, the scheduler may select flow i for service exclusively until its start/finish tag catches up with those of other flows. During this period, other flows may starve, thus leading to sudden unacceptable delays.
- If we update tags of flow i as it is in error, then other error-free flows will not be penalized. However, since the history is erased, flow i may never get back its lost services, causing unfair phenomenon.

These observations motivate us to redesign FQ algorithms for wireless networks. The wireless FQ algorithms that we will discuss are classified into four classes. The first class of algorithms still maintains the concept of wireline FQ algorithms, but appropriately swaps the channel access between backlogged flows that perceive channel errors and backlogged flows that do not. The second class of algorithms explicitly provide a mechanism to compensate those flows who receive less services due to channel errors. The third class of algorithms dynamically adjusts a flow's weight to adapt to its channel or queue condition. The last class of algorithms considers other application-specific issues, such as traffic types of flows and handoff between base stations.

4.1 Algorithms with an Error-free Service Model

Algorithms in this class usually define an *ideal* fair service model which assumes no channel errors. This error-free service model provides a reference for channel allocation. By this error-free service model, the scheduler can realize how much service that flows should receive in an ideal error-free channel environment, and then compensate those flows who receive less services due to channel errors.

The representative algorithms in this class include *IWFQ (Idealized Wireless Fair Queuing)* [10] and *CIF-Q (Channel-condition Independent Fair Queuing)* [11], which are discussed as follows.

4.1.1 Idealized Wireless Fair Queuing (IWFQ)

In [10], the authors propose a fluid model called *wireless fluid fair queuing (WFFQ)*, which is a modified version of GPS that can adapt to wireless environment. Like GPS, WFFQ cannot be practiced in the real system, so a packet-based scheduling algorithm, IWFQ, is used to approximate WFFQ. Since WFFQ is based on an unrealistic fluid model and is mainly used for the theoretical analysis, we will focus on discussing its packet-based scheduling scheme, IWFQ.

IWFQ uses WFQ [2] as its reference error-free model, which helps it to understand the state of flows and determines the service sequence. A flow is said to be *leading*, *lagging*, or *in sync* if its queue size in the real error-prone system is smaller than, larger than, or the same as the queue size in the reference error-free system, respectively. In IWFQ, the tags of packets are computed similarly to that in WFQ. When the nth packet of flow i (denoted by $t_{i,n}$) arrives at the system, it is assigned with two tags: start tag $s_{i,n}$ and finish tag $f_{i,n}$:

$$s_{i,n} = \max\{v(A(t_{i,n})), f_{i,n-1}\}, \quad (2)$$

$$f_{i,n} = s_{i,n} + \frac{L_{i,n}}{r_i}, \quad (3)$$

where $A(t_{i,n})$ is the physical arrival time of packet $t_{i,n}$, $v(A(t_{i,n}))$ is the system virtual time at time $A(t_{i,n})$ defined in IWFQ, $L_{i,n}$ is the packet size of the nth packet of flow i, and r_i is the weight of flow i. The IWFQ scheduler always selects the packet with the smallest finish tag for service.

When there is no flow suffering from errors, IWFQ works the same as WFQ. The difference between WFQ and IWFQ is: when the selected packet cannot be transmitted due to channel errors, IWFQ chooses the packet with the next smallest finish tag for service. This process will continue until the scheduler finds a packet that can be transmitted.

Since the finish tags of packets, once assigned, will never be changed later on, a flow which loses its services due to errors may accumulate many packets with small finish tags after exiting from errors. To prevent these packets from starving other flows, IWFQ sets up the following bounds:

- *Bounds on lag*: The total lag kept by all flows is bounded by a constant of B bits. A lagging flow i with weight r_i is allowed to compensate no more than b_i bit, where
$$b_i = B \times \frac{r_i}{\sum_{k \in F} r_k},$$
and F is the set of all flows. For any lagging flow i, the total length of those packets with finish tags less than the current system virtual time is bounded by b_i bits. All other such packets will be deleted.
- *Bounds on lead*: A leading flow is allowed to keep its lead by a maximum of l_i bits. That is, for each leading flow i, if the start tag $s_{i,hol}$ of the HOL packet is greater than the current system virtual time $v(t)$ by more than l_i/r_i, then its start and finish tags will be updated as:
$$s_{i,hol} = v(t) + \frac{l_i}{r_i},$$
$$f_{i,hol} = s_{i,hol} + \frac{L_{i,hol}}{r_i},$$
where $L_{i,hol}$ is the length of the packet. The tags of other packets of flow i are also updated by Eqs. (2) and (3) accordingly.

