

# Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay

Sriram Ramabhadran  
Department of Computer Science & Engineering  
University of California, San Diego  
9500 Gilman Drive  
La Jolla, CA 92093-0114  
sriram@cs.ucsd.edu

Joseph Pasquale  
Department of Computer Science & Engineering  
University of California, San Diego  
9500 Gilman Drive  
La Jolla, CA 92093-0114  
pasquale@cs.ucsd.edu

## ABSTRACT

Fair queuing is a well-studied problem in modern computer networks. However, there remains a gap between scheduling algorithms that have provably good performance, and those that are feasible and practical to implement in high-speed routers. In this paper, we propose a novel packet scheduler called Stratified Round Robin, which has low complexity, and is amenable to a simple hardware implementation. Stratified Round Robin exhibits good fairness and delay properties that are demonstrated through both analytical results and simulations. In particular, it provides a single packet delay bound that is independent of the number of flows. This property is unique to Stratified Round Robin among all other schedulers of comparable complexity.

## Categories and Subject Descriptors

C.2.6 [Computer Communication Networks]: Internetworking—Routers; F.2.2 [Analysis of Algorithms and Problem Complexity]: Non-numerical Algorithms and Problems—Sequencing and scheduling

## General Terms

Algorithms, Performance

## Keywords

output scheduling, quality of service, fair queuing

## 1. INTRODUCTION

There are an increasing number of interactive Internet-based applications, such as video and audio conferencing, that make quality-of-service (QoS) demands on the network. These applications generally transmit flows of related packets between two end-points, and require that these flows meet certain QoS requirements, in terms of throughput and

end-to-end delay. How to satisfy the QoS requirements of competing flows, given a packet-switched network with decentralized control distributed over all the routers, is a long-standing problem that has been the subject of considerable research [3] [7] [8] [9] [11] [14] [15] [16] [32] [34] in the networking community.

An important component of the many QoS architectures proposed is the packet scheduling algorithm used by routers in the network. The packet scheduler determines the order in which packets of various independent flows are forwarded on a shared output link. One of the simplest algorithms is First Come First Served (FCFS), in which the order of arrival of packets also determines the order in which they are forwarded over the output link. While almost trivial to implement, FCFS clearly cannot enforce QoS guarantees, as it allows rogue flows to capture an arbitrary fraction of the output bandwidth. In general, a packet scheduler should have the following properties.

**Fairness** The packet scheduler must provide some measure of isolation between multiple flows competing for the same shared output link. In particular, each flow should get its fair share of the available bandwidth, and this share should not be affected by the presence and (mis)behavior of other flows. For example, this share may be a pre-allocated amount of bandwidth, that should be available to the flow, regardless of other flow activity.

**Bounded delay** Interactive applications such as video and audio conferencing require the total delay experienced by a packet in the network to be bounded on an end-to-end basis. The packet scheduler decides the order in which packets are sent on the output link, and therefore determines the queuing delay experienced by a packet at each intermediate router in the network.

**Low complexity** With line rates increasing to 40 Gbps, it is critical that all packet processing tasks performed by routers, including output scheduling, be able to operate in nanosecond time frames. The time complexity of choosing the next packet to schedule should be small, and in particular, it is desirable that this complexity be a small constant, independent of the number of flows  $N$ . Equally importantly, the scheduling algorithm should be amenable to an efficient hardware implementation on a router line card.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM '03, August 25-29, 2003, Karlsruhe, Germany.  
Copyright 2003 ACM 1-58113-735-4/03/0008 ...\$5.00.

Designing a packet scheduler with all of these constraints has proved to be a difficult problem. In the long evolution of packet scheduling algorithms, one can identify two basic approaches in design. Timestamp-based (also called deadline-based) algorithms [2] [10] [28] [33] have provably good delay and fairness properties [12] [13] [18] [26] [27], but generally need to sort packet deadlines, and therefore suffer from complexity logarithmic in the number of flows  $N$ . This sorting bottleneck makes practical implementations of these algorithms problematic, and necessitates the design of simpler schemes. Round-robin-based algorithms [24] [23] [6] have  $O(1)$  complexity, and while they support fair allocation of bandwidth, they fail to provide good delay bounds. Thus, while fair queuing is a well-studied problem in modern computer networks, there remains a significant gap between schedulers that have provably good performance, and those that are feasible to implement in high-speed routers.

