Fair Round Robin: A Low Complexity Packet Scheduler with Proportional and Worst-Case Fairness [*]

Xin Yuan[†] and Zhenhai Duan
Department of Computer Science, Florida State University, Tallahassee, FL 32306
{xyuan, duan}@cs.fsu.edu

Abstract

Round robin based packet schedulers generally have a low complexity and provide long-term fairness. The main limitation of such schemes is that they do not support short-term fairness. In this paper, we propose a new low complexity round robin scheduler, called Fair Round Robin (FRR), that overcomes this limitation. FRR has similar complexity and long-term fairness properties as the stratified round robin scheduler, a recently proposed scheme that arguably provides the best quality-of-service properties among all existing round robin based low complexity packet schedulers. FRR offers better short-term fairness than stratified round robin and other existing round robin schedulers.

Keywords: Packet scheduling, proportional fairness, worst-case fairness, round robin scheduler

1 Introduction

An ideal packet scheduler should have a *low complexity*, preferably $O(1)$ with respect to the number of flows serviced, while providing *fairness* among the flows. While the definition of the complexity of a packet scheduling algorithm is well understood, the concept of fairness needs further elaboration. Many fairness criteria for packet schedulers have been proposed [10]. In this paper, we will use two well established fairness criteria to evaluate packet schedulers, the *proportional fairness* that was defined by Golestani in [6] and the *worst-case fairness* that was defined by Bennett and Zhang in [2].

Let $S_{i,s}(t_1, t_2)$ be the amount of data of flow f_i sent during time period $[t_1, t_2)$ using scheduler s. Let f_i and f_j be any two flows that are backlogged during an arbitrary time period $[t_1, t_2)$. The proportional fairness of scheduler s is measured by the difference between the normalized services received by the two flows, $|\frac{S_{i,s}(t_1,t_2)}{r_i} - \frac{S_{j,s}(t_1,t_2)}{r_j}|$. We will say that *a scheduler has a good proportional fairness property if the difference is bounded by a constant number of packets in each flow*, that is, $|\frac{S_{i,s}(t_1,t_2)}{r_i} - \frac{S_{j,s}(t_1,t_2)}{r_j}| \leq c_1\frac{L_M}{r_i} - c_2\frac{L_M}{r_j}$, where c_1 and c_2 are constants and L_M is the maximum packet size. One example scheduler with a good proportional fairness property is the Deficit Round Robin scheduler [17].

[*]A preliminary version of this paper is published in IEEE INFOCOM 2005.
[†]Corresponding author, Email xyuan@cs.fsu.edu, Phone: (850)644-9133, Fax: (850)644-0058

A scheduler with a good proportional fairness property guarantees long-term fairness: for any (long) period of time, the services given to any two continuously backlogged flows are roughly proportional to their weights. However, proportional fairness does not imply short-term fairness. Consider for example a scheduler with a good proportional fairness property serving packets from one 1Mbps flow and 1000 1Kbps flows. With a good proportional fairness property, each of the 1000 1Kbps flows can send a constant number of packets ahead of the packet that is supposed to be sent by the 1Mbps flow. Hence, during the period when the scheduler sends the few thousand packets from the 1Kbps flows, the 1Mbps flow is under-served: a scheduler with a good proportional fairness property can be short-term unfair to a flow. To better measure the short-term fairness property of a scheduler, worst case fairness is introduced in [2].

A scheduler, s, is worst-case fair to flow f_i if and only if the delay of a packet arriving at time t on flow f_i is bounded by $\frac{Q_{i,s}(t)}{r_i} + C_{i,s}$, where $Q_{i,s}(t)$ is the queue size of f_i at t, r_i is the guaranteed rate of f_i, and $C_{i,s}$ is a constant independent of the queues of other flows. A scheduler is worst-case fair if it is worst-case fair to all flows in the system. If a scheduler, s, is worst-case fair, the fairness of the scheduler is measured by the *normalized worst-case fair index* [2]. Let R be the total link bandwidth. The normalized worst-case fair index for the scheduler, c_s, is defined as $c_s = \max_i\{\frac{r_i C_{i,s}}{R}\}$. We will say that *a scheduler has a good worst-case fairness property if it has a constant (with respect to the number of flows) normalized worst-case fair index*. One example scheduler with a good worst-case fairness property is the WF^2Q scheduler [1, 2]. A scheduler with **both** a good proportional fairness property and a good worst-case fairness property provides both long-term and short-term fairness to all flows: for any (small) time period, worst-case fairness requires the guaranteed rates of all flows to be enforced within a small error margin.

Packet schedulers can be broadly classified into two types: timestamp based schemes [1, 2, 5, 6, 12] and round-robin algorithms [7, 8, 13, 17]. Timestamp based schemes have good fairness properties with a relatively high complexity, $O(log\ N)$, where N is the number of flows. Round-robin based algorithms have an $O(1)$ or quasi-$O(1)$ ($O(1)$ under practical assumptions [13]) complexity, but in general do not have good fairness properties. Round robin schemes including Deficit Round Robin (DRR) [17], Smoothed Round Robin (SRR) [7], and STratified Round Robin (STRR) [13] all have good proportional fairness properties. However, none of the existing round-robin schemes is known to have a normalized worst-case fair index that is less than $O(N)$. We will give an example in Section 3.1, showing that the normalized worst-case fair index of $STRR$ [13], a recently proposed round-robin based algorithm that arguably provides the best quality-of-service properties among all existing round-robin based schemes, is $\Omega(N)$. It can be shown that the normalized worst-case fair indexes of other round robin schemes, such as Smoothed Round Robin (SRR) [7] and Deficit Round Robin (DRR) [17], are also $\Omega(N)$. Not having a constant worst-case fair index means that the short-term service rate of a flow may significantly deviate from its fair rate, which can cause rate oscillation for a flow [2].

In this paper, we propose a new round robin based low complexity packet scheduling scheme, called Fair Round-Robin (FRR), that overcomes the limitation of not being able to guarantee short-term fairness. Like $STRR$, FRR employs a two-level scheduling structure and combines the ideas in timestamp based and round-robin schemes. FRR has a similar complexity as $STRR$: both have a low quasi-$O(1)$ complexity. However, unlike $STRR$ and other existing round robin based low complexity packet schedulers that only have a good proportional fairness property, FRR not only has a good proportional fairness property, but also maintains a quasi-$O(1)$ normalized worst-case fair index ($O(1)$ under practical assumptions). To the best of our knowledge, FRR is the only round robin based scheduler with a similar complexity that has such capability. The results of our simulation study show that FRR not only provides worst case guarantees, but also often has better short-term fairness in average cases in comparison to other round robin based schedulers.

The rest of the paper is structured as follows. Section 2 presents related work. Section 3 gives an example motivating the proposed packet scheduling scheme and introduces the background of this work. Section 4 describes FRR. Section 5 discusses the QoS properties of FRR. Section 6 reports the results of the simulation study of FRR. Finally, Section 7 concludes the paper.

2 Related work

We will briefly discuss timestamp based and round-robin packet scheduling schemes since both relate to FRR. Some timestamp based schedulers, such as Weighted Fair Queuing (WFQ) [12] and Worst-case Fair Weighted Fair Queuing (WF^2Q) [1, 2], closely approximate the Generalized Processor Sharing (GPS) [5, 12]. These schedulers compute a timestamp for each packet by emulating the progress of a reference GPS server and transmit packets in the increasing order of their timestamps. Other timestamp based approaches, such as Self-Clocked Fair Queuing (SCFQ) [6] and Virtual Clock [22], compute timestamps without referring to a reference GPS server. These methods still need to sort packets according to their timestamps and still have an $O(log\ N)$ per packet processing complexity. The Leap Forward Virtual Clock [18] reduces the sorting complexity by coarsening timestamp values and has an $O(loglog\ N)$ complexity. This scheme requires complex data structures and is not suitable for hardware implementation.

Deficit Round Robin (DRR) [17] is one of the round-robin algorithms that enjoy a good proportional fairness property. A number of methods have recently been proposed to improve delay and burstiness properties of DRR [7, 8, 13]. The Smoothed Round Robin (SRR) scheme [7] improves the delay and burstiness properties by spreading the data of a flow to be transmitted in a round over the entire round using a weight spread sequence. Aliquem [8, 9] allows the quantum of a flow to be scaled down, which results in better delay and burstiness properties. The Stratified Round Robin ($STRR$) [13] scheme bundles flows with similar rate requirements, scheduling the bundles through a sorted-priority mechanism, and using a round robin strategy to select flows in each bundle. $STRR$ guarantees that all flows get their fair share of *slots*. It enjoys a single packet delay bound that is independent of the number of flows in the system. However, as will be shown in the next section, the normalized worst-case fairness index for $STRR$ is $\Omega(N)$. This limitation of $STRR$ motivated the development of FRR. FRR is similar to $STRR$ in many aspects: FRR uses exactly the same way to bundle the flows with similar rate requirements and has the same two-level scheduling structure. FRR differs from $STRR$ in that it uses a different sorted-priority strategy to arbitrate among bundles, and a different round robin scheme to schedule flows within each bundle. The end result is that FRR has a similar complexity and a similar proportional fairness property, but a much better worst-case fairness property. Bin Sort Fair Queuing (BSFQ) [4] uses an approximate bin sort mechanism to schedule packets. The worst-case single packet delay of BSFQ is proportional to the number of flows. Hybrid scheduling schemes [14, 15] have also been proposed, where the scheduling tasks are separated into two levels. While the algorithm components of these schemes are similar to those of FRR and $STRR$, the QoS properties of these schemes are not clear. A recently proposed group round robin scheme [3] seeks to improving the complexity for time-stamp based schedulers using a two-level scheduling scheme similar to that in [15]. While group round robin is conceptually similar to FRR, it does not have the mechanism to maintain the fairness in cases when not all flows are backlogged continuously.

3 Background

Some notations used in this paper are summarized in Table 1. There are N flows f_1, f_2, ..., f_N sharing a link of bandwidth R. Each flow f_i has a minimum guaranteed rate of r_i. We will assume that $\sum_{i=1}^{N} r_i \leq R$. The

N	the number of flows in the system
n	the number of classes in the system
R	the total link bandwidth
r_i	the guaranteed bandwidth for flow f_i
$w_i = \frac{r_i}{R}$	the weight associated with flow f_i
W_k	the weight of a class F_k
L_M	the maximum packet size
$S_{i,s}(t_1, t_2)$	the amount of service received by session i during $[t_1, t_2)$ under the s server
$S_{i,s}(t)$	the amount of service received by session i during $[0, t)$ under the s server
$F_{i,s}^j$	the departure time of the jth packet of flow f_i under the s server
F_s^p	the departure time of packet p under the s server
$Q_{i,s}(t)$	the queue size of flow f_i at time t under the s server
p_i^j	the jth packet on flow f_i

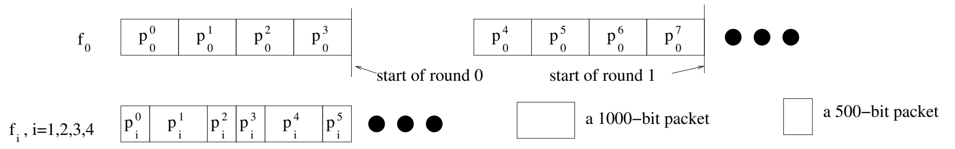
Table 1: Notations used in this paper

weight w_i of flow f_i is defined as its guaranteed rate normalized with respect to the total rate of the link, i.e., $w_i = \frac{r_i}{R}$. Thus, we have $\sum_{i=1}^{N} w_i \leq 1$.

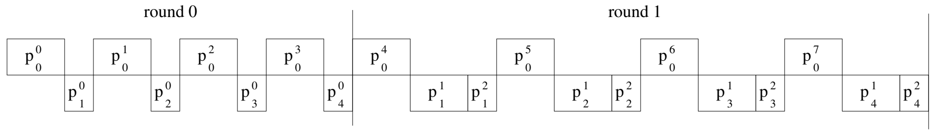
3.1 A motivating example

The development of FRR is motivated by $STRR$ [13], a recently proposed round-robin algorithm. In terms of the QoS properties of scheduling results, $STRR$ is arguably the best scheduler among all existing round-robin based low complexity schedulers. We will show that the normalized worst-case fair index of $STRR$ is $\Omega(N)$.

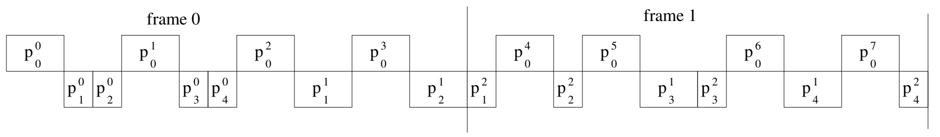
Let $N+1$ flows, f_0, f_1, ..., f_N, share an output link of bandwidth $2N$. The bandwidth of f_0 is N and the bandwidth of each flow f_i, $1 \leq i \leq N$, is 1. We will use R to denote the bandwidth of the output link, and r_i to denote the bandwidth of flow f_i, $0 \leq i \leq N$. $R = 2N$, $r_0 = N$, and $r_i = 1$, $1 \leq i \leq N$. Let the maximum packet size be $L_M = 1000$ bits. Packets in f_0 are of size $L_M = 1000$ bits. Flows f_1, f_2, ..., f_N are continuously backlogged with packets whose sizes repeat the pattern: $\frac{L_M}{2} = 500$ bits, $L_M = 1000$ bits, $\frac{L_M}{2} = 500$ bits, 500 bits, 1000 bits, 500 bits, and so on. Figure 1 (a) shows the packet arrival pattern assuming $N = 4$. In $STRR$, these flows are grouped into two classes: one class containing only f_0 and the other having flows f_1, ..., f_N. The bandwidth is allocated in the unit of slots. Let us assume that each of the flows has the minimum weight in its class and the credit assigned to each of the flows in a *slot* is $L_M = 1000$ bits. $STRR$ guarantees that slots are allocated fairly among all flows: f_0 is allocated one slot every two slots and each of the flows f_1, f_2, ..., f_N gets one slot every $2N$ slots. We will use the DRR concept of *round* to describe the scheduling results of $STRR$. In each round, all backlogged flows have a chance to send packets.



(a) Packet arrival pattern



(b) Scheduling results using STRR



(c) Scheduling results using FRR

Figure 1: A motivating example

In the example, each round contains N slots from f_0 and one slot from each of the flows f_i, $1 \leq i \leq N$. Due to the differences in packet sizes, the sizes of slots are different. For f_0, each slot contains exactly one packet of size L_M and the size of each slot is L_M. For f_i, $1 \leq i \leq N$, the size of the first slot is $\frac{L_M}{2} = 500$ bits since the second packet (size L_M) cannot be included in this slot. This results in 500-bit credits being passed to the next slot. Hence, the second slot for f_i contains 2 packets (1500-bit data). This pattern is then repeated: the size for an evenly numbered slot for f_i, $1 \leq i \leq N$, is 500 bits and the size of an oddly numbered slot is 1500 bits. The $STRR$ scheduling result is shown in Figure 1 (b), where the rate allocated to flow f_0 oscillates between $\frac{4}{3}N$ and $\frac{4}{5}N$ for the alternating rounds. Note that since the size for each round depends on N, the duration of a round depends on N and can be fairly large.

Now consider the normalized worst-case fair index of $STRR$, c_{STRR}. Let us assume that in the example the last f_0 packet in round 1, which is the $2N-1$-th packet of f_0, p_0^{2N-1}, arrives at time a_0^{2N-1} right before the starting of round 1 (after the last f_0 packet in round 0 departs). At a_0^{2N-1}, the queue length in f_0 is $Q(a_0^{2N-1}) = N \times L_M$. Following the scheduling results shown in Figure 1 (b), $(N-1) \times L_M$ data in f_0 and $(N-1) \times (L_M + \frac{L_M}{2})$ data in flows f_i, $1 \leq i \leq N$, in round 1 are scheduled before p_0^{2N-1}. Let d_0^{2N-1} be the departure time of p_0^{2N-1}. We have $d_0^{2N-1} - a_0^{2N-1} = \frac{(N-1)(L_M+1\frac{1}{2}L_M)+L_M}{R}$. $C_{0,STRR} \geq d_0^{2N-1} - a_0^{2N-1} - \frac{Q(a_0^{2N-1})}{r_0} = \frac{(N-1)(L_M+1\frac{1}{2}L_M)+L_M}{R} - \frac{N\times L_M}{r_0} = \frac{0.25\times N\times L_M}{r_0} - 0.75\frac{L_M}{r0}$. Hence, $c_{STRR} \geq c_{0,STRR}\frac{r_0}{R} = 0.25 \times N \times \frac{L_M}{R} - 0.75\frac{L_M}{R} = \Omega(N)$.