The IWFQ scheduler checks every queue after it transmits a packet. If the scheduler finds any queue that violates

the bounding principles, it adjusts the queue following the above two rules so as to ensure delay bound and throughput. Analytical results of IWFQ can be found in [10].

4.1.2 Channel-condition Independent Fair Queuing (CIF-Q)

In CIF-Q, a set of channel-condition-independent properties that a wireless fair queuing algorithm should have is proposed:

- *Delay and throughput guarantees*: Delay bound and throughput for error-free flows should be guaranteed and should not be affected by other flows that are in error.
- *Long-term fairness*: After a flow exits from errors, as long as it has enough service demand, it can get back all lost services during the period of errors.
- *Short-term fairness*: The difference between the normalized services received by any two error-free flows that are continuously backlogged and are in the same state (*leading*, *lagging*, or *satisfied*) during any time interval should be bounded.
- *Graceful degradation*: During any time interval while it is error-free, a leading backlogged flow should be guaranteed to receive at least a minimum fraction of services.

CIF-Q algorithm is derived based on these guidelines. CIF-Q selects the start-time fair queuing (SFQ) [4] as its reference error-free system to determine the service order. CIF-Q maintains a virtual time v_i for each flow i to keep track of the normalized services it received in the reference error-free system, and a parameter lag_i is used to record the difference between services that is actually received by flow i in the real error-prone system and services that is expected to be received by it in the reference error-free system. When a flow i is selected and transmits a packet with length l_p, v_i is increased by l_p/r_i, where r_i is the weight of flow i. However, when the selected flow i cannot send packets, the scheduler chooses another flow j to serve. At this time, v_i is still increased by l_p/r_i, but lag_i and lag_j are increased and decreased by l_p, respectively. Therefore, according to lag_i, we can determine the state of flow i. Flow i is called *leading* if $lag_i < 0$, called *lagging* if $lag_i > 0$, and called *satisfied* otherwise. Moreover, since CIF-Q is work-conserving, the algorithm has to maintain $\sum_{i \in A} lag_i = 0$ at all time, where A is the set of all active flows. Besides, in order to satisfy the graceful degradation property, an extra parameter α is used to define the minimal fraction of services which can be received by a leading flow. That is, a leading flow i is allowed to continue receiving services at an average rate of $\alpha \cdot r_i$. So leading flows will not be starved under the CIF-Q scheduling policy.

The overall process of CIF-Q is summarized as follows:

- The scheduler always selects a flow i with the smallest v_i for service. Flow i cannot transmit if either one of the following cases occurs: (a) It suffers from channel errors. (b) It is leading and has received more than $\alpha \cdot r_i$ amount of services. In the latter case, flow i has to give up its transmission opportunity to other lagging flows.
- When a flow i is selected but it cannot send packets, the scheduler chooses another flow j to serve. Such services obtained from flow i is called *additional services*.
- When there are additional services available, lagging flows always have a higher priority to receive such services. The additional services are distributed among lagging flows proportional to their service weights.
- If no lagging flow can enjoy the additional services and such services are obtained from an error flow (case 1 (a)), the scheduler distributes these services among leading and satisfied flows proportional to their service weights.

By applying these rules, the aforementioned guidelines can be satisfied. The detailed implementation can be found in [11]. It has been proven that the normalized services received by any two flows i and j during any time interval $[t_1, t_2)$ during which both flows are continuously backlogged and error-free is bounded by the following inequality:

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

where $W_i(t_1, t_2)$ represents the service received by flow i during $[t_1, t_2)$, L_{max} is the maximum packet length, and $\beta = 3$ if both flows are lagging or satisfied, and $\beta = 3 + \alpha$ if both flows are leading. As for delay bound, CIF-Q guarantees that for any active lagging flow i that becomes error-free after time t, if it is continuously backlogged after time t, then it can catch up with its services within Δ time units, where

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

where n is the number of active flows, R is the channel capacity, $L_{\max}$ is the maximum length of a packet, $\widehat{R}$ is the aggregate rate of all flows in the system, and $r_{\min}$ is the minimum rate of any flow.

4.2 Algorithms with Explicit Compensation Mechanisms

This type of algorithms usually uses an explicit counter (or server) to compensate those flows who receive less services due to errors. When a flow suffers from errors and cannot be served, the scheduler will record the *lost* service of this flow in the counter. After this flow recovers from errors, the scheduler compensates this flow with the amount of services recorded in the counter. Here we introduce two such schemes: *SBFA (Server Based Fairness Approach)* [12] and *Havana* [13].