It is to be emphasized here that reduction in complexity is of paramount importance. If a separate queue is maintained for each flow, the number of queues required is potentially in the millions. Flow aggregation, in the form of Stochastic Fair Queuing [20], can be employed to reduce the number of queues by hashing multiple flows to a single queue. However, in order to limit the effect of a single rogue flow on other flows, the number of queues required is still large. The same is true of many implementations of Differentiated Services [3]. Thus, due to the large number of queues, even logarithmic complexity can be a significant barrier to implementation.

This paper proposes a novel packet scheduling algorithm called Stratified Round Robin. Stratified Round Robin has a complexity of  $O(1)$  in most practical scenarios, and is amenable to an extremely simple hardware implementation. Nevertheless, it has good fairness and delay properties that we demonstrate through both analysis and simulation. In particular, Stratified Round Robin provides a single packet delay bound that is independent of the number of flows. Our contributions are two-fold: (1) we present a practically realizable scheduler that provides a good approximation of Weighted Fair Queuing, and (2) we show that this scheduler, despite its low complexity, exhibits unique asymptotic performance in the form of a constant single packet delay bound.

## 2. PREVIOUS WORK

There is a significant amount of prior work in finding scheduling disciplines that provide delay and fairness guarantees. Generalized Processor Sharing [21] (also called Fluid Fair Queuing) is considered the ideal scheduling discipline that achieves perfect fairness and isolation among competing flows. However, the fluid model assumed by GPS is not amenable to a practical implementation, as network communication takes place in the form of packets that must be transmitted atomically. Nevertheless, in terms of fairness and delay guarantees, GPS acts as a benchmark for other scheduling disciplines. Practical scheduling disciplines can be broadly classified as either timestamp schedulers or round-robin schedulers.

### 2.1 Timestamp schedulers

Timestamp schedulers [2] [11] [10] [17] [33] try to emulate the operation of GPS by computing a timestamp for each packet. Packets are then transmitted in increasing order of

their timestamps. For example, the well-known Weighted Fair Queuing [10] algorithm uses this method by computing the timestamp of a packet as the time it would finish being serviced under a reference GPS server. WFQ exhibits some short-term unfairness which is addressed by the Worst-case Weighted Fair Queuing [2] algorithm. While WFQ schedules the packet with the least timestamp among all packets, WF<sup>2</sup>Q only considers those packets that have started receiving service under the reference GPS server. As a result, WF<sup>2</sup>Q achieves "worst-case fairness", a notion defined in [2]. Although both WFQ and WF<sup>2</sup>Q have good delay bounds and fairness properties, the need to maintain a reference GPS server results in high complexity. Specifically, both algorithms have a time complexity of  $O(N)$ , where  $N$  is the number of competing flows. It has subsequently been shown how to modify WF<sup>2</sup>Q so that it has a time complexity of  $O(\log N)$  [1].

Self-Clocked Fair Queuing [17] and Virtual Clock [33] are timestamp schedulers that use computationally more efficient schemes to compute timestamps without maintaining a reference GPS server. As a result, timestamps can be computed quickly. However, it is still required to sort packets in ascending order of their timestamps. Consequently, they still have a time complexity of  $O(\log N)$  per packet.

In general, although timestamp schedulers have good delay properties, they suffer from a sorting bottleneck that results in a time complexity of  $O(\log N)$  per packet. The Leap Forward Virtual Clock [28] algorithm attempts to address this problem by coarsening the way in which timestamps are computed in Virtual Clock. This results in a reduced time complexity of  $O(\log \log N)$  per packet. This is an interesting result in terms of showing that rough sorting is almost as good as exact sorting. However, the implementation requires a complicated data structure such as a Van Emde Boas tree that typically would have higher constants, and is not suited to a hardware implementation. Thus, the high computational costs associated with timestamp schedulers prevent them from being used in practice.

Recent lower bounds [31] suggest that the  $O(\log N)$  sorting overhead is fundamental to achieving good delay bounds. In particular, any scheduler that has a complexity of below  $O(\log N)$  must incur a GPS-relative delay proportional to  $N$ , the number of flows.

### 2.2 Round-robin schedulers

Round-robin schedulers [24] [6] [19] are the other broad class of work-conserving schedulers. These schedulers typically assign time slots to flows in some sort of round-robin fashion. By eliminating the sorting bottleneck associated with timestamp schedulers, they achieve an  $O(1)$  time packet processing complexity. As a result, they tend to have poor delay bounds and output burstiness.