While we only show the normalized worst-case fair index of STRR in this section, one can easily show that the normalized worst-case fair indexes of other round robin schedulers such as smoothed round robin [7] and deficit round robin [17] are $\Omega(N)$: not having a good bound on worst-case fairness is a common problem with all of these low complexity round robin packet schedulers. FRR overcomes this limitation and grants a much better a short-term fairness property while maintaining a low complexity. The scheduling results of FRR for the example in Figure 1 are shown in Figure 1 (c). As can be seen from the figure, the short-term behavior of f_0 is much better than that in Figure 1 (b): counting from the beginning of round 0, for every 2000 bits data sent, exactly 1000 bits are from f_0.

3.2 Deficit Round Robin (DRR)

Since FRR is built over Deficit Round Robin (DRR) [17], we will briefly describe DRR and present some properties of DRR that are needed to understand the properties of FRR.

Like the ordinary round robin scheme, DRR works in rounds. Within each round, each backlogged flow has an opportunity to send packets. Each flow f_i is associated with a quantity Q_i and a variable DC_i (deficit counter). The quantity Q_i is assigned based on the guaranteed rate for f_i and specifies the target amount of data that f_i should send in each round. Since the scheduler operates in a packet-by-packet fashion, f_i may not be able to send exactly Q_i data in a round. The variable DC_i is introduced to record the quantum that is not used in a round so that the unused quantum can be passed to the next round. To ensure that each backlogged flow can send at least one packet in a round, $Q_i \geq L_M$. Some properties of DRR are summarized in the following lemmas.

Lemma 1: Assuming that flow f_i is continuously backlogged during $[t_1, t_2)$. Let X be the smallest number of continuous DRR rounds that completely enclose $[t_1, t_2)$. The service received by f_i during this period, $S_{i,DRR}(t_1, t_2)$, is bounded by $(X-3)Q_i \leq S_{i,DRR}(t_1, t_2) \leq (X+1)Q_i$.

Proof: See appendix. $\Box$

Lemma 2: Let f_1, ..., f_N be the N flows in the system with guaranteed rates r_1, ...,r_N. $\sum_{i=1}^{N} r_i \leq R$. Let $r_{min} = \min_i\{r_i\}$ and $r_{max} = \max_i\{r_i\}$. Let $r_{max} = D * r_{min}$. Assume that D is a constant with respect to N and that DRR is used to schedule the flows with $Q_i = L_M * \frac{r_i}{r_{min}}$. The following statements are true. All constants in this lemma are with respect to N.

1. Let packet p arrive at the head of the queue for f_i at time t. There exists a constant c_1 ($c_1 = O(D^2)$) such that packet p will be serviced before $t + c_1 \times \frac{L_M}{r_i}$.
2. The normalized worst-case fair index of DRR in such a system is a constant c_1 ($c_1 = O(D^2)$).
3. Let f_i and f_j be continuously backlogged during any given time period $[t_1, t_2)$, there exists two constants c_1 and c_2 ($c_1 = O(D)$ and $c_2 = O(D)$) such that the normalized service received by the two flows during this period is bounded by $|\frac{S_{i,DRR}(t_1,t_2)}{r_i} - \frac{S_{j,DRR}(t_1,t_2)}{r_j}| \leq c_1\frac{L_M}{r_i} + c_2\frac{L_M}{r_j}$.

Proof: See appendix. □

We will call $D = \frac{r_{max}}{r_{min}}$, the *maximum weight difference factor*. Lemma 2 shows that when D is a constant with respect to N, DRR has the following three properties. First, the worst-case single packet delay depends on the guaranteed rate for the flow and is independent of the number of flows in the system. Second, DRR has a constant normalized worst-case fair index. Third, DRR has a good proportional fairness property. Thus, DRR is an excellent scheduler under the assumption that D is a small constant. The problem with DRR is that when the weights of the flows differ significantly (D is a large number), which is common in practice, the QoS performance bounds, which are functions of D, become very large.

FRR extends DRR such that the QoS properties in Lemma 2 hold for any weight distribution, while maintaining an low quasi-$O(1)$ complexity. The basic idea is as follows. FRR chooses a constant C (e.g. $C = 2$) that is independent of D and N. FRR groups flows whose weights differ by at most a factor of C into classes and uses a variation of DRR to schedule packets within each class. From Lemma 2, DRR can achieve good QoS properties for flows in each class. Thus, the challenge is to isolate the classes so that flows in different classes, which are flows with significantly different weights, do not affect each other too much. FRR uses a timestamp based scheduler to isolate the classes. As a result, FRR schedules packets in two levels, a timestamp based inter-class scheduling and a DRR based intra-class scheduling.

4 FRR: a fair round robin scheduler

Like stratified round robin ($STRR$) [13], FRR groups flows into a number of classes with each class containing flows with similar weights. For $k \geq 1$, class F_k is defined as

$$F_k = \{f_i : \frac{1}{C^k} \leq w_i < \frac{1}{C^{k-1}}\},$$

where C is a constant independent of D and N. The specific value of C can be selected by the system designer. Let r be the smallest unit of bandwidth that can be allocated to a flow. The number of classes is $n = \lceil log_C(\frac{R}{r}) \rceil$. In practice, n is usually a small constant. For example, consider an extreme case with $R = 1Tbps$, $r = 1Kbps$ ($D = 10^9$). When $C = 8$, $n = \lceil log_8(10^9) \rceil = 10$. Like [13], we will consider the practical assumption that n is an $O(1)$ constant. However, since $n = \lceil log_C(\frac{R}{r}) \rceil$ in theory, we will derive the bounds on QoS properties and complexity in terms of n.

It must be noted that the constant C in FRR is very different from the constant weight difference factor D in Lemma 2. D specifies a limit on the type of flows that can be supported in the system. C is an algorithm parameter that can be selected by the scheduler designer and does not put a limit on the weights of the flows in the system. Consider the case when $R = 1Tbps$ and $r = 1kbps$. $D = \frac{10^{12}}{10^3} = 10^9$. Using DRR to schedule packets may result in extremely poor QoS bounds since $O(D^2)$ can be huge numbers. With FRR, one can select a small number C (e.g. $C = 2$) and obtain QoS bounds that are linear functions of C and n.

FRR has two scheduling components, intra-class scheduling that determines the order of the packets within each class and the weight of the class, and inter-class scheduling that determines the class, and thus, the packet within the class, to be transmitted over the link. The concept of *weight* will be used in different contexts. A weight is associated with each flow. In intra-class scheduling, the packet stream within a class is partitioned into frames. A weight that represents the aggregate weight for all active flows in a frame is assigned to the frame. The weight of a frame is then used in inter-class scheduling to decide which class is to be served. In the inter-class scheduling, we will also call the weight of current frame in a class the weight of the class. We will use notion w_i to denote the weight of a flow f_i and W_k to denote the weight of a class F_k.

4.1 Intra-class scheduling

Assuming that the inter-class scheduling scheme can provide fairness among classes based on their weights, the intra-class scheduler must be able to transfer the fairness at the class level to that at the flow level. To focus on the intra-class scheduling issues, we will assume that GPS is the inter-class scheduling scheme in this sub-section.

The intra-class scheduling scheme in FRR, called *Lookahead Deficit Round Robin with Weight Adjustment* ($LDRRWA$), is a variation of DRR with two extensions: a *lookahead* operation and a *weight adjustment* operation. To understand the needs for the two extensions, let us examine the issues when a vanilla DRR

scheme is used in intra-class scheduling. In DRR, the packet stream within a class is partitioned into *rounds*. Each of active flows is allocated a *quantum* for sending data in a round. To offset the weight differences among the flows, each flow $f_i \in F_k = \{f_i : \frac{1}{C^k} \leq w_i < \frac{1}{C^{k-1}}\}$ is assigned a quantum of $Q_i = C^k w_i L_M$. Since $\frac{1}{C^k} \leq w_i < \frac{1}{C^{k-1}}$, $L_M \leq Q_i < C \times L_M$.

DRR decides the order of the packets within a class, but not the weights for the class. Since inter-class scheduling in FRR schedules the classes based on their weights, it is crucial to assign weights to each class such that both flows within the class and flows in other classes are treated fairly. In DRR, different flows can be active in different rounds. Since the weight assigned to a class must reflect the weights of all active flows, it is natural to assign a different weight to a different round. One simple approach, which is adopted in the group round robin scheme [3], is to assign the sum of weights of all active flows in a round as the weight of the round. This simple approach, however, does not yield a fair scheduler. Consider the case when two flows, f_1 and f_2, of the same weight $w_1 = w_2 = \frac{1}{C^k}$ are in a class F_k. $Q_1 = Q_2 = L_M$. Flow f_1 is continuously backlogged and sends L_M data in each round. Flow f_2 is active and sends $\frac{Q_2}{2} = \frac{L_M}{2}$ data in each round. The simple approach will assign weight $w_1 + w_2 = 2w_1$ to each round, which results in the guaranteed service rate under GPS for this class to be $2r_1$. Since $\frac{2}{3}$ of the service is used to serve packets in f_1, the guaranteed rate for f_1 is artificially inflated to $2r_1 \times \frac{2}{3} = \frac{4}{3}r_1$, which is unfair to flows in other classes.

What is the fair weight for a round in class F_k? In each round, each active flow $f_i \in F_k$ with a rate r_i is given a quantum of Q_i. The targeted finishing time for f_i is thus $\frac{Q_i}{r_i} = \frac{C^k L_M}{R}$. From Lemma 1, we can see that for a flow f_i that is continuously backlogged in X rounds, the amount of data sent is at most a few packets from $X \times Q_i$. Hence, if the weights for all rounds are assigned such that the service time for each round is $\frac{Q_i}{r_i} = \frac{C^k L_M}{R}$ using the guaranteed service rate, all continuously backlogged flows in the X rounds obtain their fair share of the bandwidth with a small error margin: the fairness at the class level is transferred to the fairness at the flow level. Hence, **the fair weight for a round in class F_k should be one that results in the targeted finishing time of $\frac{Q_i}{r_i} = \frac{C^k L_M}{R}$**.

Let flows f_1, ..., f_m (in a class) be active in a round. Let the data sizes of f_i, $1 \leq i \leq m$, in the round be s_i. The size of the round is $round\ size = s_1 + s_2 + ... + s_m$. Let w' be the fair weight for the round and r' be the corresponding guaranteed service rate, $w' = \frac{r'}{R}$. We have $\frac{round\ size}{r'} = \frac{round\ size}{w'R} = \frac{C^k L_M}{R}$. Solving the equation, we obtain

$$w' = \frac{round\ size}{C^k L_M} = \frac{s_1+s_2+...+s_m}{C^k L_M}.$$

$\frac{s_1+s_2+...+s_m}{C^k L_M} = \frac{s_1}{C^k L_M} + \frac{s_2}{C^k L_M} + ... + \frac{s_m}{C^k L_M} = \frac{s_1}{Q_1} w_1 + \frac{s_2}{Q_2} w_2 + ... + \frac{s_m}{Q_m} w_m$: the fair weight for a round can also be interpreted as the sum of the normalized weights of active flows, $\frac{s_i}{Q_i} w_i$. The fair weight for flow f_i depends not only on its original weight w_i, but also its quantum Q_i and the size of data to be sent in the round s_i. In other words, in order to obtain the fair weight for a round, the weight of each active flow f_i must be adjusted from w_i to $\frac{s_i}{Q_i} w_i$ before the adjusted weights are aggregated.

Although the weight adjustment results in the fair weight for a round $w' = \frac{round\ size}{C^k L_M}$, w' may sometimes be more than the sum of the weights of all flows in the class. For example, if a class has only one flow f_1 and $Q_1 = L_M$, f_1 may send $0.5L_M$ in one round and $1.5L_M$ in another round. Using weight adjustment, the weight for the class is $0.5w_1$ in one round and $1.5w_1$ in the other round. This temporary raising of weights to $1.5w_1$ may violate the assumption that the sum of the weights for all classes is less than 1, which is essential for guaranteeing services. $LDRRWA$ uses the lookahead operation to deal with this problem. The lookahead operation moves some currently backlogged packets that are supposed to send in the next round under DRR into the current around. By using the lookahead operation, the size of each round (now called **frame** to be differentiated from the DRR round) is no more than the sum of the weights of all active flows in the round. This guarantees that the fair weight assigned to each frame to be less than the sum of the weights of all active flows in the frame.

4.1.1 Lookahead Deficit Round Robin with Weight Adjustment ($LDRRWA$)

$LDRRWA$ is a variation of DRR. The packet stream is partitioned into frames. It uses weight adjustment to compute the fair weight for each round and incorporates a lookahead operation to ensure that the weight of a round is less than or equal to the sum of the weights of the flows in the class. In $LDRRWA$, for each frame, a flow $f_i \in F_k = \{f_i : \frac{1}{C^k} \leq w_i < \frac{1}{C^{k-1}}\}$ is assigned a quantum of

$$Q_i = 2C^k w_i L_M.$$

Since $\frac{1}{C^k} \leq w_i < \frac{1}{C^{k-1}}$, $2L_M \leq Q_i < 2C \times L_M$. Q_i in $LDRRWA$ is two times the value in DRR. The reason is that the deficit counter may be negative in $LDRRWA$, $Q_i = 2C^k w_i L_M$ ensures that a backlogged flow can at least send one packet in a frame. Since $Q_i = 2C^k w_i L_M$, $\frac{Q_i}{r_i} = \frac{2C^k L_M}{R}$ and the fair weight for a frame of size $framsize$ is

$$W_k = \frac{framesize}{2C^k L_M}.$$

Let f_1, f_2, ..., f_m be the flows in class F_k. $W_k = \frac{framesize}{2C^k L_M} = \frac{framesize}{\sum_{i=1}^m Q_i} \sum_{i=1}^m w_i$. $LDRRWA$ employs

variable	explanation
$deficitcount_i$	the deficit count for flow f_i
$remaindeficit$	the sum of quantum not used in the DRR round
$lastingflowlist$	the flows that last to the next frame
$framesize$	the size of the frame
$frameweight$	the weight for the frame
$remainsize$	size of the part of a packet that belongs to current frame

Table 2: Major variables used in the frame calculation algorithm

the lookahead operation to ensure that $framesize \leq \sum_{i=1}^{m} Q_i$, which results in $W_k \leq \sum_{i=1}^{m} w_i$. A frame in $LDRRWA$ has two parts: the first part includes all packets that are supposed to be sent using DRR; the second part includes packets from the lookahead operation. In the lookahead operation, packets from flows that do not use up their quanta and are still backlogged after the current DRR round are sent in the current frame. In this case, the size of the first backlogged packet is larger than the remaining quota of the flow. Each of such flows may contribute at most one packet in the lookahead operation. Note that after a flow contributes its packet in the lookahead operation, the deficit counter for this flow has a negative value. The lookahead operation ensures that the aggregate deficit (the sum of the deficits) of all the backlogged flows in every frame is exactly 0 at frame boundaries. In other words, no credit is passed over frame boundaries at the frame level. As a result, the size of each frame is less than or equal to the total credits generated in that frame, which is at most $\sum_{i=1}^{m} Q_i$. Note that the frame boundary may not align with packet boundary: a packet may belong to two frames. Note also that while the aggregate deficit of all backlogged flows is 0 at frame boundaries, each individual flow may have a positive, zero, or negative deficit counter. Allowing a flow to have a negative deficit may potentially cause problems: a flow may steal credits by over sending in a frame (and having a negative deficit at the frame boundary), becoming inactive for a short period of time (so that the negative deficit can be reset), and over sending again. To deal with this situation, $LDRRWA$ keeps the negative deficit for one frame when the flow becomes inactive before it resets the negative deficit counter for the flow.