4.2.1 Server Based Fairness Approach (SBFA)

The basic idea of SBFA is to reserve part of network bandwidth for compensation. The reserved service is called the *long-term fairness server (LTFS)*. LTFS is a *virtual* data flow created for compensation purpose. Although LTFS is a virtual flow, it still shares the bandwidth and is scheduled by the system.

In SBFA, every data flow (except for LTFS) is associated with two queues: *packet queue* (PQ) and *slot queue* (SQ). Whenever there is a packet of flow i arriving, it is inserted

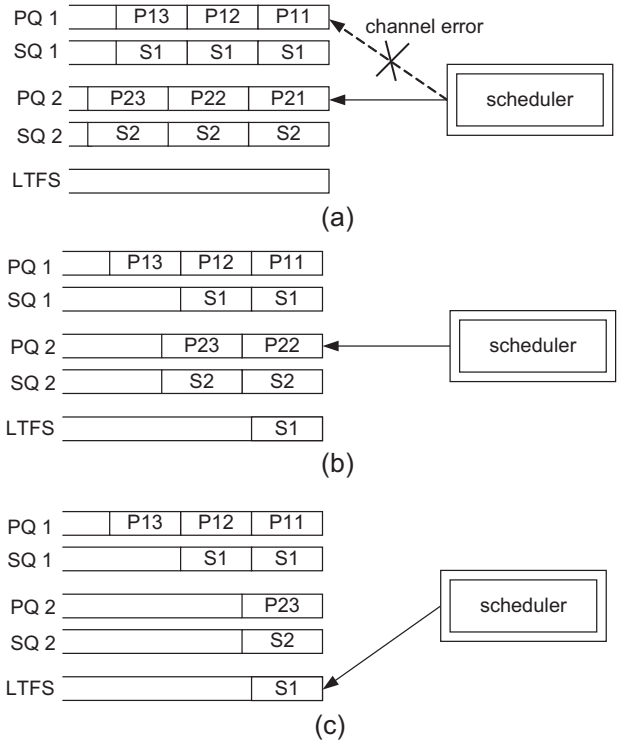

Fig. 3: An example of SBFA using a simple round-robin scheduling policy. (a) Time 0. (b) Time 1. (c) Time 2.

into PQ_i. At the same time, an abstract entity called *slot* is created in SQ_i with a tag i. In SBFA, any wireline packet scheduling algorithm can be used, and the policy is applied to slot queues. More precisely, when a slot in SQ_i is chosen, the HOL packet in corresponding PQ_i is transmitted as long as flow i is error-free. In this case, both the slot in SQ_i and the packet in PQ_i will be removed.

To demonstrate how SBFA works, we use the example in Fig. 3, which adopts a simple round-robin policy. There are two flows $f1$ and $f2$ in the system. Flow $f1$ suffers from errors during time $[0, 1]$, but flow $f2$ is always error-free. At time 0 (refer to Fig. 3(a)), the scheduler turns to serve $f1$, but it finds that $f1$ suffers from errors. So it dequeues a slot from SQ_1 and inserts it into LTFS. Then the scheduler transmits P21 in PQ_2 of flow $f2$. Also, a slot in SQ_2 is removed (refer to Fig. 3(b)). At time 1, by the round-robin policy, the scheduler will try to serve $f2$. This will succeed, so the first entry in both PQ_2 and SQ_2 will be removed. At time 2, the scheduler will try to serve LTFS. It finds that the HOL slot in LTFS has a tag of 1, so the scheduler goes to serve $f1$ (refer to Fig. 3(c)). At this time, only the packet P11 in PQ_1 is removed. Note that if $f1$ is still in error state at time 2, the slot of $f1$ will be kept in LTFS and the scheduler will turn to select another error-free flow to serve. So $f1$ can still be compensated next time.

To summarize, LTFS provides a mechanism to record which flows need to be compensated in the future. There can be more than one LTFS in SBFA, and the number of LTFS depends on the requirements of the flows sharing the wireless link. SBFA suggests that it is better to assign flows with similar requirements to the same LTFS. SBFA can work with other wireline scheduling algorithms too, such as WFQ [2], hierarchical packet fair queueing (H-PFQ) [22], and hierarchical fair service curve (HFS-C) [23].

4.2.2 Havana

In Havana, a modified version of *Deficit Round Robin (DRR)* [24] scheduler is used to credit and compensate flows in response to potential unfairness experienced by mobile stations due to different channel conditions.