Deficit Round Robin (DRR) [24] is a well-known example of a round-robin scheme. DRR assigns a quantum size to each flow that is proportional to the weight of the flow. Each flow has a deficit counter that measures the current unused portion of the allocated bandwidth. Packets of backlogged flows are transmitted in rounds, and in each round, each backlogged flow can transmit up to an amount of data equal to the sum of its quantum and deficit counter. The unused portion of this amount is carried over to the next round as the value of the deficit counter. Once a flow is serviced, irrespective of its weight, it must wait for  $N-1$  other flows to be

serviced until it is serviced again. Also, during each round, a flow transmits its entire quantum at once. As a result, DRR has poor delay and burstiness properties. However, due to its extreme simplicity, DRR (or some variant) is the scheduling discipline typically implemented in high speed routers such as the Cisco GSR [30].

Summarizing, timestamp schedulers have good fairness and delay properties but high complexity, while round-robin schedulers are simple to implement but have poor delay bounds and show output burstiness. More recently proposed schemes [6] [19] [5] have attempted to achieve the best of both worlds by combining the fairness and delay properties of timestamp schedulers with the low complexity of round-robin schedulers. This is typically done by evolving a round-robin scheme like DRR and incorporating some elements of a timestamp scheduler. The Stratified Round Robin scheduler proposed in this paper follows this approach.

The Smoothed Round Robin [6] discipline addresses the output burstiness problem of DRR. This is done by spreading the quantum allocated to a flow over an entire round using a Weight Spread Sequence. Although SRR also results in better delay bounds than DRR, the worst case delay experienced by a packet is still proportional to  $N$ , the number of flows.

Aliquem [19] is an evolution of DRR that permits scaling down the quantum assigned to a flow in each round. However, since the quantum may be less than the maximum packet size, a flow may not be able to transmit any data in each round. Therefore a mechanism is required to keep track of the round in which a flow has accumulated enough credit to transmit a packet. Their mechanism, called Active List Management, can be implemented using a priority encoder, similar to Stratified Round Robin. The scaling down of the quanta results in better delay and burstiness properties.

Bin Sort Fair Queuing [5] uses an approximate bin sorting mechanism to schedule packets. Each packet is assigned a deadline similar to a timestamp scheduler. Packets with close deadlines are assigned to the same bin. Within a bin, there is no sorting of packets based on deadlines. Therefore, packets are transmitted in approximately the same order as their deadlines.

Although both Aliquem and BSFQ significantly improve the delay bounds of DRR, it appears that the worst case delay of even a single packet is still proportional to  $N$ , the number of flows. The Stratified Round Robin scheduler proposed in this paper improves upon this by reducing the worst case delay of a single packet to a small constant.

### 3. STRATIFIED ROUND ROBIN

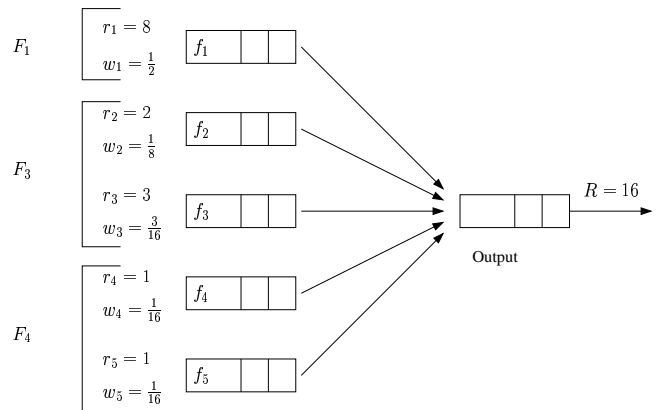
This section describes the main contribution of this paper, which is a new scheduling algorithm called Stratified Round Robin.

#### 3.1 Model

There are  $N$  backlogged flows  $f_1, f_2, \dots, f_N$  that share an output link of bandwidth  $R$ . Flow  $f_i$  has a reserved bandwidth of  $r_i$  with admission control ensuring that

$$\sum_{i=1}^{i=N} r_i \leq R \quad (1)$$

In addition, it is assumed that for every flow  $f_i$ ,  $r_i < R$ . Otherwise scheduling is trivial as there is only one flow which



**Figure 1:** There are five flows  $f_1, \dots, f_5$  sharing an output link of bandwidth  $R = 16$ . Flow  $f_1$  has a reserved bandwidth  $r_1 = 8$  and therefore has a weight  $w_1 = \frac{1}{2}$ , which means that half the total output bandwidth is reserved for  $f_1$ . Weights for the other flows are computed similarly.