Each frame is decided at the time it starts. Packets arrive during the current frame are sent in later frames. Since each flow only sends a small number of packets in each frame, delaying a packet for one frame does not affect the fairness of the scheduler. Next, we will describe the high level logical view of $LDRRWA$.

Algorithm for computing the next frame for class F_k

```
(1) $remaindeficit = framesize = 0$
(2) $lastingflowlist = NULL$
(3) **if** $(remainsize > 0)$ **then**
        /*The partial packet belongs to this frame */
(4)     $framesize = framesize + remainsize$
(5) **end if**
    /* forming the $DRR$ round */
(6) **for** each active flow $f_i$ **do**
(7)     $deficitcount_i = deficitcount_i + quantum_i$
(8)     **while** $(deficitcount_i > 0)$ **and** $(f_i$ not empty$)$ **do**
(9)         $pktsize = size(head(f_i))$
(10)        **if** $(pktsize < deficitcount_i)$ **then**
(11)            remove head from $f_i$ and put it in the frame
(12)            $framesize = framesize + pktsize$
(13)            $deficitcount_i = deficitcount_i - pktsize$
(14)        **else** break
(15)        **end if**
(16)    **end while**
(17)    **if** $(f_i$ is empty $)$ **then**
(18)        $deficitcount_i = 0$
(19)    **else**
(20)        $remaindeficit = remaindeficit + deficitcount_i$
(21)        insert $f_i$ to $lastingflowlist$
(22)    **end if**
(23) **end for**
    /* lookahead operation */
(24) $f_i$ = head(lastingflowlist)
(25) **while** $(f_i \neq NULL)$ **and** $(remaindeficit > 0)$ **do**
(26)    $pktsize = size(head(f_i))$
(27)    **if** $(pktsize < remaindeficit)$ **then**
(28)        remove head from $f_i$ and put it in the frame
(29)        $framesize = framesize + pktsize$
(30)        $remaindeficit = remaindeficit - pktsize$
(31)        $deficitcount_i = deficitcount_i - pktsize$
(32)    **else** break
(33)    **end if**
(34)    $f_i = nextflow(f_i)$
(35) **end while**
(36) **if** $(f_i \neq NULL)$ **then**
(37)    $pktsize = size(head(f_i))$
(38)    remove head from $f_i$ and put it in the frame
(39)    $framesize = framesize + remaindeficit$
(40)    $remainsize = pktsize - remaindeficit$
(41)    $deficitcount_i = deficitcount_i - pktsize$
(42) **end if**
    /* computing the weight */
(43) $frameweight = \frac{framesize}{2C^k L_M}$
(44) **if** $(frameweight < \frac{1}{C^k})$ $frameweight = \frac{1}{C^k}$
```

Figure 2: The algorithm for computing the next frame for class F_k

A detailed implementation of $LDRRWA$, which consists of packet-by-packet operations, is described in Appendix B.

Figure 2 shows the logical algorithm for computing each $LDRRWA$ frame and its weight. In practice, these frame and weight calculation operations are distributed to the operations in packet arrivals and packet departures (Appendix B). Table 2 summarizes the major variables in the algorithm. Like DRR, variable $deficitcount_i$ is associated with flow f_i to maintain the credits to be passed over to the next DRR round and decide the amount of data to be sent in one round. After each DRR round, $remaindeficit$ maintains the sum of the quanta not used in the current DRR round, that is, the quanta that cannot be used since the size of the next backlogged packet is larger than the remaining quanta for a flow. In traditional DRR, these unused quanta will be passed to the next DRR round. In $LDRRWA$, in addition to passing the unused quanta to the next DRR round, some packets that would be sent in the next DRR round are placed in the current $LDRRWA$ frame so that at frame boundaries $remaindeficit$ is always equal to 0. This is the lookahead operation. The $lastingflowlist$ contains the list of flows that are backlogged at the end of the current DRR round. Flows in $lastingflowlist$ are candidates to supply packets for the lookahead operation. $Frameweight$ is the weight to be used by inter-class scheduling for the current frame. Variable $framesize$ records the size of the current frame. Since FRR needs to enforce that $remaindeficit = 0$ at frame boundaries, frame boundaries may not align with packet boundaries and a packet may belong to two frames. Variable $remainsize$ is the size of the part of the last packet in the frame that belongs to the next frame, and thus, should be counted in the $framesize$ for the next frame.

Let us now examine the algorithm in Figure 2. In the initialization phase, line (1) to line (5), variables are initialized and $remainsize$ is added to $framesize$, which effectively includes the partial packet in the frame to be computed. After the initialization, there are three main components in the algorithm: forming a DRR round, lookahead operation, and weight calculation. In the first component, line (6) to line (23), the algorithm puts all packets in the current DRR round that have not been served into the current frame. In the second component, line (24) to line (42), the algorithm performs the lookahead operation by moving some packets in the next DRR round into the current frame so that $remaindeficit = 0$ at the frame boundary. This is done by allowing some flows to borrow credits from the next DRR round. Since $remaindeficit = 0$, no credit is passed from one frame to the next frame for the class that aggregates many flows. Notice that each backlogged flow can contribute at most one packet in the lookahead operation. Notice also that a class as a whole does not pass credits between frames. However, for an individual flow, credits may still pass from one frame to the next. As a result, the $deficitcount_i$ variable may have a negative or positive value at frame boundaries. Finally, lines (43) and (44) compute the weight for the frame.

The complexity of the algorithm in Figure 2 is $O(M)$, where M is the number of packets in a frame. Clearly, this high level algorithm cannot be directly used in a scheduler to determine the next frame and frame weight. Otherwise, it will introduce an $O(M)$ processing complexity, which is larger than $O(N)$, in a packet scheduling event since the algorithm goes through each packet in the frame. This algorithm is only used to illustrate the logical operations of $LDRRWA$. The operations described in this algorithm, however, can be realized in the packet-by-packet operations when packets arrive and depart. By distributing the $O(M)$ operations for determining a frame into M packets in a frame, an additional of $O(1)$ operations are needed in each packet arrival and departure. In other words, $LDDRWA$ only introduces $O(1)$ per packet processing overheads.

A detailed description of the packet-by-packet operations of $LDRRWA$ is given in Appendix B, where we show how to realize $LDDRWA$ with $O(1)$ per packet processing overheads. The detailed packet-by-packet operations are rather tedious. The idea, however, is straight-forward. $LDRRWA$ maintains active flows in different queues. To determine a frame and its weight, our scheme determines (1) the total size of the frame, and (2) for each active flow in the frame the size of the data in that flow that belong to the frame. Such information is obtained by maintaining the following information at each packet arrival and departure: the size of the remaining current frame, the size of the next frame, the size of the partial packet in the current frame, the starting time of the current frame, the deficit counter for each flow, the size of the data for each flow in the current frame, the size of the data for each flow in the next frame, the size of all backlogged data in a flow, and the last time that a flow is serviced. Clearly, bookkeeping for all these variables takes $O(1)$ operations. With such information, the computation of the next frame, as well as the whole $LDRRWA$ can be realized with $O(1)$ per packet processing overhead.

4.1.2 Properties of $LDRRWA$

Lemma 3: Assuming that flow f_i is continuously backlogged during $[t_1, t_2)$. Let X be the smallest number of continuous $LDRRWA$ frames that completely enclose $[t_1, t_2)$. The service received by f_i during this period, denoted as $S_{i,LDRRWA}(t_1, t_2)$, is bounded by

$$(X-3)Q_i \leq S_{i,LDRRWA}(t_1,t_2) \leq (X+1)Q_i.$$

Proof: See appendix. □

Comparing Lemma 3 and Lemma 1, we can see the similarity between DRR and $LDRRWA$: in both schemes, the amount of data sent from a flow f_i that is continuously backlogged for X frames (rounds) is a few packets from $X \times Q_i$. Note that Q_i in $LDRRWA$ is twice that in DRR.

Lemma 4: In $LDRRWA$, the weight for a frame is less than or equal to the sum of the weights of all flows in the class.

Proof: Obvious from the previous discussion. □

At any given time, let W_k, $1 \leq k \leq n$ be the weights for the n classes (W_k may change over time). Lemma 4 establishes that $\sum_{i=1}^{n} W_k \leq \sum_{i=1}^{N} w_i \leq 1$. Thus, under GPS, the bandwidth allocated to class k is given by $\frac{W_k}{\sum_{i=1}^{n} W_k} R \geq R \times W_k$. We will call $R \times W_k$ the *GPS guaranteed rate*.

Lemma 5: Under GPS, the time to serve each $LDRRWA$ frame in class F_k is at most $\frac{2C^k L_M}{R}$.

Proof: Normally, the frame weight is computed as $W_k = \frac{framesize}{2C^k L_M}$ (line (43) in Figure 2). In cases when $\frac{framesize}{2C^k L_M}$ is less than the smallest weight for a flow in a class, the weight is increased (line (44) in Figure 2) to the smallest weight.

When $W_k = \frac{framesize}{2C^k L_M}$, the GPS guaranteed rate is $R\frac{framesize}{2C^k L_M}$ and the total time to serve the frame is at most $\frac{framesize}{R\frac{framesize}{2C^k L_M}} = \frac{2C^k L_M}{R}$. If W_k is increased, the conclusion still holds. □

Lemma 6: Under GPS, the time to service X bytes of data in class F_k is at most $\frac{XC^k}{R}$.

Proof: The minimum weight assigned to a backlogged class F_k is $\frac{1}{C^k}$. Thus, the GPS guaranteed rate for class F_k is at least $\frac{R}{C^k}$. Thus, the time to serve a queue of size X bytes in class F_k is at most $\frac{X}{\frac{R}{C^k}} = \frac{XC^k}{R}$. □

Lemma 7: For a class F_k frame of size no smaller than $2L_M$, the service time for the frame is exactly $2C^k \frac{L_M}{R}$ using the GPS guaranteed rate.

Proof: When $framesize \geq 2L_M$, $W_k = \frac{framesize}{2C^k L_M} \geq \frac{1}{C^k}$. Thus, the GPS guaranteed rate for the frame is $R\frac{framesize}{2C^k L_M}$ and the service time for the frame with the guaranteed rate is $\frac{framesize}{R\frac{framesize}{2C^k L_M}} = \frac{2C^k L_M}{R}$. □

Lemma 8: Let a class F_k frame contain packets of a continuously backlogged flow f_i, the size of frame is no smaller than $2L_M$.

Proof: Straight-forward from the fact that no credit is passed from the previous frame and to the next frame and that $Q_i \geq 2L_M$. □

Lemma 9: Let $f_i \in F_k$ and $f_j \in F_m$ be continuously backlogged during $[t_1, t_2)$. $k \geq m$. Let X_k and X_m be the smallest numbers of F_k and F_m frames that completely enclose $[t_1, t_2)$. Assume that classes F_k and F_m are served with the GPS guaranteed rate,

$$(X_k - 1)C^{k-m} \leq X_m \leq X_k C^{k-m} + 1.$$

Proof: Since $f_i \in F_k$ and $f_j \in F_m$ are continuously backlogged during $[t_1, t_2)$, the sizes of all frames during this period are no smaller than $2L_M$ (Lemma 8). From Lemma 7, using the GPS guaranteed rate, the time to service a class F_k frame is exactly $\frac{2C^k L_M}{R}$ and the time for a class F_m frame is exactly $\frac{2C^m L_M}{R}$. Since X_k and X_m are the smallest numbers of F_k and F_m frames that completely enclose $[t_1, t_2)$, we have

$$t2 - t1 \leq X_k \frac{2C^k L_M}{R} \leq t2 - t1 + \frac{2C^k L_M}{R} \text{ and } t2 - t1 \leq X_m \frac{2C^m L_M}{R} \leq t2 - t1 + \frac{2C^m L_M}{R}.$$

Hence, $(X_k - 1)C^{k-m} \leq X_m \leq X_k C^{k-m} + 1$. □

Lemma 10: Let $f_i \in F_k$ and $f_j \in F_m$ be continuously backlogged during $[t_1, t_2)$. $k \geq m$. Let X_k and X_m be the smallest numbers of F_k and F_m frames that completely enclose $[t_1, t_2)$. Assume that the inter-class scheduler is GPS,

$$(X_k - 1)C^{k-m} \leq X_m \leq X_k C^{k-m} + 1.$$

Proof: See appendix. □

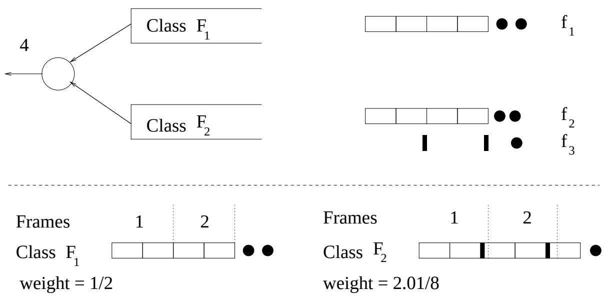


Figure 3: An example

We will use an example to illustrate how $LDRRWA$ interacts with inter-class scheduling to deliver fairness among flows in different classes. Let us assume that GPS is the inter-class scheduling algorithm. Consider scheduling for a link with 4 units of bandwidth with the following settings. $C = 2$ and there are two classes where $F_1 = \{f_i : \frac{1}{2} \leq w_i < 1\}$ and $F_2 = \{f_i : \frac{1}{4} \leq w_i < \frac{1}{2}\}$. Three flows, f_1, f_2 and f_3, with rates $r_1 = 2$ and $r_2 = r_3 = 1$ are in the system. $w_1 = 1/2$, $w_2 = 1/4$, and $w_3 = 1/4$. Thus, f_1 is in F_1, and f_2 and f_3 are in F_2. Let L_M be the maximum packet size. The quantum for each of the three flows is $2L_M$. All packets in f_1 are of size L_M, all packets in f_2 are of size $0.99L_M$ and all packets in f_3 are of size $0.01L_M$. Flows f_1

and f_2 are always backlogged. Flow f_3 is not always backlogged, its packets arrive in such a way that exactly one packet arrives before a new frame is to be formed. Thus, each F_2 frame contains one packet from f_3. The example is depicted in Figure 3. As shown in the figure, each F_1 frame contains exactly two packets from f_1. For F_2, the lookahead operation always moves part of the f_2 packet in the next DRR round into the current frame, and thus, the frame boundaries are not aligned with packet boundaries.

The weight for F_1 is always $1/2$. For F_2, the lookahead operation ensures that the size of f_2 data in a frame is $2L_M$, and thus, the size of each F_2 frame is $2L_M + 0.01L_M = 2.01L_M$. The weight of F_2 is computed as $W_2 = \frac{framesize}{2C^kL_M} = \frac{2.01L_M}{8L_M} = \frac{2.01}{8}$. Hence, F_1 (and thus f_1) is allocated a bandwidth of $4 * \frac{\frac{1}{2}}{\frac{1}{2}+\frac{2.01}{8}} = \frac{16}{6.01} > 2$. F_2 is allocated a bandwidth of $4 * \frac{\frac{2.01}{8}}{\frac{1}{2}+\frac{2.01}{8}}$. For each F_2 frame of size $2.01L_M$, $2L_M$ belongs to f_2. Thus, the rate allocated to f_2 is $4 * \frac{\frac{2.01}{8}}{\frac{1}{2}+\frac{2.01}{8}} * \frac{2L_M}{2.01L_M} = \frac{8}{6.01} > 1$. The rates allocated to f_1 and f_2 are larger than their guaranteed rates and the worst-case fairness is honored. The ratio of the rates allocated to f_1 and f_2 is equal to $\frac{\frac{16}{6.01}}{\frac{8}{6.01}} = 2$, which is equal to the ratio of their weights. Thus, the proportional fairness is also honored.

4.2 Inter-class scheduling

$LDRRWA$ assigns different weights for different frames in a class. Moreover, the weight of a frame is decided only after the last packet in the previous frame is sent. Hence, the inter-class scheduling must be able to handle these situations while achieving fair sharing of bandwidth. Although GPS can achieve fair sharing, none of the existing timestamp based schemes can closely approximate GPS under such conditions. We develop a new scheme called Dynamic Weight Worst-case Fair weighted Fair Queuing (DW^2F^2Q). DW^2F^2Q has the same scheduling result as WF^2Q [2] when the weights do not change. Theorems presented later show that the difference between the packet departure times under DW^2F^2Q and GPS is at most $(n-1)L_M$, where n is the number of classes. This bound is sufficient for FRR to achieve its QoS performance bounds.