In DRR, every flow has its own queue. The scheduler serves queues in a round-robin fashion. In each round, the number of packets that can be served in each queue i is determined by two parameters: *quantum* (Q_i) and *deficit counter* (DC_i). Q_i accounts for the quota given to flow i in each round. DC_i keeps track of the credit already accumulated by flow i. A *round* is the process of visiting each queue once by the scheduler. At the beginning of each round, a credit of Q_i is added to DC_i. When the scheduler turns to serve flow i, the scheduler will transmit the first j packets of queue i such that their total length does not exceed DC_i and j is the largest possible. After the transmission, DC_i is deducted by the exact amount of data being sent. If the scheduler finds that there is no backlogged packet in flow i, DC_i is reset to zero.

To compensate those flows who lose their services due to errors, Havana maintains an extra *compensation counter* (CC_i) to keep track of the amount of lost services of each flow i. If the scheduler defers transmission of an error flow i, the corresponding DC_i is decreased by the Q_i, but CC_i is increased by Q_i. At the beginning of each round, an amount of $\alpha_i \cdot CC_i$ credits is added to DC_i, and CC_i is decreased by the same amount, where $0 < \alpha_i \leq 1$. The value of α_i represents the fraction of the compensation credit actually given to flow i in this round.

An example is in Fig. 4. There are two flows in the system with $Q_1 = 50$, $Q_2 = 100$, $\alpha_1 = 1/2$, and $\alpha_2 = 1/3$. The status of flows at the beginning of round k is shown in Fig. 4(a). The number in each packet represents its size. Assume that flow 2 is error-free but flow 1 suffers from channel errors at round k. Then in round k, the transmission of flow 1 is deferred and that of flow 2 will proceed. Fig. 4(b) shows flows' status at the end of round k. DC_1 is decreased by Q_1 and CC_1 is increased by Q_1. DC_2 is decreased by 100 and CC_2 is unchanged. At the beginning of round $k+1$, as shown in Fig. 4(c), flow i's DC_i is increased by the value of $Q_i + \alpha_i \cdot CC_i$, and CC_i is decreased by $\alpha_i \cdot CC_i$.

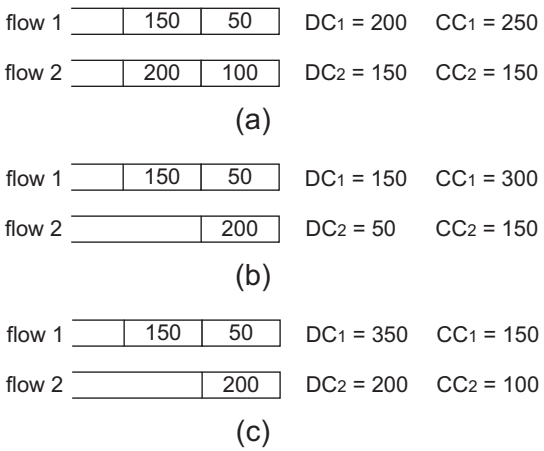

Fig. 4: An example of Havana scheduling ($Q_1 = 50$, $Q_2 = 100$, $\alpha_1 = 1/2$, and $\alpha_2 = 1/3$). (a) Beginning of round k. (b) End of round k. (c) Beginning of round $k + 1$.

Note that to avoid unbounded compensation, upper limits are imposed on deficit counters. That is, the credit

accumulated in DC_i should not exceed an upper bound DC_i^{max} at any time.

4.3 Algorithms that Use Weight Adjustment

Up to now, the weights of flows in the algorithms that we have reviewed are all considered static. That is, once the weight of a flow is decided, it is not changed during the whole scheduling time. Several algorithms take a different viewpoint. They dynamically adjust flows' weights according to different conditions. We will discuss two such algorithms: *Effort-Limit Fair (ELF) scheduling algorithm* [14] and *Fair queuing with adaptive compensation (AC-FQ)* [15].

ELF is proposed to extend WFQ [2] via a dynamic weight-adjustment mechanism. The ELF scheduler will adjust each flow's weight in response to the error rate of that flow, up to a maximum weight defined by that flow's *power factor*, which is also provided by the admission control module (refer to Fig. 5). More precisely, assume that there are n flows sharing a wireless channel. Each flow i maintains a weight r_i and a power factor p_i. Also, let flow i experience an error rate e_i, $0 \leq e_i < 1$. Then ELF defines the adjusted weight a_i of flow i as

$$a_i = \min\left(\frac{r_i}{1-e_i}, p_i \times r_i\right).$$

Intuitively, for an error flow i, the ELF scheduler scales its weight r_i to make up its loss due to channel errors, but we limit the adjustment to a factor p_i. By this dynamic weight adjustment, ELF can control the scheduler's behavior in the presence of channel errors.