is allocated the entire output bandwidth. The *weight*  $w_i$  of flow  $f_i$  is defined as its reserved bandwidth normalized with respect to the total bandwidth of the output link, i.e.,

$$w_i = \frac{r_i}{R} \quad (2)$$

In other words, the weight  $w_i$  of a flow represents the fraction of the output bandwidth that is reserved for flow  $f_i$ . Therefore

$$\sum_{i=1}^{i=N} w_i \leq 1$$

Figure 1 shows an output link being shared by multiple flows and how the weight of each flow is computed. Bandwidth is allocated to flows in proportion to their weights. If all  $N$  flows are backlogged, i.e., have packets queued for transmission, the average bandwidth allocated to  $f_i$  should be

$$\frac{w_i}{\sum_{j=1}^{j=N} w_j} R \geq r_i$$

Therefore the bandwidth allocated to a backlogged flow  $f_i$  is always at least its reserved bandwidth  $r_i$ . Equality occurs when  $\sum_{i=1}^{i=N} w_i = 1$ , i.e., the output bandwidth is completely allocated among the  $N$  flows.

In a fashion similar to round-robin schedulers, Stratified Round Robin assigns slots to flows. When a flow is assigned a slot, it is allowed to send a certain number of its queued packets over the output link. The main scheduling decision is to decide which of the  $N$  flows is assigned the next slot. However, in order to avoid the poor delay properties of round-robin schemes, while simultaneously retaining their low complexity, Stratified Round Robin must do something beyond simply allocating slots in circular fashion. The key idea, which enables getting the best of both worlds, is to aggregate flows into flow classes.

### 3.2 Flow stratification

Flows are “stratified” into *flow classes* based on their weights. More formally, for  $k \geq 1$ , flow class  $F_k$  is defined as

$$F_k = \left\{ f_i : \frac{1}{2^k} \leq w_i < \frac{1}{2^{k-1}} \right\} \quad (3)$$

Flow class  $F_k$  groups together all flows whose weights are approximately  $2^{-k}$ . Figure 1 shows an example of how flows are aggregated into flow classes based on their weights<sup>1</sup>. Flow  $f_1$  has a weight of  $w_1 = \frac{1}{2}$  and belongs to class  $F_1$ . Flows  $f_2$  and  $f_3$  have weights of  $w_2 = \frac{1}{8}$  and  $w_3 = \frac{3}{16}$  respectively and belong to flow class  $F_3$ . Finally, flows  $f_4$  and  $f_5$  have weights  $w_4 = \frac{1}{16}$  and  $w_5 = \frac{1}{16}$  respectively and belong to flow class  $F_4$ . Notice that, by definition, the weights of any two flows  $f_i$  and  $f_j$  belonging to the same class  $F_k$  are within a multiplicative factor of 2 of each other. For each  $k$ , let  $N_k$  denote the cardinality of  $F_k$ . For simplicity, it is assumed that all flows belonging to a flow class are backlogged. When a flow  $f_i$  is not backlogged, it is removed from the corresponding flow class  $F_k$ , and when it becomes backlogged again, it is added to  $F_k$ .

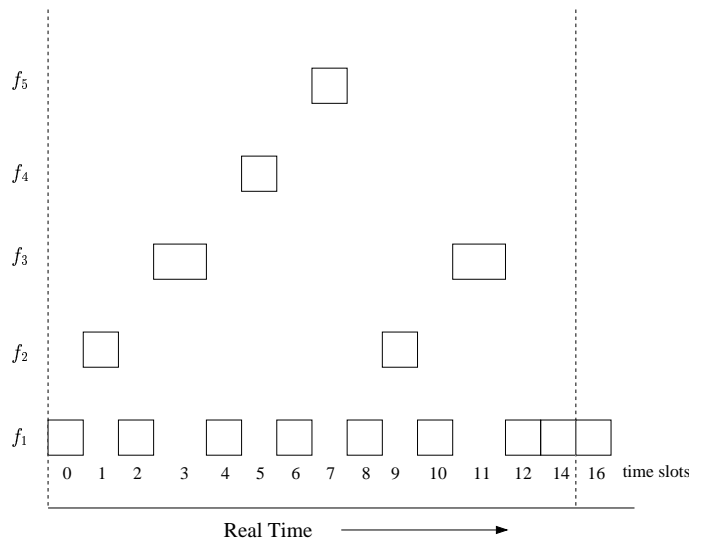
In theory, bandwidth is infinitely divisible, and a flow could reserve an arbitrarily small fraction of the output bandwidth. In practice, bandwidth would be allocated at a certain granularity. Let  $r$  denote the smallest unit of bandwidth that can be allocated to a flow. If a flow has a reserved bandwidth of  $r$ , then it belongs to flow class  $F_n$  where  $\frac{1}{2^n} \leq r < \frac{1}{2^{n-1}}$ . The number of flow classes required to be maintained is therefore  $n = \lceil \log_2 \frac{R}{r} \rceil$ .