DW^2F^2Q uses the virtual time concept in [12] to track the GPS progress up to the point that it can accurately track, and schedules packets based on their virtual starting/finishing times. Let us denote an event in the system the following: (1) the arrival of a packet to the GPS server, (2) the departure of a packet from the GPS server, and (3) the weight change of a class ($LDRRWA$ may change weight within a packet). Let t_j be the time at which the jth event occurs. Let the time of the first arrival of a busy period be denoted as $t_1 = 0$. For each $j = 2, 3, ...$, the set of classes that are busy in the interval $[t_{j-1}, t_j)$ is denoted as B_{j-1}. Let us denote $W_{k,j-1}$ the weight for class F_k during the interval $[t_{j-1}, t_j)$, which is a fixed value. Virtual time $V(t)$ is defined to be zero for all times when the system is idle. Assuming that each busy period begins with time 0, $V(t)$ evolves as follows:

$$V(0) = 0$$
$$V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{k \in B_{j-1}} W_{k,j-1}}, 0 < \tau \leq t_j - t_{j-1}, j = 2, 3, ...$$

As discussed in [12], the rate of change of V, $\frac{\partial V(t_j+\tau)}{\partial \tau}$, is $\frac{1}{\sum_{k \in B_j} W_{k,j}}$, and each backlogged class F_k receives service at rate $W_{k,j}\frac{\partial V(t_j+\tau)}{\partial \tau}$. Let us denote the virtual starting time, virtual finishing time, real arrival time, and size of the i-th packet P_k^i in F_k as $Vstart(P_k^i)$, $Vfinish(P_k^i)$, $arrive(P_k^i)$, and $size(P_k^i)$, respectively. Let W_k be the weight for the frame that includes P_k^i. We have

$$Vstart(P_k^i) = max(Vfinish(P_k^{i-1}), V(arrive(P_k^i)))$$
$$Vfinish(P_k^i) = Vstart(P_k^i) + \frac{size(P_k^i)}{W_k}$$

DW^2F^2Q keeps track of B_j in order to track the progress of the virtual time. When the last packet in a frame is sent later than its virtual finishing time (the packet departed under GPS, but not under DW^2F^2Q), the weight of the class after the frame virtual finishing time is unknown (until the last packet is sent). Hence, DW^2F^2Q cannot always track the virtual time up to the current time. Before each scheduling decision is made, DW^2F^2Q tracks the virtual time either to the earliest time when there is a class with an unknown weight or to the current time when the weights of all classes are known up to the current time. We will denote this time (the latest time that DW^2F^2Q can track GPS progress accurately) T and the corresponding virtual time $V(T)$. Hence, either T is the current time, or there exists one class F_k whose current frame virtual finishing time is $V(T)$ and the last packet in that frame has not been sent. DW^2F^2Q only deals with n classes. As a result, it can afford to recompute timestamps for the first packets in all classes in every scheduling step without introducing excessive overheads ($O(n) = O(1)$ under our assumption). To ease composition, we will ignore the issue related to the timing to assign the timestamps to the packets since the timestamps can be recomputed before each scheduling decision is made. DW^2F^2Q has exactly the same criteria as WF^2Q for scheduling packets: (1) only packets whose virtual starting times are earlier than the current virtual time are eligible; and (2) among all eligible packets, the one with the smallest virtual finishing time is selected.

There are potentially two problems in DW^2F^2Q. First, determining the virtual finishing time for the last packet in a frame can be a problem when only a part of the packet belongs to the current frame. The weight for the part of the packet in the next frame is unknown until the packet is scheduled. DW^2F^2Q assigns the frame virtual finishing time as the packet virtual finishing time for this type of packets. Although this creates some inaccuracy, the scheduling results are still sufficiently good as will be shown later.

The other problem is that T may not be the current time and the virtual time corresponding to the current time is unknown. In this case, there must exist one class F_k whose current frame virtual finishing time is $V(T)$ and the last packet in that frame, P, has not been sent. The virtual finishing time (and the virtual starting time) of P must be less than or equal to $V(T)$. Since DW^2F^2Q has accurate virtual time up to time T, the timestamps for all packets with finishing times less than $V(T)$ are available. All unknown virtual finishing times for packets must be larger than $V(T)$. In this case, the packet to be scheduled must have a virtual finishing time less than or equal to $V(T)$. Hence, DW^2F^2Q can simply assign a large timestamp as the virtual finishing time for packets with unknown virtual finishing times (to prevent these packets to be scheduled) and only consider packets whose virtual finishing time is less than or equal to V(T) when T is less than the current time. Not being able to tracking virtual time up to the current time does not prevent DW^2F^2Q from selecting the right packet for transmission. Note that when T equals the current time, DW^2F^2Q schedules packets exactly like WF^2Q.

The packet-by-packet operations of DW^2F^2Q are given in Appendix B, where we show that the worst-case per packet scheduling complexity is $O(nlg(n))$. In practical cases, $n = O(1)$ and hence, the per packet complexity of DW^2F^2Q is also $O(1)$. The following theorems shows properties of DW^2F^2Q.

Theorem 1: DW^2F^2Q is work conserving.

Proof: See appendix. □

Since both GPS and DW^2F^2Q are work-conserving disciplines, their busy periods coincide. We will consider packet scheduling within one busy period. Let $F_{i,s}^k$ be the departure time of the kth packet in class i under server s in a busy period.

Lemma 11: If $F_{i,GPS}^k \leq F_{j,GPS}^m$, $F_{i,DW^2F^2Q}^k < F_{j,DW^2F^2Q}^{m+1}$.

Proof: Let p_i^l be the packet at the head of class i at time t when p_j^{m+1} is at the head of class j and is eligible to be transmitted. Let the timestamp assigned to p_j^{m+1} be VT, we have $VT > V(F_{j,GPS}^m)$. This applies even when p_j^{m+1} is the last packet in a frame and is assigned an inaccurate timestamp.

If $l > k$, we have $F^k_{i,DW^2F^2Q} < F^{m+1}_{j,DW^2F^2Q}$ and the lemma is proved. If $l \leq k$, $F^l_{i,GPS} < F^{l+1}_{i,GPS} < ... < F^k_{i,GPS} \leq F^m_{j,GPS}$ and $V(F^l_{i,GPS}) < V(F^{l+1}_{i,GPS}) < ... < V(F^k_{i,GPS}) \leq V(F^m_{j,GPS}) < VT$. For a packet p^X_i that is in the frame boundary, its timestamp is less than or equal to $V(F^X_{i,GPS})$. Since at time t, p^{m+1}_j is eligible for scheduling, $V(t) \geq V(F^m_{j,GPS})$ and the accurate virtual times for these packets are available, all of these packets have smaller virtual starting and finishing times than p^{m+1}_j and will depart before p^{m+1}_j under DW^2F^2Q. Thus, $F^k_{i,DW^2F^2Q} < F^{m+1}_{j,DW^2F^2Q}$. $\square$

Lemma 11 indicates that DW^2F^2Q can at most introduce one packet difference between any two classes in comparison to GPS. This leads to the following theory that states relation of GPS departure time and DW^2F^2Q departure time. Let F^p_s be the time packet p departs under server s.

Theorem 2: Let n be the number of classes in the system,

$$F^p_{DW^2F^2Q} - F^p_{GPS} \leq (n-1)\frac{L_M}{R}.$$

Proof: Consider any busy period and let the time that it begins be time zero. Let p_k be the kth packet of size s_k to depart under GPS. We have $F^{p_k}_{GPS} \geq \frac{s_1+s_2+...+s_k}{R}$. Now consider the departure time of p_k under DW^2F^2Q. From Lemma 11, each class can have at most one packet whose GPS finishing time is after packet p_k and whose DW^2F^2Q finishing time is before packet p_k. Hence, there are at most $n-1$ packets (from the $n-1$ other classes) that depart before packet p_k under DW^2F^2Q and have a GPS finishing time after $F^{p_k}_{GPS}$. Let the $n-1$ packets be e_1, e_2, ..., e_{n-1} with sizes se_1, se_2, ..., se_{n-1}. All other packets depart before p^k under DW^2F^2Q must have GPS finishing times earlier than $F^{p_k}_{GPS}$. We have $F^{p_k}_{DW^2F^2Q} \leq \frac{s_1+...+s_k+se_1+...+se_{n-1}}{R}$. Thus, $F^{p_k}_{DW^2F^2Q} - F^{p_k}_{GPS} \leq (n-1)\frac{L_M}{R}$. $\square$

Theorem 2 indicates that, under the assumption that n is a small constant, the difference in the packet departure times under DW^2F^2Q and GPS is within a constant number of packets.

5 Properties of FRR

This session formally analyzes the QoS properties of FRR. We will prove that the three statements in Lemma 2 hold for FRR with an arbitrary weight distribution. As will be shown in the following theorems, even when the weight different factor D is large or related to N (e.g. $D = O(N)$), using FRR, one can select a constant C (e.g. $C = 2$) that is independent of D or N and enjoy QoS performance bounds that are linear functions of C and n. As discussed earlier, we consider practical cases when n is an $O(1)$ constant, but will express the bounds in terms of C and n.

Theorem 3 (single packet delay bound): Let packet p arrives at the head of flow $f_i \in F_k$ at time t. Using FRR, there exists a constant c_1 ($c_1 = O(C+n)$), such that p will depart before $t + c_1 * \frac{L_M}{r_i}$.

Proof: If class F_k is idle under GPS at time t, a new frame that includes packet p will be formed at time t. From Lemma 5, under GPS, the frame will be served at most at time $t + 2C^k\frac{L_M}{R} \le t + 2C\frac{L_M}{r_i}$. Hence, from Theorem 2, the frame will be served under DW^2F^2Q before $t+2C\frac{L_M}{r_i}+(n-1)\frac{L_M}{R} \le t+(2C+n-1)\frac{L_M}{r_i}$, where n is the number of classes in the system. Thus, there exists $c_1 = 2C+n-1 = O(C+n)$ such that packet departs before $t + c_1 * \frac{L_M}{r_i}$.

If class F_k is busy under GPS at time t, packet p will be included in the next frame. From Lemma 5, $F^p_{GPS} \le t + 2 * \frac{2L_M C^k}{R} \le t + \frac{4CL_M}{r_i}$. From Theorem 2, the frame will be served under DW^2F^2Q before $t + 4C\frac{L_M}{r_i} + (n-1)\frac{L_M}{R} \le t + (4C+n-1)\frac{L_M}{r_i}$. Thus, there exists $c_1 = 4C+n-1 = O(C+n)$ such that packet p departs before $t + c_1 * \frac{L_M}{r_i}$. $\Box$

Theorem 4 (worst-case fairness): FRR has a constant ($O(C+n)$) normalized worst-case fairness index.

Proof: Let a packet belonging to flow $f_i \in F_k$ arrive at time t, creating a total backlog of q_i bytes in f_i's queue. Let packet p_1 be the first packet in the backlog. From the proof of Theorem 3, we have $F^{p_1}_{GPS} \le t + 4C\frac{L_M}{r_i}$. After the first packet is served under GPS, from Lemma 3, at most $\lceil \frac{q_i}{Q_i} \rceil + 3 \le \frac{q_i}{Q_i} + 4$ frames will be needed to drain the queue. From Lemma 5, under GPS, servicing the $\frac{q_i}{Q_i} + 4$ frames will take at most

$$(\frac{q_i}{Q_i} + 4) * 2C^k\frac{L_M}{R} = \frac{q_i}{C^k w_i L_M}\frac{C^k L_M}{R} + 8C^k\frac{L_M}{R} \le \frac{q_i}{r_i} + 8C\frac{L_M}{r_i}$$

Thus, under GPS, the queue will be drained before $t + \frac{q_i}{r_i} + 4C\frac{L_M}{r_i} + 8C\frac{L_M}{r_i}$. From Theorem 2, under DW^2F^2Q, the queue will be drained before $t + \frac{q_i}{r_i} + 12C\frac{L_M}{r_i} + (n-1)\frac{L_M}{R}$. Thus, there exists a constant $d = 12C + n - 1 = O(C+n)$ such that the queue will be drained before $t + \frac{q_i}{r_i} + d\frac{L_M}{r_i}$ and the normalized worst-case fair index for FRR is $\max_i\{\frac{r_i * d\frac{L_M}{r_i}}{R}\} = d\frac{L_M}{R}$. $\Box$

The normalized worst-case fair index for FRR is $(12C+n-1)\frac{L_M}{R}$. While this constant is still quite large, it significantly improves over $STRR$ whose index is $\Omega(N)$. Next we will consider FRR's proportional fairness.

Lemma 12: Assuming that $f_i \in F_k$ and $f_j \in F_m$ are continuously backlogged during $[t_1, t_2)$, $k \ge m$. Assume that the inter-class scheduler is GPS and the intra-class scheduler is $LDRRWA$. Let $S_i(t_1, t_2)$ be the services given to flow f_i during $[t_1, t_2)$ and $S_j(t_1, t_2)$ be the services given to flow f_j during $[t_1, t_2)$. There exists two constants c_1 and c_2 ($c_1 \leq O(C)$ and $c_2 \leq O(C)$) such that

$$|\frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j}| \leq \frac{c_1 * L_M}{r_i} + \frac{c_2 * L_M}{r_j}.$$

Proof: Let X_k and X_m be the smallest numbers of F_k and F_m frames that completely enclose $[t_1, t_2)$. Since f_i and f_j are continuously backlogged during the $[t_1, t_2)$ period, from Lemma 3, the services given to f_i and f_j during this period satisfy:

$$(X_k - 3)Q_i \leq S_i(t_1,t_2) \leq (X_k+1)Q_i \text{ and } (X_m - 3)Q_j \leq S_j(t_1,t_2) \leq (X_m+1)Q_j.$$

The conclusion follows by manipulating these in-equations and applying Lemma 10, which gives the relation between X_k and X_m, $(X_k - 1)C^{k-m} \leq X_m \leq X_k C^{k-m} + 1$.

In the following, we will derive the bound for $\frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j}$.

$$\begin{aligned}
\frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j} &\leq \frac{(X_k+1)Q_i}{r_i} - \frac{(X_m-3)Q_j}{r_j} \leq \frac{Q_i X_k}{r_i} - \frac{Q_j X_m}{r_j} + \frac{Q_i}{r_i} + \frac{3Q_j}{r_j} \\
&\leq \frac{Q_i X_k}{r_i} - \frac{Q_j (X_k - 1)C^{k-m}}{r_j} + \frac{2CL_M}{r_i} + \frac{6CL_M}{r_j} \\
&= \frac{Q_i X_k}{r_i} - \frac{Q_j (X_k)C^{k-m}}{r_j} + \frac{Q_j C^{k-m}}{r_j} + \frac{2CL_M}{r_i} + \frac{6CL_M}{r_j}
\end{aligned}$$

We have $\frac{Q_j C^{k-m}}{r_j} = \frac{C^m w_j L_M C^{k-m}}{w_j R} \leq \frac{C * L_M}{r_i}$ and $\frac{Q_i X_k}{r_i} - \frac{Q_j (X_k) C^{k-m}}{r_j} = \frac{C^k w_i L_M X_k}{w_i R} - \frac{C^m w_j L_M X_k C^{k-m}}{w_j R} = 0$. Thus, $\frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j} \leq \frac{3CL_M}{r_i} + \frac{6CL_M}{r_j}$. The bound for $\frac{S_j(t_1,t_2)}{r_j} - \frac{S_i(t_1,t_2)}{r_i}$ can be derived in a similar fashion. Hence, there exists two constants c_1 and c_2, $c_1 = O(C)$ and $c_2 = O(C)$, such that $|\frac{S_i(t_1,t_2)}{r_i} - \frac{S_j(t_1,t_2)}{r_j}| \leq \frac{c_1 * L_M}{r_i} + \frac{c_2 * L_M}{r_j}$.□

Lemma 12 shows that if GPS is used as the inter-class scheduling algorithm, the scheduling algorithm provides proportional fairness. Since DW^2F^2Q closely approximates GPS (at most one packet can be out-of-order between any two classes (Lemma 11)), we will show in the next theorem that FRR, which uses DW^2F^2Q as the inter-class scheduling algorithm, also supports proportional fairness.