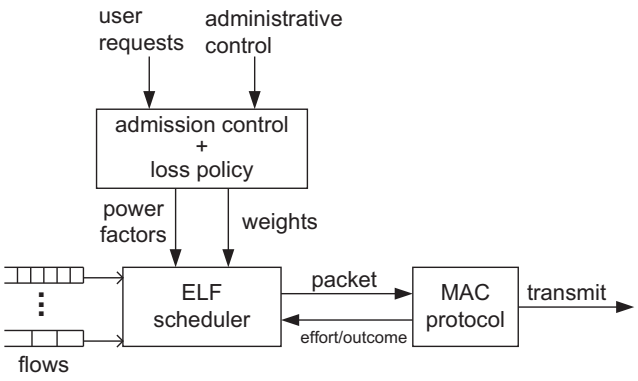

Fig. 5: The concept of the ELF scheduler.

Based on the observation in [25], which points out that CIF-Q may cause the services of error-free flows becoming deteriorated, AC-FQ [15] is proposed. AC-FQ modifies the way how leading flows give up their services for compensating other lagging flows in CIF-Q. The design architecture of AC-FQ is shown in Fig. 6. In AC-FQ, when a flow becomes leading, it is divided into two flows: *transmission flow* and *compensation flow*. A leading flow can receive a fraction of services through the transmission flow, and provide part of bandwidth to other lagging flows through its compensation flow. Each flow i maintains three different weights r_i, r_i^H, and r_i^L, where $r_i > r_i^H > r_i^L$. When a leading flow i has a larger queue length, the weights of its transmission flow and compensation flow are set to r_i^H and $r_i - r_i^H$, respectively. So flow i can keep more services, and its queuing delay can be alleviated. Otherwise, the weights of its transmission flow and compensation flow are set to r_i^L and $r_i - r_i^L$, respectively. In this case, flow i has to give up more services to compensate other lagging flows. In a word, AC-FQ alleviates the queuing delays of leading flows by dynamically changing their weights according to their queue lengths.

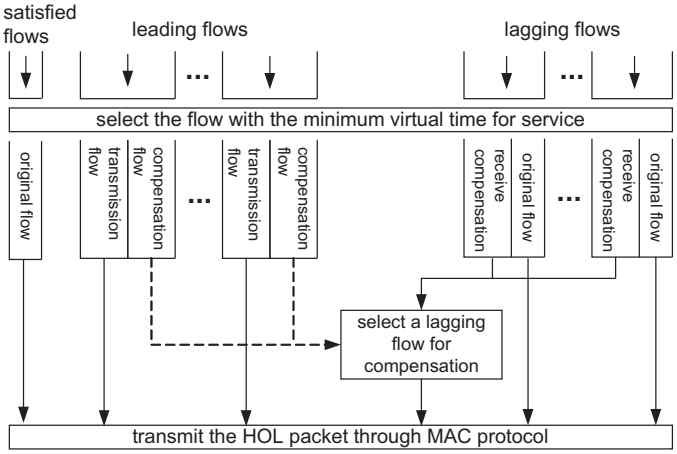

Fig. 6: The design architecture of AC-FQ.

4.4 Algorithms that Consider Traffic Types

An inherent limitation of GPS is that the delay observed by the packets of a flow is tightly coupled with the fraction of the channel given to the flow among all backlogged flows [16]. So those algorithms who modify wireline fair queuing without considering traffic types of flows, such as IWFQ and CIF-Q, may suffer from this limitation too. Therefore, several algorithms, which take traffic types of flows into consideration when scheduling packets, are proposed to decouple delay and bandwidth requirements of flows. Here we discuss three algorithms in this classification: *Wireless Fair Service (WFS)* [16], the *handoff compensation scheme (HCS)* [17], and *Traffic-Dependent wireless Fair Queuing (TD-FQ)* [26].

WFS assigns each flow i with a *rate weight* r_i and a *delay weight* Φ_i, and associates each packet p_i^k with a start tag $S(p_i^k)$ and a finish tag $F(p_i^k)$ as follows:

$$\begin{aligned} S(p_i^k) &= \max\{V(A(p_i^k)), S(p_i^{k-1}) + \frac{L_i^{k-1}}{r_i}\}, \\ F(p_i^k) &= S(p_i^k) + L_i^k/\Phi_i, \end{aligned}$$

where L_i^k is the length of the kth packet of flow i, $A(p_i^k)$ is the arrival time of this packet, and $V(t)$ is the virtual time at time t. Essentially, flow i is drained into the scheduler according to the rate weight r_i, but served according to the delay weight Φ_i. Besides, at each time t, the WFS scheduler transmits the packet with the minimum finish tag among those packets whose start tag is not greater than $V(t) + \varrho$, where ϱ provides a measure of the number of packets over which the scheduler can locally switch the schedule of packets without disrupting the long-term rate. By varying ϱ, WFS can provide different levels of delay-bandwidth decoupling.