Given the stratification of flows into flow classes, Stratified Round Robin may be thought of as a two-step scheduler. The first step (inter-class scheduling) chooses one class from  $n$  flow classes to schedule next. The stratification of flows into classes makes this step easier because rather than considering a potentially large number  $N$  of flows, it needs to consider only a relatively small number  $n$  of classes. The second step (intra-class scheduling) chooses one flow among all flows belonging to a particular flow class to schedule next. This step is easier because all flows belonging to the same class have approximately equal weights. While the first step uses a deadline mechanism similar to many timestamp schedulers, the second step is essentially a round-robin scheme. Therefore, Stratified Round Robin may also be considered a hybrid between the two types of packet schedulers.

### 3.3 Inter-class scheduling

In order to explain the inter-class scheduling algorithm, it is necessary to clarify two concepts: *clock time* and *scheduling interval*. For the purpose of scheduling, time is measured in slots, which are numbered with non-negative integers starting with 0. The clock time, denoted by  $t_C$ , refers to the current slot. Each slot is assigned to at most one flow, during which the flow is serviced and packets belonging to that flow are sent on the output link. At the end of the slot, the clock time is advanced. There may be slots that are not assigned to any flow. Note that slots and clock time are merely accounting mechanisms that help the scheduler in

<sup>1</sup>Similar ideas of using exponential grouping have been developed by others, including [22] [25] in the context of scheduling in ATM networks, and [4] in the context of processor scheduling. However, the underlying scheduling algorithms are different from Stratified Round Robin.



**Figure 2: An example of how Stratified Round Robin allocates slots to flows. Flow weights are as in Figure 1. Scheduling intervals of all classes are aligned in this example.**

deciding which flow to service next. Real time elapses only when packets belonging to a flow are sent on the output link.

Scheduling intervals are fixed length and contiguous intervals of slots that are associated with each flow class. For each class  $F_k$ , the length of a scheduling interval is always  $2^k$  slots. If a scheduling interval for  $F_k$  starts at slot  $t$ , the next scheduling interval for  $F_k$  starts at slot  $t + 2^k$ , and so on. The motivation for defining scheduling intervals in this manner is as follows. Every flow  $f_i \in F_k$  has a weight of approximately  $2^{-k}$ . Therefore, if slots represent fixed-size units of bandwidth allocation,  $f_i$  is entitled to exactly one slot every  $2^k$  slots. In fact, Stratified Round Robin does exactly this by trying to assign every flow  $f_i \in F_k$  one slot in each scheduling interval of  $F_k$ .

A flow class  $F_k$  is called *active* if it contains at least one backlogged flow, i.e.,  $N_k > 0$ . Let  $A$  denote the set of active flow classes. A backlogged flow  $f_i \in F_k$  is called *pending* if  $f_i$  has not been assigned a slot in  $F_k$ 's current scheduling interval. Intuitively, a pending flow is one that has not recently received its fair share of bandwidth. A flow class is called pending if it contains at least one pending flow. Let  $P$  denote the set of pending flow classes.

As mentioned before, the basic strategy is to assign every flow  $f_i \in F_k$  exactly one slot in each scheduling interval of  $F_k$ . The end of the current scheduling interval of a flow class is therefore a deadline for all backlogged flows belonging to that flow class to be assigned a slot. Like a deadline-based scheme, the inter-class scheduler selects the flow class  $F_k$  with the earliest deadline. The intra-class scheduler then assigns a flow  $f_i \in F_k$  the current slot. Note that while choosing a flow class, only those that are pending are considered for assigning slots. A flow class  $F_k$  ceases to be pending when all flows belonging to  $F_k$  have been assigned a slot in its current scheduling interval. When the last pending flow  $f_i \in F_k$  is assigned a slot,  $F_k$  is no longer pending. In this case,  $F_k$  remains like that until the start of its next

scheduling interval, when all flows belonging to  $F_k$  become pending again.

After servicing a flow in the current time slot, the clock time  $t_C$  must be advanced. If there are any pending flow classes,  $t_C$  is incremented by 1. Otherwise,  $t_C$  is advanced to the earliest time when some flow class becomes pending again<sup>2</sup>. In either case, the set of pending flow classes  $P$  must be updated to include all flow classes that become pending at the new clock time  $t_C$ .