Theorem 5 (proportional fairness): In any time period $[t_1, t_2)$ during which flows $f_i \in F_k$ and $f_j \in F_m$ are continuously backlogged in FRR. There exists two constants $c_1 = O(C)$ and $c_2 = O(C)$ such that $|\frac{S_{i,FRR}(t_1,t_2)}{r_i} - \frac{S_{j,FRR}(t_1,t_2)}{r_j}| \leq \frac{c_1 * L_M}{r_i} + \frac{c_2 * L_M}{r_j}$.

Proof: See appendix. □

6 Simulation experiments

We compare FRR with Weighted Fair Queueing (WFQ) and two recently proposed deficit round robin (DRR) based schemes, Smoothed Round Robin (SRR) [7] and STratified Round Robin ($STRR$) [13]. All experiments are performed using *ns-2* [11], to which we added WFQ, $STRR$, and FRR queuing classes. Figure 4 shows the network topology used in the experiments. All links have a bandwidth of $2Mbps$ and a propagation delay of $1ms$. In all experiments, CBR flows have a fixed packet size of 210 bytes, and all other background flows have a fixed packet size uniformly chosen between 128 and 1024 bytes. Except for the experiment summarized in Figure 8 where only CBR and deterministic flows are considered, for all other experiments, we report the results using the confidence interval with a 99% confidence level.

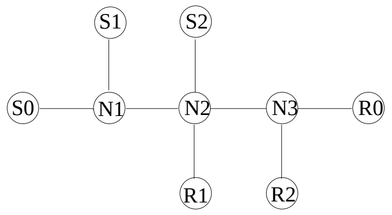


Figure 4: Simulated network.

Figure 5 shows the average end-to-end delays for flows with different rates. Figure 6 shows the worst-case end-to-end delays. In the experiment, there are 10 CBR flows from $S0$ to $R0$ with average rates of $10Kbps$, $20Kbps$, $40Kbps$, $60Kbps$, $80Kbps$, $100Kbps$, $120Kbps$, $160Kbps$, $200Kbps$, and $260Kbps$. The average packet delays of these CBR flows are measured. The background traffic in the system is as follows. There are five exponential on/off flows from $S1$ to $R1$ with rates $60Kbps$, $80Kbps$, $100Kbps$, $120Kbps$, and $160Kbps$.

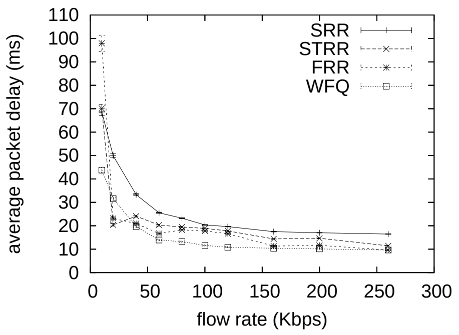


Figure 5: Average end-to-end delay.

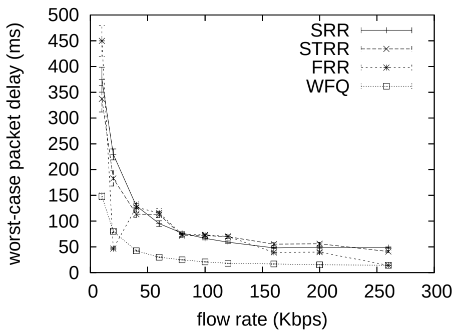
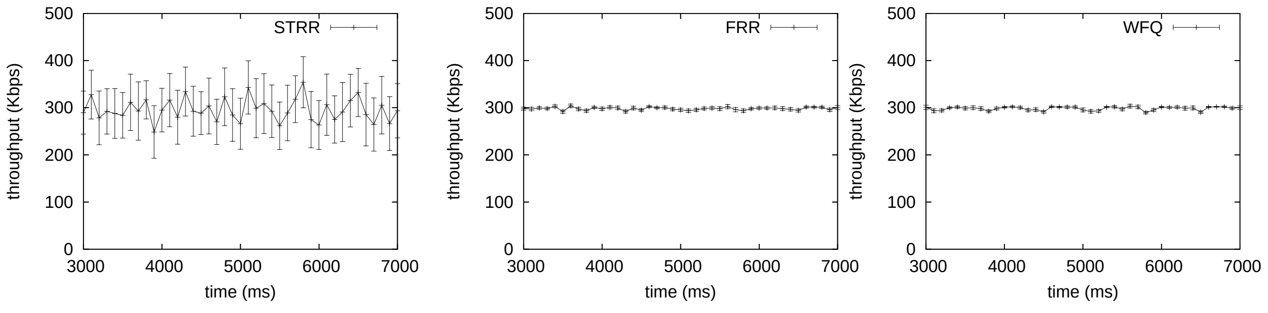


Figure 6: Worst-case end-to-end delay

The on-time and the off-time are 0.5 second. There are five Pareto on/off flows from $S2$ to $R2$ with rates $60Kbps$, $80Kbps$, $100Kbps$, $120Kbps$, and $160Kbps$. The on-time and the off-time are 0.5 second. The shape parameter of the Pareto flows is 1.5. Two $7.8Kbps$ FTP flows with infinite traffic are also in the system, one from $S1$ to $R1$ and the other one from $S2$ to $R2$.



(a) $STRR$ (b) FRR (c) WFQ

Figure 7: Short-term throughput

In this experiment, all of the three deficit round-robin based schemes, SRR, $STRR$, and FRR, give reasonable approximation of WFQ for all the flows with different rates. In comparison to SRR and $STRR$, FRR achieves average and worst-case end-to-end delays that are closer to the ones for WFQ for flows with large rates ($\geq 150Kbps$ in the experiment). In FRR, the timestamp based inter-class scheduling mechanism is added on top of DRR so that flows with small rates do not significantly affect flows with large rates. Thus, in a way, FRR gives preference to flows with larger weights in comparison to other DRR bases schemes.

Figure 7 (a), (b), and (c) shows the short-term throughput achieved by different schemes. Since the results for SRR are very similar to those for $STRR$, we only show the results for $STRR$. In this experiment, the short-term throughput in an interval of 100ms is measured. We observe one $300Kbps$ CBR flow and one $600Kbps$ flow from $S0$ to $R0$. In addition, 50 $10Kbps$ CBR flows from $S0$ and $R0$ are introduced. Other background flows are the same as the previous experiment.

Figure 7 (a), (b), and (c) shows the results for the 300 $Kbps$ flow. The results for the 600 $Kbps$ flow have a similar trend. As can be seen from the figure, the short term throughput for $STRR$ (and SRR) exhibit heavy fluctuations. On the other hand, WFQ and FRR yield much better short term throughput: within each interval of 100ms, the throughput is always close to the ideal rate. This demonstrates that FRR has a much better short-term fairness property than SRR and $STRR$.

Figure 8 shows the proportional fairness of FRR. In this experiment, we observe four deterministic flows

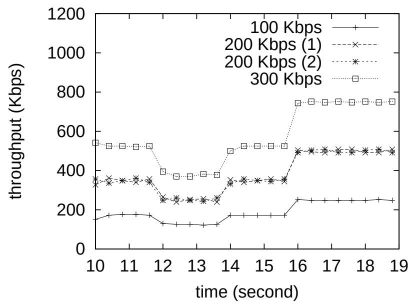


Figure 8: Proportional fairness

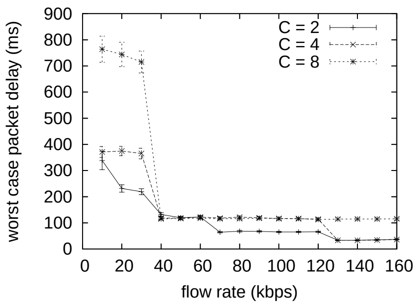


Figure 9: Worst-case end-to-end delay.

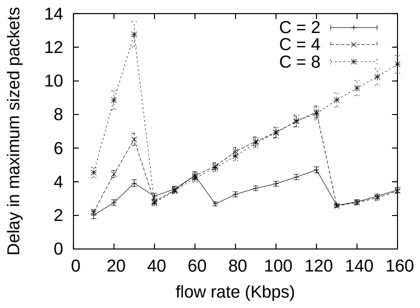


Figure 10: Delay in terms of numbers of packets

from $S0$ to $R0$ with average rates of 100Kbps, 200Kbps, 200Kbps, and 300Kbps. These flows follow an off/on pattern with each off/on period being 10 seconds. Hence, the flows are quiet for 10 seconds and then send in a doubled rate for the next 10 seconds. One 600Kbps CBR flow from $S1$ to $R1$ is introduced in period [10s, 16s] and another 400Kbps CBR flow from $S1$ to $R1$ is introduced in period [12s, 14s]. The CBR flows and the observed flows share the link from $N1$ to $N2$. The bandwidth allocation in the link from $N1$ to $N2$ to each of the flows during period [10s, 19s] is showed in Figure 8. As can be see from the figure, for all periods with different traffics sharing the link, the bandwidth allocation to the four observed flows is exactly proportional to their reserved bandwidths.

The last experiment is designed to study the impacts of C, a parameter in FRR. The background traffics used in this experiment are the same as those in Figure 5. We observe the worst case end-to-end packet delay for 16 CBR flows from $S0$ to $R0$ with average rates of $10Kbps$, $20Kbps$, $30Kbps$, $40Kbps$, $50Kbps$, $60Kbps$, $70Kbps$, $80Kbps$, $90Kbps$, $100Kbps$, $110Kbps$, $120Kbps$, $130Kbps$, $140Kbps$, $150Kbps$, and $160Kbps$. When $C = 8$, there are two classes in the system, F_3 containing flows with rates $10Kbps$, $20Kbps$, and $30Kbps$, and F_2 containing the rest of the flows. When $C = 4$, there are three classes in the system, F_4 ($10Kbps$ to $30Kbps$), F_3 ($40Kbps$ to $120Kbps$), and F_2 ($130Kbps$ to $160Kbps$). When $C = 2$, there are 5 classes: F_8 ($10Kbps$), F_7 ($20Kbps$ and $30Kbps$), F_6 ($40Kbps$ to $60Kbps$), F_5 ($70Kbps$ to $120Kbps$), and F_4 ($130Kbps$ to $160Kbps$).

Figure 9 shows the worst case delay in milli-seconds. We can see that the worst case delay for flows within one class are similar, which is evidenced by the ladder shape curves in the figure. This is expected as the DRR based scheme is used for intra-class scheduling. The packet delay is directly related to C. A smaller C value results in a smaller worst case packet delay.

Figure 10 shows a different view of Figure 9. In this figure, we represent the absolute worst case delay time as the time to send a number of packets (packets are of the same size, $210B$, in this experiment). This allows the delay to be normalized by the flow rate. There are two interesting observations in Figure 10. First, within each class, FRR biases against flows with larger weights. This is due to the use of a DRR based scheme for intra-class scheduling. Biasing against flows with large weights is a common problem for all DRR based schemes. However, in FRR, this problem is limited since the weight difference within a class is bounded. Second, FRR treats different classes fairly. It can be seen that for flows in different classes, the worst case packet delays fall in ranges with similar lower bounds and upper bounds as shown in the seesaw shape curve (e.g. when $C = 2$).

7 Conclusion

In this paper, we describe Fair Round Robin (FRR), a low quasi-$O(1)$ complexity round robin scheduler that provides proportional and worst-case fairness. In comparison to other DRR based scheduling schemes, FRR has similar complexity and proportional fairness, but better worst-case fairness. The simulation study demonstrates that even in average cases, FRR has better short-term behavior than other DRR based schemes, including smoothed round robin and stratified round robin. The constant factors in the complexity and QoS performance bounds for FRR are still fairly large. Recent improvements on GPS tracking [19, 20, 23] and DRR implementations [8] may be applied to improve FRR.

References

[1] J. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," *ACM/IEEE Trans. on Networking*, 5(5):675-689, Oct. 1997.

[2] J. Bennett and H. Zhang, "WF^2Q: Worst Case Fair Weighted Fair Queuing", in *IEEE INFOCOM'96* (1996), pages 120-128.

[3] B. Caprita, J. Nieh, and W. Chan, "Group Round Robin: Improving the Fairness and Complexity of Packet Scheduling." *Proceedings of the 2005 Symposium on Architecture for Networking and Communications Systems* (ANCS'05), Princeton, New Jersey, 2005.

[4] S.Cheung and C. Pencea, "BSFQ: Bin Sort Fair Queuing," in *IEEE INFOCOM'02* (2002), pages 1640-1649.

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," in *ACM SIGCOMM'89* (1989), pages 1-12.

[6] S. Golestani, "A Self-clocked Fair Queueing Scheme for Broadband Applications", in *IEEE INFOCOM'94* (1994), pages 636-646.

[7] C. Guo, "SRR, an O(1) Time Complexity Packet Scheduler for Flows in MultiService Packet Networks," *IEEE/ACM Trans. on Networking*, 12(6):1144-1155, Dec. 2004.

[8] L. Lenzini, E. Mingozzi, and G. Stea, "Aliquem: a Novel DRR Implementation to Achieve Better Latency and Fairness at O(1) Complexity," in *IWQoS'02* (2002), pages 77-86.

[9] L. Lenzini, E. Mingozzi, and G. Stea, "Tradeoffs Between Low Complexity, Low Latency, and Fairness with Deficit Round-Robin Schedulers." *IEEE/ACM Trans. on Networking*, 12(4):681-693, April 2004.

[10] L. Massouli and J. Roberts, "Bandwidth Sharing: Objectives and Algorithms," *IEEE/ACM Trans. on Networking*, 10(3):320-328, June 2002.

[11] "The Network Simulator - ns-2," available at http://www.isi.edu/nsnam/ns.

[12] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Single Node Case," *IEEE/ACM Trans. on Networking*, 1(3):344-357, June 1993.

[13] S. Ramabhadran and J. Pasquale, "The Stratified Round Robin Scheduler: Design, Analysis, and Implementation." *IEEE/ACM Transactions on Networking*, 14(6):1362-1373, Dec. 2006.

[14] J. Rexford and A. Greenberg and F. Bonomi, "Hardware-Efficient Fair Queueing Architectures for High-Speed Networks," *IEEE INFOCOM'96*, (1996), pages 638-646.

[15] J. L. Rexford, F. Bonomi, A. Greenberg, A. Wong, “A Scalable Architecture for Fair Leaky-Bucket Shaping,” in *INFOCOM’97* (1997), pages 1056–1064.

[16] D. Stiliadis and A. Varma, “Design and Analysis of Frame-based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks,” in *ACM SIGMETRICS’96* (1996), pages 104-115.

[17] M. Shreedhar and G. Varghese, “Efficient Fair Queuing using Deficit Round Robin,” in *ACM SIGCOMM’95* (1995), pages 231-242.

[18] S. Suri, G. Varghese, and G Chandranmenon, “Leap Forward Virtual Clock: An $O(loglog\ N)$ Queuing Scheme with Guaranteed Delays and Throughput Fairness,” in *IEEE INFOCOM’97* (1997), pages 557-565.

[19] P. Valente, “Exact GPS Simulation with Logarithmic Complexity, and its Application to an Optimally Fair Scheduler,” in *ACM SIGCOMM’04* (2004), pages 269-280.

[20] J. Xu and R. J. Lipton, “On Fundamental Tradeoffs between Delay Bounds and Computational Complexity in Packet Scheduling Algorithms,” in *ACM SIGCOMM’02* (2002), pages 279-292.

[21] X. Yuan and Z. Duan, “FRR: a Proportional and Worst-Case Fair Round Robin Scheduler,” *IEEE INFOCOM*, pages 831-842 (Vol. 2), Miami, March 13-17, 2005.

[22] L. Zhang, “Virtual Clock: A New Traffic Control Scheme for Packet Switching Networks”, in *ACM SIGCOMM’90* (1990), pages 19-29.

[23] Q. Zhao and J. Xu, “On the Computational Complexity of Maintaining GPS Clock in Packet Scheduling,” in *IEEE INFOCOM’04* (2004), pages 2383-2392.