In [17], a compensation scheme to handle the channel errors and handoff is proposed. Any fair queuing algorithm can be used as a basic scheduling algorithm here, and the proposed compensation scheme is executed when a channel error or a handoff occurs. This compensation scheme is

designed based on a priority swapping and a compensation flow. A flow experiencing channel errors defers transmission by swapping its resource to other error-free flows, and receives additional resource when it recovers from errors in the future. A compensation flow is a virtual flow with a pre-assigned weight r_{CS}. It participates in scheduling and redistributes its resources to active flows according to the state of flows. All flows are classified into four groups: *poor*, *poorer*, *rich*, and *normal*. A flow is said to be *poor* if it receives less services than it expects. When a poor flow transmitting real-time traffic is about to drop packets due to long waiting time, this flow is changed to a *poorer* flow. When there are additional services available, the poorer flows always have the highest priority to receive such services, so the queuing delays of real-time flows can be alleviated. As for handoff flows, it uses the compensation flow to give/receive weights to/from handoff flows. That is, when a flow i handoffs into a cell, it will be given a weight r_i from the compensation flow if $\alpha \cdot r_{CS} \geq r_i$. On the contrary, when flow i leaves a cell, it returns its service share to the scheduler by increasing r_{CS} by r_i, so other flows can share the unused services left by flow i.

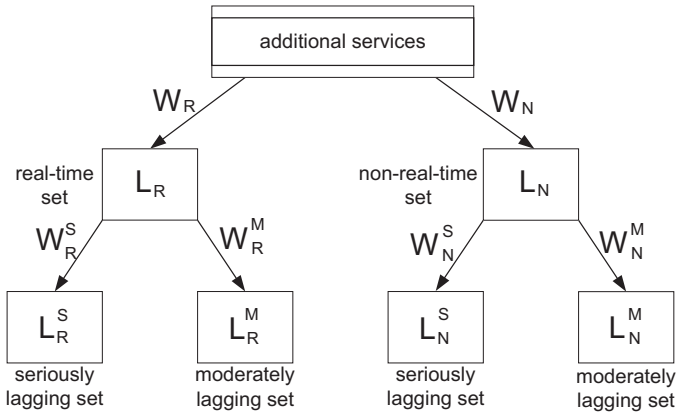

Fig. 7: Structure of the class-based weight compensation (CWC) mechansim.

TD-FQ is developed based on CIF-Q [11], but it adds extra mechanisms to reduce queuing delays of real-time flows by giving them higher priorities. The main characteristic of TD-FQ is that it proposes a *class-based weight compensation (CWC)* mechanism to dispatch additional services to lagging flows. The structure of CWC is shown in Fig. 7. CWC first divides lagging flows into a real-time set and a non-real-time set. These sets are each further divided into a seriously lagging set and a moderately lagging set according to the lagging degrees of the corresponding flows. Individual flows are at the bottom. The concept of weight is used to dispatch additional services to these sets. By classifying lagging flows according to their traffic types and giving a larger weight to the real-time set, TD-FQ can alleviate queuing delays of real-time flows. Besides, since the non-real-time set can still receive a fraction of additional services, it is guaranteed that non-real-time lagging flows will not be starved in TD-FQ scheduling policy.

Here we use an experiment to compare TD-FQ with CIF-Q. In this experiment, we mix real-time and non-real-time traffics together. We observe the packet dropping ratios and queuing delays of real-time flows in TD-FQ and CIF-Q. Eight flows are used, as shown in Table 1. The first six flows are real-time flows, which have two traffic models:

flow	assigned rate	packet size	error scenario
CBR1	256 Kb/s	1 Kb	T_{good} = 6 sec., T_{bad} = 1.5 sec.
CBR2	256 Kb/s	1 Kb	T_{good} = 5 sec., T_{bad} = 0.5 sec.
CBR3	512 Kb/s	2 Kb	No error occurs
voice1	32 Kb/s	1 Kb	T_{good} = 6 sec., T_{bad} = 1.5 sec.
voice2	32 Kb/s	1 Kb	T_{good} = 5 sec., T_{bad} = 0.5 sec.
voice3	64 Kb/s	2 Kb	No error occurs
FTP1	2 Mb/s	4 Kb	T_{good} = 6 sec., T_{bad} = 1.5 sec.
FTP2	2 Mb/s	4 Kb	No error occurs