Figure 2 illustrates an example of Stratified Round Robin assigning slots to flows in the example of Figure 1. Notice that each flow receives slots in proportion to its approximate weight as given by the flow class it belongs to. For example, flow  $f_1$  receives 8 slots out of 16 because it belongs to class  $F_1$ . Flows  $f_2$  and  $f_3$  each receive 2 slots out of 16 because they belong to the same class  $F_3$  even though their actual weights are different. Also note that slots 13 and 15 are not shown because no packets are sent in these slots and therefore no real time elapses. After slots 12 and 14, no flows classes are pending and therefore the clock advances to the next slot when a flow class becomes pending.

### 3.4 Intra-class scheduling

Within a flow class, flows are scheduled in round-robin fashion. Every time a flow  $f_i$  is assigned a slot, it is given a credit  $c_i$ , which determines how many bytes the flow is allowed to send in that slot. Round-robin schedulers like DRR give each flow a fixed-size credit of  $L_M$  bits where  $L_M$  is the maximum packet size. This ensures that at least one packet will be sent every time a flow is assigned a slot. A deficit counter is used to carry over unused credit to the next slot in the event the flow is backlogged but unable to send a packet due to insufficient credit. This mechanism is necessary to prevent unfair allocation of bandwidth caused by different packet sizes used by different flows. Therefore the number of bytes transmitted by flow  $f_i$  in one slot is at most  $c_i + d_i$ .

Stratified Round Robin operates in a similar manner, but with an important difference. Every time flow  $f_i \in F_k$  is assigned a slot, rather than being given a fixed-size credit, it is given a credit that is proportional to its weight  $w_i$ . Recall that two flows  $f_i$  and  $f_j$  may belong to the same class  $F_k$  even though  $w_i$  may be up to twice as large as  $w_j$ . However both  $f_i$  and  $f_j$  are assigned exactly one slot in every scheduling interval of  $F_k$ . To ensure that the service received by these flows are proportional to their respective weights, the credit given to them every time they are assigned a slot must also be made proportional to their weights<sup>3</sup>. Therefore, a flow  $f_i \in F_k$  is assigned a weight-proportional credit of

$$c_i = 2^k w_i L_M \quad (4)$$

Since

$$\frac{1}{2^k} \leq w_i < \frac{1}{2^{k-1}}$$

the scaling factor of  $2^k L_M$  ensures that for all flows  $f_i$  where

<sup>2</sup>This is similar to LFVC [28] except that the clock accounts for time in the form of slots.

<sup>3</sup>This is similar to using a weighted version of DRR [24] within each class except that weight-proportional credits must be normalized so that they are approximately equal (within a multiplicative factor of 2) across classes.

$1 \leq i \leq N$ ,

$$L_M \leq c_i < 2L_M \quad (5)$$

This is significant for the following two reasons. Firstly, since the credit given is greater than the maximum packet size, at least one packet is guaranteed to be sent every time a flow is assigned a slot. Secondly, it ensures that credits are approximately equal (within a multiplicative factor of 2) across flow classes. Slots are assigned to a flow  $f_i \in F_k$  in proportion to its approximate weight  $2^{-k}$ . Therefore, the credit given to a flow every time it is assigned a slot must be approximately equal to ensure weight-proportional allocation of bandwidth. This variable credit allocation is illustrated in Figure 2 where even though flows  $f_2$  and  $f_3$  are each given 1 slot every 8 slots,  $f_3$  is given a proportionally larger credit (1.5 times) in each slot.

---

#### SCHEDULE

---

```

Choose  $F_k \in P$  s.t.  $F_k$  has the earliest deadline
Choose any pending flow  $f_i \in F_k$ 
 $d_i = d_i + c_i$  where  $c_i = 2^k w_i L_M$ 
while true do
  if  $f_i$  has a packet  $pkt$  backlogged then
    if  $size(pkt) \leq d_i$  then
      dequeue( $pkt$ )
      send( $pkt$ )
       $d_i = d_i - size(pkt)$ 
    end
    else
      if  $F_k$  is not pending then  $P = P - \{F_k\}$ 
      break
    end
  end
  else
    if  $F_k$  is not pending then  $P = P - \{F_k\}$ 
    DEACTIVATE  $f_i$ 
    break
  end
end
if  $P \neq \phi$  then  $t_C = t_C + 1$ 
else
  while  $A = \phi$  do
    Idle when there are no backlogged flows
  end
   $t_C = \min \{t > t_C : \exists k \text{ s.t. } F_k \text{ is pending at } t\}$ 
end
 $P = P \cup \{F_k \in A : F_k \text{ becomes pending at } t_C\}$ 

```

---



---

#### ACTIVATE $f_i$

---

```

 $F_k = F_k \cup \{f_i\}$  where  $\frac{1}{2^k} \leq w_i < \frac{1}{2^{k-1}}$ 
if  $F_k \notin A$  then
   $A = A \cup \{F_k\}$ 
end