Appendix A: Proofs

Lemma 1: Assuming that flow f_i is backlogged during $[t_1, t_2)$. Let X be the smallest number of continuous DRR rounds that completely enclose $[t_1, t_2)$. The service received by f_i during this period, $S_{i,DRR}(t_1, t_2)$, is given by

$$(X-3)quantum_i \leq S_{i,DRR}(t_1, t_2) \leq (X+1)quantum_i.$$

Proof: Since X is the smallest number of continuous DRR rounds that completely enclose $[t_1, t_2)$, f_i is served in at least $X-2$ rounds. Thus, $S_{i,DRR}(t_1, t_2) \geq (X-2) * quantum_i - L_M \geq (X-3)quantum_i$ since we assume that $quantum_i \geq L_M$. On the other hand, f_i is served in at most all X rounds, in this case, the total number of data sent should be less than the total quantum generated during the rounds plus the left over from the previous DRR round, which is less than L_M. Thus, $S_{i,DRR}(t_1, t_2) \leq X * quantum_i + L_M \leq (X+1)quantum_i$. $\square$

Lemma 2: Let f_1, ..., f_N be the N flows in the system with guaranteed rates r_1, ...,r_N. $\sum_{i=1}^{N} r_i \leq R$. Let $r_{min} = \min_i\{r_i\}$ and $r_{max} = \max_i\{r_i\}$. Let $r_{max} = D * r_{min}$. Assume that D is a constant with respect to N and that DRR is used to schedule the flows with $Q_i = L_M * \frac{r_i}{r_{min}}$. The following statements are true. All constants in this lemma are with respect to N.

1. Let packet p arrive at the head of the queue for f_i at time t. There exists a constant c_1 ($c_1 = O(D^2)$) such that packet p will be serviced before $t + c_1 \times \frac{L_M}{r_i}$.

2. The normalized worst-case fair index of DRR in such a system is a constant c_1 ($c_1 = O(D^2)$).

3. Let f_i and f_j be continuously backlogged during any given time period $[t_1, t_2)$, there exists two constants c_1 and c_2 ($c_1 = O(D)$ and $c_2 = O(D)$) such that the normalized service received by the two flows during this period is bounded by $|\frac{S_{i,DRR}(t_1,t_2)}{r_i} - \frac{S_{j,DRR}(t_1,t_2)}{r_j}| \leq c_1 \frac{L_M}{r_i} + c_2 \frac{L_M}{r_j}$.

Proof: Since $N * r_{min} \leq r_1 + r_2 + ... + r_N \leq R$, $r_{min} \leq \frac{R}{N}$. $quantum_i = L_M * \frac{r_i}{r_{min}} \leq D * L_M$. Thus, the total size of a round is at most $\sum_{i=1}^{N}\{quantum_i + L_M\} \leq (D+1) * N * L_M$. The time to complete service in a round is at most $\frac{(D+1)N*L_M}{R} \leq (D+1) * \frac{L_M}{r_{min}} \leq (D+1) * \frac{L_M}{r_{max}/D} = D(D+1) * \frac{L_M}{r_{max}} \leq D(D+1) * \frac{L_M}{r_i}$.

Packet p arrives at the head of the queue for f_i time t. It takes at most two rounds for the packet to be serviced. There exists a constant $c_1 = 2 * D(D+1) = O(D^2)$ such that packet p will be serviced before $t + c_1 \times \frac{L_M}{r_i}$. This proves the first statement. Next, we will prove the second statement.

Let a packet belonging to flow f_i arrives at time t, creating a total backlog of size q_i in f_i's queue. From statement 1., there exists a constant c_1 such that the first packet in the queue will be serviced in $t + c_1 \times \frac{L_M}{r_i}$. After the first packet is serviced, there will be at most $\lceil \frac{q_i}{quantum_i} \rceil + 1 \leq \frac{q_i}{quantum_i} + 2$ rounds for the q_i data to be sent. During the $\frac{q_i}{quantum_i} + 2$ rounds, at most $(\frac{q_i}{quantum_i} + 2) * \sum_{j=1}^{N} quantum_j$ quanta are generated, and thus, at most $(\frac{q_i}{quantum_i} + 2) * \sum_{j=1}^{N} quantum_j + N * L_M$ data are sent since each flow can have at most L_M credits left from the previous round. Thus, the total time to complete the $\frac{q_i}{quantum_i} + 2$ rounds is at most

$$\begin{aligned}
\frac{(\frac{q_i}{quantum_i}+2)*\sum_{j=1}^{N} quantum_j + N*L_M}{R} &= (\frac{q_i}{quantum_i} + 2)\frac{\sum_{j=1}^{N} quantum_j}{R} + \frac{N*L_M}{R} \\
&= (\frac{q_i}{quantum_i} + 2)\frac{\sum_{j=1}^{N} L_M * \frac{r_j}{r_{min}}}{R} + \frac{N*L_M}{R} \\
&= (\frac{q_i}{quantum_i} + 2)\frac{L_M}{r_{min}}\frac{\sum_{j=1}^{N} r_j}{R} + \frac{N*L_M}{R}
\end{aligned}$$

$$\leq (\frac{q_i}{quantum_i}+2)\frac{L_M}{r_{min}}+\frac{N*L_M}{R} \leq (\frac{q_i}{L_M*\frac{r_i}{r_{min}}})\frac{L_M}{r_{min}}+\frac{3L_M}{r_{min}}$$
$$\leq \frac{q_i}{r_i}+\frac{3*L_M}{r_{min}} \leq \frac{q_i}{r_i}+\frac{3*L_M}{r_{max}/D} \leq \frac{q_i}{r_i}+\frac{3D*L_M}{r_i}$$

Thus, there exists a constant $c_2 = c_1 + 3D = O(D^2)$ such that the queue of size q_i will be sent before $t + \frac{q_i}{r_i} + c_2 * \frac{L_M}{r_i}$. This is true for all flows. The normalzied worst case fair index is $c_{DRR} = max_i\{\frac{r_i C_{i,DRR}}{R}\} = \frac{c_2 L_M}{R}$. This proves the second statement.

For any given time period, $[t_1, t_2)$, let f_i and f_j be backlogged during this period that is enclosed by X rounds. From Lemma 1, we have

$$(X-3)quantum_i \leq S_{i,DRR}(t_1,t_2) \leq (X+1)quantum_i$$
$$(X-3)quantum_j \leq S_{j,DRR}(t_1,t_2) \leq (X+1)quantum_j$$

By manipulating these inequations, it can be shown that there exist two constants $c_1 = c_2 = 4D = O(D)$, such that $|\frac{S_{i,DRR}(t_1,t_2)}{r_i} - \frac{S_{j,DRR}(t_1,t_2)}{r_j}| \leq c_1\frac{L_M}{r_i} + c_2\frac{L_M}{r_j}$. □

Lemma 3: Assuming that flow f_i is continuously backlogged during $[t_1, t_2)$. Let X be the smallest number of continuous $LDRRWA$ frames that completely enclose $[t_1, t_2)$. The service received by f_i during this period, denoted as $S_{i,LDRRWA}(t_1, t_2)$, is bounded by

$$(X-3)Q_i \leq S_{i,LDRRWA}(t_1,t_2) \leq (X+1)Q_i.$$

Proof: The notation $S_{i,LDRRWA}(t_1, t_2)$ is abused in this lemma since $LDRRWA$ does not decide the actual timing to service packets. In this lemma, $S_{i,LDRRWA}(t_1, t_2)$ denotes the amount of data for a continuously backlogged flow f_i in X continuous $LDRRWA$ frames (of a particular class) using any inter-class scheduling scheme.

Since f_i is continuously backlogged, it will try to send as many packets as possible in each frame. Since X frames enclose $[t_1, t_2)$, flow f_i will fully utilize at least $X-2$ frames (all but the first frame and the last frame). In the $X-2$ frames, $(X-2) \times Q_i$ credits are generated for flow f_i. The lookahead operation in the frame prior to the $X-2$ frames may borrow at most one packet, whose size is less than L_M, from f_i in the first of the $X-2$ frames and flow f_i in the last of the $X-2$ frames may pass at most L_M credits to the next frame. Note that the lookahead operation borrows at most one packet from each backlogged flow. Thus, $S_{i,LDRRWA}(t_1, t_2) \geq (X-2) \times Q_i - L_M - L_M$. Since $Q_i \geq 2L_M$, $S_{i,LDRRWA}(t_1, t_2) \geq (X-3) \times Q_i$.

On the other hand, f_i will be served in at most all the X frames, which produces $X \times Q_i$ credits for f_i during this period of time. Flow f_i in the frame prior to the X frames may have at most L_M left-over credits and the lookahead operation in the last of the X frames may borrow at most L_M credits from f_i in the next frame. Thus,

$$S_{i,LDRRWA}(t_1,t_2) \leq X \times Q_i + L_M + L_M \leq (X+1)Q_i.\Box$$

Lemma 10: Let $f_i \in F_k$ and $f_j \in F_m$ be continuously backlogged during $[t_1, t_2)$. $k \geq m$. Let X_k and X_m be the smallest numbers of F_k and F_m frames that completely enclose $[t_1, t_2)$. Assume that the inter-class scheduler is GPS,

$$(X_k - 1)C^{k-m} \leq X_m \leq X_k C^{k-m} + 1.$$

Proof: This lemma relaxes the condition in Lemma 10 by not requiring each class to be serviced with its GPS guaranteed rate. Since $f_i \in F_k$ and $f_j \in F_m$ be continuously backlogged during $[t_1, t_2)$, the sizes of all frames during this period are no smaller than L_M (Lemma 9). Let us partition the duration $[t_1, t_2)$ into smaller intervals $[a_1 = t_1, b_1)$, $[a_2 = b_1, b_2)$, ..., $[a_Y = b_{Y-1}, b_Y = t_2)$ such that within each interval $[a_h, b_h)$, $1 \leq h \leq Y$, the weights of all classes are fixed. Let F_1, ..., F_n be the n classes in the system. Let class F_k have weight w_k^h during interval $[a_h, b_h)$, $1 \leq h \leq Y$ (If F_k is not backlogged, $w_k^h = 0$). The amount of class F_k data sent during $[a_h, b_h)$ is thus,

$$\frac{w_k^h}{\sum_{i=1}^n w_i^h} R * (b_h - a_h).$$

Consider a reference scheduling system that contains three classes RF_k, RF_m, and RF_o. Let us use intervals $[aa_1 = t_1, bb_1)$, $[aa_2 = bb_1, bb_2)$, ..., $[aa_Y = bb_{Y-1}, bb_Y)$ to emulate the behavior of classes F_k and F_m during intervals $[a_1 = t_1, b_1)$, $[a_2 = b_1, b_2)$, ..., $[a_Y = b_{Y-1}, b_Y)$ respectively. Let rw_k^h be the weight for class RF_k during interval $[aa_h, bb_h)$, $1 \leq h \leq Y$. Let rw_m^h be the weight for class RF_m during interval $[aa_h, bb_h)$, $1 \leq h \leq Y$. Let rw_o^h be the weight for class RF_o during interval $[aa_h, bb_h)$, $1 \leq h \leq Y$. The weights and the duration of each interval are given as follows:

$$rw_k^h = w_k^h, rw_m^h = w_m^h, rw_o^h = 1 - w_k^h - w_m^h, 1 \leq h \leq Y$$

and

$$bb_h = aa_h + \frac{b_h - a_h}{\sum_{i=1}^n w_i^h}, 1 \leq h \leq Y.$$

It can be verified that the amount of classes RF_k and RF_m data sent in an interval $[aa_h, bb_h)$, $1 \leq h \leq Y$, is exactly the same as the amount of classes F_k and F_m data sent in an interval $[a_h, b_h)$, $1 \leq h \leq Y$, respectively.

In an interval $[aa_h, bb_h)$, $1 \leq h \leq Y$, let us further assume that Class RF_k has exactly the same sequence of packets as Class F_k has in interval $[a_h, b_h)$ and that Class RF_m has exactly the same sequence of packets as Class F_m has in interval $[a_h, b_h)$. The progress of classes F_k and F_m during $[t_1, t_2)$ is exactly the same as the progress of class RF_k and RF_m during $[aa_1, bb_Y)$

In the reference system, classes RF_m and RF_k are serviced with the GPS guaranteed rate during $[aa_1, bb_Y)$. Let RX_k and RX_m be the smallest numbers of RF_k and RF_m frames that completely enclose $[aa_1, bb_Y)$. From Lemma 10, $(RX_k - 1)C^{k-m} \leq RX_m \leq RX_k C^{k-m} + 1$.

Let X_k and X_m be the smallest number of F_k and F_m frames that completely enclose $[t_1, t_2)$. Since the progress of classes F_k and F_m during $[t_1, t_2)$ is exactly the same as the progress of class RF_k and RF_m during $[aa_1, bb_Y)$, we have $X_k = RX_k$ and $X_m = RX_m$. Thus, $(X_k - 1)C^{k-m} \leq X_m \leq X_k C^{k-m} + 1$. □

Theorem 1: DW^2F^2Q is work conserving.

Proof: Since GPS is work-conserving, we will prove the theorem by showing that DW^2F^2Q has the same idle and busy periods as GPS. Assuming that DW^2F^2Q and GPS have different idle and busy periods. Let t be the first occurrence when GPS and DW^2F^2Q are not in the same state. There are two cases.

Case 1: GPS is idle and DW^2F^2Q is busy, serving packet p. Since t is the first occurrence when GPS and DW^2F^2Q are not in the same state, the amount of data served during $[0, t)$ must be the same for the two scheduling schemes. Since p is currently being served under DW^2F^2Q, p must be started before t under GPS. Since GPS is idle at time t, packet p must finish before t under GPS. Hence, there must exist a packet q such that q has not been served under GPS during $[0, t)$ and has been served by DW^2F^2Q during $[0, t)$. Since GPS is idle at t, packet q should start after t under GPS, which indicates that q cannot be served under DW^2F^2Q during $[0, t)$. This is the contradiction.

Case 2: GPS is busy and DW^2F^2Q is idle. Let packets p_1, p_2, ..., p_i be the packets departed under GPS during $[0, t)$ and packets $cp_1, ..., cp_j$ be the packets currently in progress under GPS. Since GPS is busy, at least one packet is being serviced at time t. Since DW^2F^2Q is idle at t, all packets that starts before t under GPS should have been served, that is, packets p_1, p_2, ..., p_i and $cp_1, ..., cp_j$ are all served during $[0, t)$ under DW^2F^2Q. Thus, during $[0, t)$, DW^2F^2Q sends more data than GPS and t cannot be the first occurrence that GPS and DW^2F^2Q are not in the same state. □

Theorem 5 (proportional fairness): In any time period $[t_1, t_2)$ during which flows $f_i \in F_k$ and $f_j \in F_m$ are continuously backlogged in FRR. There exists two constants $c_1 = O(C)$ and $c_2 = O(C)$ such that

$|\frac{S_{i,FRR}(t_1,t_2)}{r_i} - \frac{S_{j,FRR}(t_1,t_2)}{r_j}| \le \frac{c_1 * L_M}{r_i} + \frac{c_2 * L_M}{r_j}$.

Proof: There are two cases. The first case is when flows f_i and f_j are in the same class, that is, $k = m$. The second case is when flows f_i and f_j are not in the same class, that is, $k \neq m$. The proof of the first case is similar to the proof of the statement 3 in Lemma 2. Here, we will focus on the second case. Let us assume that $k > m$.

Let packets p_k^1, p_k^2, ..., p_k^a be the sequence of class F_k packets sent under FRR during $[t_1, t_2)$. Let packets p_m^1, p_m^2, ..., p_m^b be the sequence of class F_m packets sent under FRR during $[t_1, t_2)$. Since flows f_i and f_j are continuously backlogged during $[t_1, t_2)$, there exists a packet p_k^0 that departed before p_k^1 and p_k^{a+1} that will depart after p_k^a. Under the simulated GPS, there is no idle time between packet p_k^0 and packet p_k^1 and between packet p_k^a and packet p_k^{a+1}. Packets p_m^0 and p_m^{b+1} are defined similarly.