TABLE 1: Traffic specification of the flows used in the experiment in Table 2 and Table 3.

flow	CBR1	CBR2	CBR3	voice1	voice2	voice3
CIF-Q	34.2	22.7	11.2	30.3	18.5	2.2
TD-FQ	31.6	20.3	10.9	22.6	11.0	0.7

TABLE 2: Packet dropping ratios (%) of real-time flows.

constant-bit-rate (CBR) and ON-OFF model. The latter is to model voice communication. The average durations of ON and OFF states are set to 2.5 and 0.5 seconds, respectively. During the ON period, packets are generated with fixed intervals. No packet is generated during the OFF period. The last two flows are non-real-time FTP flows, and their traffics are modeled as greedy sources whose queues are never empty. As for error scenarios, we use two parameters T_{good} and T_{bad} to control the average time when the channel stays in error-free and error states, respectively. The total channel capacity is set to 5 Mb/s. The total simulation time in this experiment is 100 seconds. The packet dropping ratios and queuing delays of real-time flows are shown in Table 2 and Table 3, respectively, where the *packet dropping ratio* is defined as the ratio of the number of packets dropped due to exceeding deadlines to the number of packet generated, where the deadline of a packet is set to twice of the packet interarrival time. From Table 2 and Table 3, we can observe that the packet dropping ratios and queuing delays of real-time flows in TD-FQ are smaller than those in CIF-Q, especially when the flows are voice traffic. This is because TD-FQ not only lets real-time flows give up less services to compensate other lagging flows, but also gives more services to real-time lagging flows for compensation. More performance comparisons can be found in [26].

5 Summary of Algorithms

In this section, we summarize and compare the algorithms discussed in this chapter. Table 4 gives a summary. For algorithms that adopt an error-free reference model, IWFQ uses WFQ [2] as its reference model, CIF-Q and AC-FQ use SFQ [4] as their reference model, and WFS uses a modified version of WFQ as its reference model. For compensation mechanisms, SBFA and HCS use virtual flows to compensate lagging flows, while Havana uses clouters to record the services that should be compensated to lagging flows. For the dynamic weight adjustment property, ELF adjusts flows' weights according to their channel conditions, while AC-FQ changes flows' weights based on their queue lengths. To separate different traffic types, WFS proposes the concept of rate weights and delay weights to decouple delay and bandwidth, while HCS and TD-FQ take special treatments for real-time flows. To reserve resources for compensation purpose , SBFA uses virtual

algorithm	error-free reference model	explicit compensation mechanism	dynamic weight adjustment	separating traffic types	reserving resources for compensation	short-term fairness
IWFQ	√					
CIF-Q	√					√
SBFA		√			√	
Havana		√				
ELF			√			
AC-FQ	√		√			√
WFS	√			√		√
HCS		√		√	√	
TD-FQ				√		√

TABLE 4: Comparison of the algorithm properties.

Flow	CBR1	CBR2	CBR3	voice1	voice2	voice3
CIF-Q	4.7	4.0	3.4	35.0	30.0	24.9
TD-FQ	4.2	3.6	3.2	29.7	25.8	23.0

TABLE 3: Average queuing delays (ms) of real-time flows.

flows called LTFS to compensate lagging flows, while HCS uses a compensation flow to redistribute its services to real flows and to give/receive weights to/from handoff flows. As for short-term fairness, CIF-Q, AC-FQ, and TD-FQ guarantee that the difference between the normalized services received by any two error-free flows that are continuously backlogged and are in the same state (leading, lagging, or satisfied) during any time interval will be bounded. The work in WFS has provided a bound for satisfied flows.

6 Conclusions

Since wireless networks have become more popular and are widely accessed nowadays, there will be more and more multimedia applications executed upon such networks. So how to support different QoS demands and fair sharing of wireless bandwidth becomes one of the most important issues for wireless networks. Although many fair queuing algorithms have been proposed to provide fairness and bounded delay properties for wireline networks, they cannot be directly applied to wireless networks. This motivates many researchers to develop new fair queuing algorithms for wireless networks. This chapter provides a comprehensive survey on current research in wireless fair qeueing. Various representative algorithms are discussed in the chapter.