```

---



---

#### DEACTIVATE $f_i$

---

```

 $F_k = F_k - \{f_i\}$  where  $\frac{1}{2^k} \leq w_i < \frac{1}{2^{k-1}}$ 
if  $N_k = 0$  then
   $A = A - \{F_k\}$ 
end

```

---

To summarize, Stratified Round Robin operates as a two-step scheduler. The first step uses the flow class mechanism to assign slots to each flow  $f_i$  in proportion to its approximate weight as defined by the flow class  $F_k$  to which it belongs. The second step uses the weight-proportional credit mechanism to ensure that each flow  $f_i$  receives service in proportion to its actual weight  $w_i$ . The advantage of this approach is that it considerably simplifies the scheduling decision to be made. A WFQ-type scheduler must choose among a large number  $N$  of flows to service next, whereas Stratified Round Robin must choose among a small number  $n$  of flow classes to service next. For all practical values of  $n$ , Stratified Round Robin can be implemented in a few bit operations, as described in Section 4.

The remaining operations that must be implemented are activating and deactivating a flow. A flow  $f_i$  is activated when it becomes backlogged due to the arrival of a new packet. Flow  $f_i$  is added to the appropriate class  $F_k$  and if necessary,  $F_k$  is added to the set of active flow classes  $A$ . Note that when  $f_i$  becomes backlogged, it is not pending until the start of  $F_k$ 's next scheduling interval. A flow  $f_i$  is deactivated when it is no longer backlogged as a result of all its queued packets being sent over the output link. Flow  $f_i$  is removed from the appropriate class  $F_k$  and if necessary,  $F_k$  is removed from the set of active flow classes  $A$ . Pseudocode for scheduler operations is provided above.

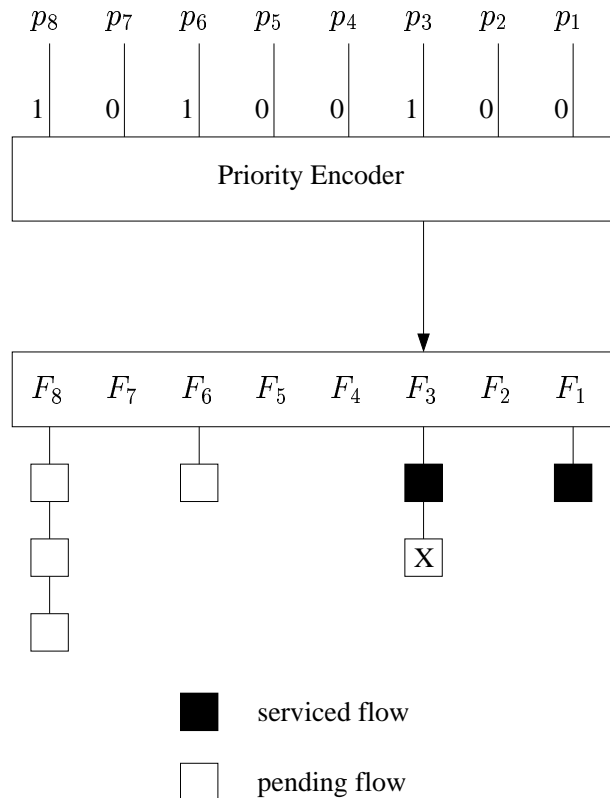
#### 4. IMPLEMENTATION

This section describes how the Stratified Round Robin scheduler can be implemented in  $O(1)$  complexity with modest hardware support.

To realize Stratified Round Robin, both the inter-class and intra-class scheduling operations must be implemented. Scheduling within a flow class is essentially round-robin and therefore almost trivial. For each flow class  $F_k$ , it is necessary to maintain a list of all flows belonging to  $F_k$ . Since the number of flow classes  $n$  is small, this does not significantly add to the complexity of maintaining a single list of flows that an  $O(1)$  scheme like DRR would have to do. Scheduling among flow classes is more involved and must be implemented efficiently.

Inter-class scheduling in Stratified Round Robin is deadline-based, and therefore could suffer from the same sorting bottleneck faced by schedulers such as WFQ. The critical difference is that while WFQ-type schedulers much choose the earliest deadline among  $N$  flows, Stratified Round Robin must choose the earliest deadline among only  $n$  flow classes. The number  $n$  of flow classes is given by  $\log_2 \frac{R}{r}$ , where  $R$  is the bandwidth of the output link, and  $r$  is the minimum unit of bandwidth allocation. For practical values of  $R$  and  $r$ , the number  $n$  is a small number. For example, for an OC-768 link,  $R = 40$  Gbps. Even if  $r = 1$  bps, i.e., flows are allocated bandwidth in units of 1 bps, the number of flow classes  $n$  is at most 36. It is entirely feasible to implement in hardware a priority queue of 36 elements that runs in  $O(1)$  complexity<sup>4</sup>.