Consider the progress of these packets under the simulated GPS. Let B_{GPS}^p denote the beginning time of packet p under GPS and F_{GPS}^p denote the finishing time of packet p under GPS. There are four cases: (1) $B_{GPS}^{p_k^1} \ge B_{GPS}^{p_m^1}$ and $F_{GPS}^{p_k^a} < F_{GPS}^{p_m^b}$, (2) $B_{GPS}^{p_k^1} \ge B_{GPS}^{p_m^1}$ and $F_{GPS}^{p_k^a} \ge F_{GPS}^{p_m^b}$, (3) $B_{GPS}^{p_k^1} < B_{GPS}^{p_m^1}$ and $F_{GPS}^{p_k^a} < F_{GPS}^{p_m^b}$, and (4) $B_{GPS}^{p_k^1} < B_{GPS}^{p_m^1}$ and $F_{GPS}^{p_k^a} \ge F_{GPS}^{p_m^b}$.

In the next, we will prove case (1). Other three cases can be proven in a similar fashion. Consider case (1) when $B_{GPS}^{p_k^1} \ge B_{GPS}^{p_m^1}$ and $F_{GPS}^{p_k^a} < F_{GPS}^{p_m^b}$. Let $tt_0 = B_{GPS}^{p_m^1}$, $tt_1 = B_{GPS}^{p_k^1}$, $tt_2 = F_{GPS}^{p_k^a}$, and $tt_3 = F_{GPS}^{p_m^b}$. We have $tt_0 \le tt_1 \le tt_2 \le tt_3$. Let $S_{i,GPS}(t_1, t_2)$ be the services that flow f_i received during time $[t_1, t_2)$ in the simulated GPS. We have $S_{i,FRR}(t_1, t_2) = S_{i,GPS}(tt_1, tt_2)$ and $S_{j,FRR}(t_1, t_2) = S_{j,GPS}(tt_0, tt_1) + S_{j,GPS}(tt_1, tt_2) + S_{j,GPS}(tt_2, tt_3)$.

In the simulated GPS system, flows f_i and f_j are continuously backlogged during $[tt_1, tt_2)$. From Lemma 12, there exist two constants cc_1 and cc_2 such that $|\frac{S_{i,GPS}(tt_1,tt_2)}{r_i} - \frac{S_{j,GPS}(tt_1,tt_2)}{r_j}| \le \frac{cc_1 * L_M}{r_i} + \frac{cc_2 * L_M}{r_j}$.

Thus,

$$\begin{aligned} |\frac{S_{i,FRR}(t_1,t_2)}{r_i} - \frac{S_{j,FRR}(t_1,t_2)}{r_j}| &\le |\frac{S_{i,GPS}(tt_1,tt_2)}{r_i} - \frac{S_{j,GPS}(tt_1,tt_2)}{r_j}| + \frac{S_{j,GPS}(tt_0,tt_1)}{r_j} + \frac{S_{j,GPS}(tt_2,tt_3)}{r_j} \\ &\le \frac{cc_1 * L_M}{r_i} + \frac{cc_2 * L_M}{r_j} + \frac{S_{j,GPS}(tt_2,tt_3)}{r_j} + \frac{S_{j,GPS}(tt_0,tt_1)}{r_j} \end{aligned}$$

Next, we will consider the two terms $\frac{S_{j,GPS}(tt_0,tt_1)}{r_j}$ and $\frac{S_{j,GPS}(tt_2,tt_3)}{r_j}$. First, consider class F_m packets serviced during $[tt_0, tt_1)$. Since all these packets are serviced after packet p_k^0 under FRR (DW^2F^2Q as the inter-class scheduler), from Lemma 3, at most one of the packets can have a GPS finishing time before $F_{GPS}^{p_k^0} = B_{GPS}^{p_k^1} = tt_1$. That is, there can be at most one class F_m packet finishing during $[tt_0, tt_1)$. Thus, in the simulated GPS, at most two class F_m packets can be serviced during $[tt_0, tt_1)$ and $\frac{S_{j,GPS}(tt_0,tt_1)}{r_j} \leq \frac{2L_M}{r_j}$. Now, consider class F_m packets serviced during $[tt_2, tt_3)$. Since all these packets are serviced under FRR before packet p_k^{a+1}, at most one of the packets can have a GPS finishing time after $F_{GPS}^{p_k^{a+1}}$. From Lemma 7, the duration of packet p_k^{a+1} is less than $\frac{C^m L_M}{R}$ in the simulated GPS, which is less than one frame whose size is larger than L_M. Let X be the number of frames for class F_m during this period when p_k^{a+1} is in progress under GPS. Since f_j is continuously backlogged during this period of time, from Lemma 11, $X \leq C^{k-m} * 1 + 1$. Thus, from Lemma 4, during the period that packet p_k^{a+1} is in progress under GPS, the amount of services given to flow f_j is at most $(C^{k-m} + 1 + 2)quantum_j$.

$$\frac{S_{j,GPS}(tt_2,tt_3)}{r_j} \leq \frac{(C^{k-m}+1+2)quantum_j+L_M}{r_j} = \frac{(C^{k-m}+3)w_jC^mL_M+L_M}{w_jR} \leq \frac{L_MC^k}{R} + \frac{3L_MC^m}{R} + \frac{L_M}{r_j}$$
$$\leq C\frac{L_M}{r_i} + (3C+1)\frac{L_M}{r_j}$$

Thus, there exists two constants $c_1 = cc_1 + C = O(C)$ and $c_2 = cc_2 + 2 + 3C + 1 = O(C)$ such that

$$|\frac{S_{i,FRR}(t_1,t_2)}{r_i} - \frac{S_{j,FRR}(t_1,t_2)}{r_j}| \leq \frac{c_1 * L_M}{r_i} + \frac{c_2 * L_M}{r_j}.\square$$

Appendix B: A packet-by-packet implementation of FRR

We will describe a packet-by-packet implementation of FRR. This is one of the potential implementations of FRR that are logical equivalent to the high level description in the main body of the paper. All operations for $LDRRWA$ and DW^2F^2Q are distributed to the operations at packet arrivals and departures. The main objective is to show that FRR can be realized with a psedo-$O(1)$ complexity ($O(1)$ assuming that the number of classes n is a constant).

LDRRWA Packet-by-packet operations

$LDRRWA$ is a minor variation of DRR. Let us assume that all active flows (flows with backlogged packets) are maintained in a queue. Realizing DRR is straight-forward: one just needs to go through each active flow in the queue and sends packets until either the quota is reached or there is no more packet in the flow; after that, the flow is moved to the end of the queue to be served in the next round. The main difference between $LDRRWA$ and DRR is that the total size of the packets in each flow that belongs to the next flame must be maintained as well as the total frame size (of the next frame). Our scheme uses three queues for realizing $LDRRWA$, a queue for active flows in the current flow, a queue for active flows in the next frame, and a queue for flows in the lookahead operation. We will call these three queues $DRRqueue$, $NFqueue$, and $LAqueue$ respectively. A continuously backlogged flow will likely move from $DRRqueue$ to $LAqueue$, to $NFqueue$, and then back to $DRRqueue$ again. When a packet arrives or departs, which may results in a flow being moved from a queue to another, being removed from the queue, being inserted into a queue, our scheme keeps tracks of the following information: the size of the remaining current frame, the size of the next frame, the size of the partial packet in the current frame, the starting time of the current frame, the deficit counter for each flow, the size of the data for each flow in the current frame, the size of the data for each flow in the next frame, the size of all backlogged data in a flow, and the last time that a flow is serviced. Clearly, bookkeeping for all these variables takes $O(1)$ operations. With all these information, the scheme knows the amount of data in each flow to be sent in the DDR round as well as the lookahead operation. In addition, the size of a new frame and the weight of the frame can be decided in $O(1)$ time. Hence, the whole $LDRRWA$ can be realized with $O(1)$ per packet processing overhead in the worst case.

Next, we will describe the operations when packets arrive and depart, which implement the algorithms in Figure 2. The following data structures are maintained in the $LDRRWA$ implementation. For each class F_k, there are three queues, $DRRqueue_k$ that is used to emulate the DRR operation, $LAqueue_k$ that is used to perform the lookahead operation, and $NFqueue_k$ that stores all active flows in the next frame. Variable W_k is used to store the weight for the current frame; $csize_k$ keeps track of the amount of data to be sent in the current frame; $nsize_k$ keeps track of the amount of data in the next round; $starttime_k$ records the starting time of the current round; and $remainsize_k$ stores the size of the partial packet in the current frame (part of the packet is counted in the previous frame). For each flow f_i, variable Q_i records the quantum for f_i; DC_i stores the deficit counter; $fsize_i$ is the size of f_i data in the current frame when f_i is in $DRRqueue_k$ and the size of f_i data in the next frame when f_i is in $LAqueue_k$ or $NFqueue_k$; $bsize_i$ is the size of all backlogged data; and $lastsend_i$ records the time that flow f_i was last served. These variables are summarized in Table 3.

The functions used in the scheduler include the following: $extractflow(pkt)$ extracts the flow id from packet pkt; $enqueue(i, pkt)$ inserts packet pkt in the queue for flow f_i; $dequeue(i, pkt)$ removes the first packet in f_i and stores it in pkt; $class(i)$ returns the class of flow f_i; $size(pkt)$ returns the size of packet pkt; $front(i)$ returns the information about the first packet in flow f_i.

$LDRRWA$ is depicted in Figures 11, 12, and 13. A frame in F_k starts when flows in $NFqueue_k$ are moved to $DRRqueue_k$ ($Activateclasses()$ in Figure 13). For flows in $DRRqueue_k$, $LDRRWA$ operates

variable	explanation
$DRRqueue_k$	queue for emulating DRR in class F_k
$LAqueue_k$	queue for the lookahead operation in F_k
$NFqueue_k$	queue for the flows in the next frame in F_k
W_k	weight for the current frame in class F_k
$csize_k$	size of the remaining (current) frame in F_k
$nsize_k$	size of the next frame in F_k
$remainsize_k$	size of the partial packet in the current frame
$starttime_k$	the starting time of the current frame in F_k
Q_i	quantum for flow f_i
DC_i	deficit counter for flow f_i
$fsize_i$	the size of f_i data in the current/next frame
$bsize_i$	the size of all backlog data in f_i
$lastsend_i$	the last time f_i was served

Table 3: Variables used in LDRRWA

the same as DRR. When a flow finishes its DRR round, if the flow does not use up its quantum and is still backlogged, it is inserted into $LAqueue_k$ for the lookahead operation. $LDRRWA$ continues sending packets from flows in $LAqueue_k$ until $csize_k = 0$. One flow in $LAqueue_k$ will at most send one packet in the lookahead operation. $LDRRWA$ maintains sufficient information to decide the frame size ($nsize_k$) for a class F_k and the data size ($fsize_i$) in the frame for each flow f_i when the frame is determined (in $Activateclasses()$).

```
enqueuing module (p): on the arrival of packet $p$

( 1) $i = extractflow(p); k = class(i);$
( 2) enqueue($i$, p);
( 3) if (ExistinNFqueueorLAqueue(i)) then
( 4)     if ($bsize_i < DC_i$) then
( 5)         $NFsize = min(DC_i, bsize_i + size(p)) - bsize_i;$
( 6)         $fsize_i = fsize_i + NFsize; nsize_k = nsize_k + NFsize;$
( 7)     end if
( 8)     $bsize_i = bsize_i + size(p);$
( 9) else if (ExistinDRRqueue(i)) then
(10)     $bsize_i = bsize_i + size(p);$
(11) else /* $f_i$ is inactive */
     /* keep negative $DC_i$ for a frame, reset otherwise */
(12)     if (($lastsend_i > starttime_k$) and ($DC_i < 0$))
(13)         $DC_i = DC_i + Q_i;$
(14)     else $DC_i = Q_i;$
(15)     $bsize_i = size(p); fsize_i = size(p); nsize_k = nsize_k + size(p);$
(16)     insert $f_i$ into $NFqueue_k$
(17) end if
```

Figure 11: Enqueuing module

Figure 11 shows the enqueuing module, which is executed on the arrival of a packet. This module first obtains the flow id (f_i), class id (F_k), and put the packet to the queue for f_i. After that, bookkeeping operations are performed to maintain $bsize_i$, $fsize_i$, $nsize_k$, and DC_i (when f_i just becomes active). $Bsize_i$ is updated in all cases. If f_i is in $NFqueue_k$ or $LAqueue_k$, the next frame may include this packet (if $bsize_i < DC_i$), and $nsize_k$ and $fsize_i$ are adjusted. $LDRRWA$ counts packets in $LAqueue_k$ in the next frame. If f_i is not in any queue (it was inactive before), f_i is inserted to $NFqueue_k$ to be served in the next frame and its deficit counter is updated. For a flow with a negative deficit counter, if this flow just turns inactive in the current frame ($lastsent_i > starttime_k$), $LDRRWA$ keeps the negative deficit (Line (12) and (14)). In other words, negative deficit is always kept for one frame, which prevents a flow from stealing credits by repeatedly sending more than its credits and then turning inactive. For all other cases, the deficit counter is reset. Note that the $ExistinNFqueueorLAqueue()$ and $ExistinDRRqueue()$ can be realized in $O(1)$ operations by associating flags with f_i. The enqueuing module has an $O(1)$ complexity.

The FRR scheduling module is shown in Figure 12. This module is executed every time when a packet departs and when the system is idle. At packet boundaries, the algorithm activates classes if necessary ($Activateclasses()$), calls the inter-class scheduling algorithm (to be discussed in the next sub-section), $DW^2F^2Q()$, to decide the class to be served. Once the class is decided, depending on whether the class is in the DRR phase (DRRqueue is not empty) or the Lookahead phase ($DRRqueue$ is empty), either $Send\_DRR\_packet(F_k)$ or $Send\_Lookahead\_packet(F_k)$ is called to send one packet from this class.

```
**Scheduling: on the departure of a packet or system idle**

(1) **if** (all flows are idling) {$V(T) = 0$; **return**};
(2) $Activateclasses()$;
(3) $F_k = DW^2F^2Q()$
(4) if ($DRRqueue_k$ is not empty) $Send\_DRR\_packet(F_k)$;
(5) else $Send\_lookahead\_packet(F_k)$;
```

Figure 12: FRR scheduling module

Functions $Activateclasses()$, $Send\_DRR\_packet(F_k)$ and $Send\_Lookahead\_packet(F_k)$, are shown in Figure 13. VS_k and VF_k in $Activateclasses()$ are used in inter-class scheduling and will be discussed later. All frames start when the $Activateclasses()$ is called: flows in $NFqueue_k$ are moved to $DRRqueue_k$, the start time of the frame is recorded, and the weight is set to $\frac{framesize}{2C^k L_M}$. Line (8) in this function adjusts the weight so that the weight for a class is no less than that for the smallest weight flow in the class. The

```
**Activateclasses()**
( 1) **for** (each class $F_k$) **do**
( 2)     **if** (($DRRqueue_k$ and $LAqueue_k$ are empty)
( 3)         $VS_k = VF_k = BigNUM$;
( 4)         **if** ($NFqueue_k$ is not empty)) **then**
( 5)             $DRRqueue_k = NFqueue_k$;$NFqueue_k = NULL$;
( 6)             $csize_k = nsize_k$; $nsize_k = 0$;
( 7)             $W_k = \frac{csize_k}{2C^k L_M}$;
( 8)             **if** ($W_k < \frac{1}{C^k}$) $W_k = \frac{1}{C^k}$;
( 9)             $starttime_k = current\_time()$;
(10)             $Do\_frame\_arrival(F_k)$;
(11)         **end if**
(12)     **end if**
(13) **end for**
```

```
**Send_DRR_packet($F_k$)**
( 1) Let $f_i$ be the first flow in $DRRqueue_k$.
( 2) dequeue(i, p); send(p);
( 3) $lastsend_i = current\_time()$;
( 4) $bsize_i = bsize_i - size(p)$; $DC_i = DC_i - size(p)$; $fsize_i = fsize_i - size(p)$;
( 5) **if** ($fsize_i = 0$) $DC_i = 0$;
( 6) $csize_k = csize_k - size(p)$; $remainsize_k = 0$;
( 7) **if** ($f_i$ is not backlogged) remove $f_i$ from $DRRqueue_k$
( 8) **else if** ($DC_i < size(front(f_i))$) **then**
( 9)     **if** ($DC_i == 0$) **then**
(10)         move $f_i$ from $DRRqueue_k$ to $NFqueue_k$;
(11)     **else** move $f_i$ from $DRRqueue_k$ to $LAqueue_k$
(12)     $DC_i = DC_i + Q_i$;
(13)     $fsize_i = min(DC_i, bsize_i)$;
(14)     $nsize_k = nsize_k + fsize_i$;
(15) **end if**
```

```
**Send_lookahead_packet($F_k$)**
( 1) Let $f_i$ be the first flow in $LAqueue_k$.
( 2) $dequeue(i, p)$; $send(p)$
( 3) $lastsend_i = current\_time()$;
( 4) $bsize_i = bsize_i - size(p)$;
( 5) $DC_i = DC_i - size(p)$; $fsize_i = fsize_i - size(p)$;
( 6) $nsize_k = nsize_k - size(p)$;$csize_k = csize_k - size(p)$;$remainsize_k = 0$;
( 7) **if** ($f_i$ is not backlogged) **then**
( 8)     remove $f_i$ from $LAqueue_k$;
( 9) **else** move $f_i$ from $LAqueue_k$ to $NFqueue_k$;
(10) **if** ($csize_k <= 0$) **then**
(11)     $remainsize_k = -csize_k$;
(12)     $nsize_k = nsize_k + remainsize_k$;
     /* counting the partial packet in the next frame */
(13)     append $LAqueue_k$ to $NFqueue_k$;
(14)     $LAqueue_k = NULL$;
(15) **end if**
```