References

[1] S. Lu, T. Nandagopal, and V. Bharghavan, "Design and analysis of an algorithm for fair service in error-prone wireless channels," *Wireless Networks*, vol. 6, no. 4, pp. 323–343, 2000.

[2] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, pp. 1374–1396, 1995.

[3] A.K. Parekh and R.G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Trans. Networking*, vol. 1, pp. 344–357, 1993.

[4] P. Goyal, H.M. Vin, and H. Cheng, "Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, pp. 690–704, 1997.

[5] S.J. Golestani, "A self-clocked fair queueing scheme for broadband applications," *Proc. IEEE INFOCOM*, pp. 12–16, 1994.

[6] J.C.R. Bennett and H. Zhang, "WF^2Q: worst-case fair weighted fair queueing," *Proc. IEEE INFOCOM*, vol. 1, pp. 120–128, 1996.

[7] V. Bharghavan, S. Lu, and T. Nandagopal, "Fair queuing in wireless networks: issues and approaches," *IEEE Personal Commun.*, vol. 6, pp. 44–53, 1999.

[8] Y. Cao and V.O.K. Li, "Scheduling algorithms in broadband wireless networks," *Proc. IEEE*, vol. 89, pp. 76–87, 2001.

[9] T. Nandagopal, S. Lu, and V. Bharghavan, "A unified architecture for the design and evaluation of wireless fair queueing algorithms," *Wireless Networks*, vol. 8, pp. 231–247, 2002.

[10] S. Lu, V. Bharghavan, and R. Srikant, "Fair scheduling in wireless packet networks," *IEEE/ACM Trans. Networking*, vol. 7, pp. 473–489, 1999.

[11] T.S.E. Ng, I. Stoica, and H. Zhang, "Packet fair queueing algorithms for wireless networks with location-dependent errors," *Proc. IEEE INFOCOM*, pp. 1103–1111, 1998.

[12] P. Ramanathan and P. Agrawal, "Adapting packet fair queuing algorithms to wireless networks," *Proc. ACM/IEEE Int'l Conf. Mobile Computing and Networking*, pp. 1–9, 1998.

[13] J. Gomez, A.T. Campbell, and H. Morikawa, "The Havana framework for supporting application and channel dependent QoS in wireless networks," *Proc. IEEE Int'l Conf. Network Protocols*, pp. 235–244, 1999.

[14] D. A. Eckhardt and P. Steenkiste, "Effort-limited fair (ELF) scheduling for wireless networks," *Proc. IEEE INFOCOM*, pp. 1097–1106, 2000.

[15] K.C. Wang and Y.L. Chin, "A fair scheduling algorithm with adaptive compensation in wireless networks," *Proc. IEEE Global Telecomm. Conf.*, pp. 3543–3547, 2001.

[16] S. Lu, T. Nandagopal, and V. Bharghavan, "A wireless fair service algorithm for packet cellular networks," *Proc. ACM Int'l Conf. Mobile Computing and Networking*, pp. 10–20, 1998.

[17] S. Lee, K. Kim, and A. Ahmad, "Channel error and handoff compensation scheme for fair queueing algorithms in wireless networks," *Proc. IEEE Int'l Conf. Comm.*, pp. 3128–3132, 2002.

[18] J. Nagle, "On packet switches with infinite storage," *IEEE Trans. Comm.*, vol. 35, pp. 435–438, 1987.

[19] H. S. Wang and N. Moayeri, "Finite-state Markov channel: a useful model for radio communication channels," *IEEE Trans. Vehicular Technology*, vol. 44, pp. 163–171, 1995.

[20] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," *J. Internetworking Research and Experience*, vol. 1, pp. 3–26, 1990.

[21] J. Turner, "New directions in communications, or which way to the information age?" *IEEE Comm. Magazine*, vol. 24, pp. 8–15, 1986.

[22] J.C.R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, pp. 675–689, 1997.

[23] I. Stoica, H. Zhang, and T.S.E. Ng, "A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services," *IEEE/ACM Trans. Networking*, vol. 8, no. 2, pp. 185–199, 2000.

[24] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 375–385, 1996.

[25] M.R. Jeong, H. Morikawa, and T. Aoyama, "Wireless packet scheduler for fair service allocation," *Proc. IEEE Asia-Pacific Conf. Comm. and Optoelectronics*, pp. 794–797, 1999.

[26] Y.C. Wang, S.R. Ye, and Y.C. Tseng, "A fair scheduling algorithm with traffic classification in wireless networks," *Computer Comm.*, vol. 28, no. 10, pp. 1225–1239, 2005.