Figure 13: Supporting functions in the scheduling module

$Do\_frame\_arrival()$ performs operations related to inter-class scheduling and will be discussed later. In $Send\_DRR\_packet()$, after each packet in f_i is sent, the values for DC_i, $fsize_i$, $bsize_i$, $lastsend_i$, and $csize_k$ are adjusted. If a flow f_i does not have enough data to use up its quantum in the current frame ($fsize_i = 0$ and $DC_i \neq 0$), it should lose the remaining credits. The flow is deactivated if it is no longer backlogged. When $0 \leq DC_i < size(front(f_i))$, the DRR round finishes for this flow. If $DC_i = 0$, the flow sends exactly its quantum in the frame and is moved to $NFqueue_k$. Otherwise, the flow is moved to $LAqueue_k$ for the lookahead operation. When a flow is moved to $LAqueue_k$, packets in the flow are counted in the next frame (the update of $fsize_i$ and $nsize_k$ in Lines (13) and (14)). The $Send\_lookahead\_packet$ is similar except that every time a packet is sent, the flow is moved from $LAqueue_k$ to $NFqueue_k$ (or removed if not backlogged). When $csize <= 0$, all data in the current frame have been sent, and the remaining flows in $LAqueue_k$ are move to $NFqueue_k$. In this case, $nsize_k$ does not need to be adjusted since packets in $LAqueue_k$ are already counted in the next frame. The value of $remainsize_k$ is always properly maintained in the routine.

The complexity of $Send\_DRR\_Packet()$ and $Send\_Lookahead\_Packet()$ is $O(1)$. The complexity of the $Activateclasses()$, excluding the operations in $Do\_frame\_arrival()$, is $O(n)$. Excluding the operations in inter-class scheduling ($DW^2F^2Q()$ and $Do\_frame\_arrival()$) the per-packet computational complexity is $O(n) = O(1)$.

DW^{2}F^{2}Q packet-by-packet operations

The details of inter-class scheduling in FRR, including $Do\_frame\_arrival()$ and $DW^2F^2Q()$, are shown in Figure 14 and Figure 15. The system maintains the latest time with accurate GPS virtual time T, the corresponding virtual time $V(T)$, and the sum of the weights of all active classes at time T, $Wsum_T$. $Wsum_T = \sum_{k \in B_j} W_{k,j}$ (the sum of the weights of all classes). For each class, four variables are maintains for inter-class scheduling: the current frame virtual finishing time $V(e_k)$, the weight W_k^T of F_k at time T, the virtual start time of the first packet in the class VS_k, and the virtual finishing time of the first packet in the class VF_k. For an inactive class F_k, VS_k and VF_k are assigned a large number ($BigNUM$ in Line (3) of $Activateclasses()$) to prevent packets in this class from been scheduled. Note that the first packet in a class is the first packet in the queue for the first flow in $DRRqueue_k$ when $DRRqueue_k$ is not empty, or the first packet in the queue for the first flow in $LAqueue_k$. If both $DRRqueue_k$ and $LAqueue_k$ are empty, the class is inactive. The inter-class scheduling also uses variables, W_k, $csize_k$, and $remainsize_k$, which are also used in intra-class scheduling.

The system uses two queues to help tracking the GPS virtual time: the frame finishing queue, $FFqueue$, and the new arrival queue, $NAqueue$. $NAqueue$ stores the newly arrived classes whose virtual starting time cannot be determined ($arrival\ time > T$). The entry in $NAqueue$ has two fields, $(k, arrivetime)$, where k is the class id, and $arrivetime$ is the real arrival time of the new frame. The $NAqueue$ is sorted by $arrivetime$ and can be maintained as a simple FIFO queue: new entries are inserted at the end of the queue and the item with the smallest arrive time is always at the front of the queue.

```
Do_frame_arrival($F_k$)
( 1) if ($V(T) = 0$) then /* starting a busy period */
( 2)     $VS_k = 0$;
( 3)     $VF_k = VS_k + size(first(F_k))/W_k$;
( 4)     $W_k^T = W_k$; $Wsum_T = Wsum_T + W_k$
( 5)     $V(e_k) = 0 + csize_k/W_k$;
( 6)     insert $(k, V(e_k), \perp)$ to $FFqueue$.
( 7) else if (the frame is a back-to-back frame) then
             /* the last packet sent is from $F_k$ */
( 8)     change $(k, V(e_k), \perp)$ in $FFqueue$ to $(k, V(e_k), W_k)$;
( 9)     $VS_k = V(e_k) + remainsize_k/W_k$;
(10)     $csize_k = csize_k - remainsize_k$;
(11)     $pktsize = min(csize_k, size(first(F_k)))$;
(12)     $VF_k = VS_k + pktsize/W_k$;
(13)     $V(e_k) = V(e_k) + csize_k/W_k$;
(14)     insert $(k, V(e_k), \perp)$ to $FFqueue$.
(15) else /* a newly active class */
(16)     insert $(k, current\_time())$ to $NAqueue$.
(17) end if
```

Figure 14: Operations when a frame arrives

The entry in the $FFqueue$ has three fields, $(k, V(e_k), next\_weight_k)$, where k is the class id, $V(e_k)$ is the virtual finishing time of the frame in F_k, and $next\_weight_k$ is the weight in the next frame. $FFqueue$ stores all known frame finishing times after time T ($V(e_k) \geq V(T)$). It is a priority queue sorted by $V(e_k)$. We will use notation $\perp$ to represent an unknown weight. An entry, $(k, V(e_k), \perp)$, is inserted in $FFqueue$ when a frame in F_k arrives **AND** $V(e_k)$ can be determined. When the last packet of a frame is sent, the weight for the next frame W is known (if there is no next frame, $W = 0$), the entry is modified from $(k, V(e_k), \perp)$ to $(k, V(e_k), W)$. An entry $(k, V(e_k), next\_weight_k)$ is removed (in the $track\_virtual\_time()$ routine in Figure 15) when $V(T) = V(e_k)$ and $next\_weight_k \neq \perp$. An entry $(k, V(e_k), \perp)$ can never be removed

($V(T)$ cannot pass this time since the future weight for F_k is unknown). For such an entry, the $next\_weight_k$ is always changed to a known value when the last packet is sent before it can be removed. Since a new frame for a class is formed only after the last packet of the current frame is sent, there can be at most two entries for each class F_k in $FFqueue$ and the size of $FFqueue$ is at most $2n$. We assume that each class has pointers to access their entries in $FFqueue$ and $NAqueue$.

Figure 14 shows the operations when a frame arrives. If the virtual start time and the weight of a new frame in F_k are known, variables $V(e_k)$, VS_k, VF_k can be computed based on the frame virtual start time, frame size ($csize_k$), $remainsize_k$, and W_k. We will use the term ***frame starting sequence*** to denote the operations for computing these variables and inserting $(k, V(e_k), \perp)$ to $FFqueue$. Lines (9) to (14) in Figure 14 constitute a frame starting sequence. We will use the term ***frame finishing sequence*** to denote the operations to update the $next\_weight_k$ field when the last packet in a frame is sent. An example of a frame finishing sequence is shown in Line (8) in Figure 14. There are three cases in $Do\_frame\_arrival()$. First, when the busy period just starts, the virtual start time of the frame is 0 and the frame starting sequence is carried out. W_k^T and $Wsum_T$ are adjusted accordingly. Second, when the frame F_k is a back-to-back frame (the last packet is from F_k), the virtual starting time of the next frame is the virtual finishing time of the last frame. In this case, the frame finishing sequence for the old frame is carried out before the frame starting sequence for the new frame. Last, when a new frame arrives, a new entry $(k, atime = current\_time())$ is added to the end of the $NAqueue_k$. Note that the timestamps for the first packet in the class are $BigNUM$ in this case and packets in F_k are ineligible for scheduling at this time. This class will become eligible when $T = atime$ in the $track\_virtual\_time()$ routine. Since $Do\_frame\_arrival()$ is called before $track\_virtual\_time()$ in the FRR scheduling module, when the arrival time of a new frame is always larger than the current T, which was updated in the previous invocation of the FRR scheduling module.

$DW^2F^2Q()$, shown in Figure 15, has three tasks: (1) bookkeeping after each packet is sent ($Do\_packetsent()$); (2) updating $T/V(T)$ and performing the related operations ($track\_virtual\_time()$); and (3) selecting the class based on the timestamps of the first packets in the classes.

There are two cases in $Do\_packetsent()$. First, when the packet sent is not the last packet in a frame, then the timestamps, VS_k and VF_k, for the head of line packet in this class are updated. Lines (3) and (4) imply the logic to assign the virtual finishing time of the frame as the timestamp of the last packet of the frame. Second, the packet sent is the last packet in a frame. In this case, when the class has a back-to-back frame,

```
$DW^2F^2Q()$
(1) Let $lpkt$ be the last packet sent. Let lpkt be from $F_k$;
(2) $Do\_packetsent(lpkt, F_k)$;
(3) track_virtual_time();
(4) Find a class $F_k$ whose $VS_k \leq T$ and $VF_k$ is the smallest
(5) return $F_k$.
**Do_packetsent($lpkt$, $F_k$)**
( 1) **if** ($lpkt$ is not the last packet in the frame) **then**
( 2)     $VS_k = VF_k$;
( 3)     $pktsize = min(csize_k, size(first(F_k)))$;
( 4)     $VF_k = VS_k + pktsize/W_k$;
( 5) **else** /* lpkt is the last packet in a frame */
( 6)     **if** ($F_k$ is not backlogged) **then**
( 7)         **if** ($remainsize_k = 0$) **then**
( 8)             change $(k, V(e_k), \perp)$ to $(k, V(e_k), 0)$
( 9)         **else**
(10)             $W_k = 1/C^k$;
(11)             change $(k, V(e_k), \perp)$ to $(k, V(e_k), W_k)$
(12)             $V(e_k) = V(e_k) + remainsize_k/W_k$;
(13)             insert $(k, V(e_k), 0)$ to $FFqueue$;
(14)         **end if**
(15)     **end if**
(16) **end if**
**track_virtual_time()**
( 1) $tracking = 1$;
( 2) **while** ($tracking = 1$) **do**
( 3)     Let $(k, V(e_k), X)$ be the entry with the smallest $V(e_k)$ in $FFqueue$;
( 4)     Let $(l, atime)$ be the first entry in $NAqueue$;
( 5)     $vctime = V(T) + (current\_time() - T)/Wsum_T$;
( 6)     $vatime = V(T) + (atime - T)/Wsum_T$;
( 7)     Let $earliestv$ be the smallest among $vctime$, $vatime$, and $V(e_k)$.
( 8)     $T = (earliestv - T) * Wsum_T + T$;
( 9)     $V(T) = earliestv$;
(10)     **if** ($earliestv = vatime$) **then**
(11)         remove $(l, atime)$ from $NAqueue$;
(12)         **if** ($earliestv < V(e_l)$) **then**
(13)             $vstime = V(e_l)$;
(14)             change $(l, V(e_l), X)$ to $(l, V(e_l), W_l)$ ;
(15)         **else** $vstime = earliestv$;
(16)         $VS_l = vstime$;
(17)         $VF_l = VS_l + size(first(F_l))/W_l$;
(18)         $V(e_l) = vstime + csize_l/W_l$;
(19)         insert $(l, V(e_l), \perp)$ to $FFqueue$;
(20)         $Wsum_T = Wsum_T + W_l$;
(21)         $W_l^T = W_l$;
(22)     **end if**
(23)     **if** ($earliestv = V(e_k)$) **then**
(24)         **if** ($X \neq \perp$) **then**
(25)             $Wsum_T = Wsum_T + X - W_k^T$;
(26)             $W_k^T = X$;
(27)             remove $(k, V(e_k), X)$ from $FFqueue$;
(28)         **else** $tracking = 0$;
(29)     **end if**
(30)     **if** ($earliestv = vctime$) $tracking = 0$;
(31) **end while**
```

Figure 15: Inter-class scheduling

the case is handled in $Do\_frame\_arrival()$. Otherwise, the frame finishing sequence must be carried out. In addition, if $remainsize_k \neq 0$, there is a new frame (whose data is the partial packet) with no physical packets to schedule. This frame needs to be taken into consideration when tracking the virtual time. A degenerated frame starting sequence (lines (10) to (13)) is carried out.

In $track\_virtual\_time()$, the system tracks the virtual time, trying to bring T up-to-date. This routine tracks every frame arrival (from $NAqueue$) and departure (from $FFqueue$) event from T to the current time. The tracking stops either when $T = current\ time$ or when $V(T)$ is a virtual finishing time of a frame F_k and $next\_weight_k = \perp$. Lines (3) to (7) finds the first event after T, which happens at virtual time $earliestv$. Since there is no other event between $V(T)$ and $earliestv$ and $Wsum_T$ is known, T is updated to the corresponding real time of $earliestv$ (line (8)) and $V(T) = earliestv$. Note that $earliestv$ may be equal to old $V(T)$ when there is a class with a finishing time $V(T)$ and an unknown $next\_weight_k$. In this case, T can only progress after the unknown $next\_weight_k$ is changed to a known weight when the last packet in that frame is sent. If the first event is a frame arrival ($earlistv = vatime$), the corresponding entry is removed from $NAqueue$ and the frame virtual starting time is decided. The last frame for this class F_l may not finish under GPS at $vatime$. Thus, the virtual starting time of this frame is $max(vatime, V(e_l))$. In this case, the $next\_weight_k$ field of the entry $(l, V(e_l), X)$ must be updated. After that, the frame starting sequence is performed. When $earlistv = V(e_k)$, the first event is a frame departure. In this case, if the weight of the next frame in F_k is unknown, the tracking stops. Otherwise, the corresponding entry in $FFqueue$ is removed; $Wsum_T$ is adjusted; and the tracking continues. The tracking also stops when $earliestv = V(current\ time)$. In all the events, $Wsum_T$ and W_k^T are properly maintained.

The size of $FFqueue$ is at most $2n$ and the size of $FAqueue$ is at most n. Assuming that the priority queue is used to implement $FFqueue$, the complexity of the insert and remove operations is $O(lg(n))$ in the worst case. Among the routines in $DW^2F^2Q()$, $track\_virtual\_time()$ has the highest complexity. In the worst case, $O(n)$ ($size(FFqueue) + size(NAqueue)$) frame arrival and departure events must be processed in one invocation of the FRR scheduling module. The worst case complexity for tracking the virtual time is $O(nlg(n))$. Similarly, the worst case time for all $Do\_frame\_arrival()$ calls in one invocation of the FRR scheduling module is $O(nlg(n))$ since n insertions to $FFqueue$ may need to be performed. Hence, the worst case per packet scheduling complexity in FRR is $O(nlg(n)) = O(1)$.