<sup>4</sup>The notion of asymptotic complexity arises when  $r \rightarrow 0$ . In this case, the complexity of implementing Stratified Round Robin is the complexity of doing priority queue operations over  $n$  elements, i.e.,  $O(\log n) = O(\log \log \frac{R}{r})$ . With alignment of scheduling intervals, it further reduces to priority

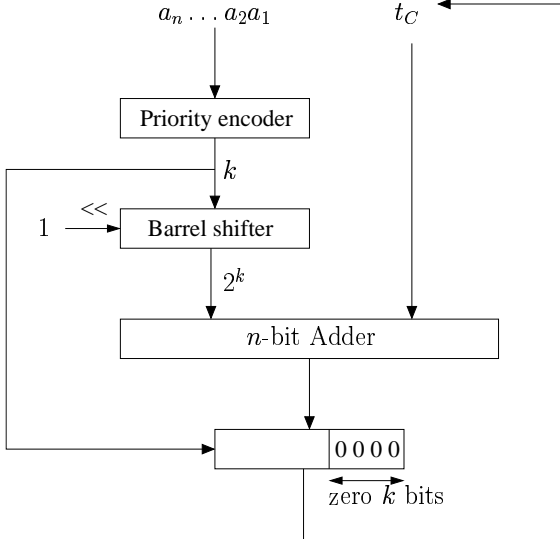


**Figure 3: A priority encoder is used to select the lowest-numbered pending flow class. Within the class, a pending flow is scheduled, as indicated by the cross.**

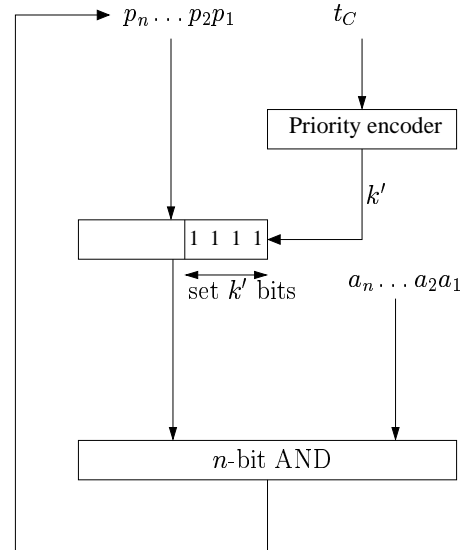
While using a priority queue for small values of  $n$  is feasible, it is possible to exploit a degree of freedom that reduces the complexity of implementation even further. Recall that, for each flow class  $F_k$ , a scheduling interval is a contiguous interval of  $2^k$ . Scheduling intervals for each class are aligned, i.e., scheduling intervals for  $F_k$  always start at clock times corresponding to multiples of  $2^k$ . For example, the first scheduling interval for  $F_k$  is the range of slots  $0 \dots 2^k - 1$ , the second scheduling interval for  $F_k$  is the range of slots  $2^k \dots 2 * 2^k - 1$ , and so on. Note that this is not essential for implementing Stratified Robin. For example, one could imagine a new scheduling interval for  $F_k$  starting at time  $t$  when it firsts becomes active, with the next interval starting at  $t + 2^k$ , and so on. In this case, scheduling intervals for different flow classes would cross each other, and a priority queue would be required to keep track of deadlines. It turns out that aligning these intervals with each other admits a much simpler implementation by dispensing with the priority queue.

Aligning the scheduling intervals for flow class  $F_k$  along multiples of  $2^k$  has the following advantage. The start (end) of a scheduling interval for  $F_k$  is also the start (end) of a scheduling interval for every flow class  $F_{k'}$  where  $k' < k$ . For example, the start of a scheduling interval for  $F_4$  must also be the start of a scheduling interval for each of  $F_1, F_2$

queue operations over the set  $\{1, 2, \dots, n\}$ , which can be done in  $O(\log \log n)$  using a finite universe priority queue.



(a) Advancing the clock



(b) Updating the pending set

Figure 4: A simple hardware implementation of Stratified Round Robin.

and  $F_3$ . This property is what leads to a simple and efficient hardware implementation.

Bitmaps  $a = a_n \dots a_2 a_1$  and  $p = p_n \dots p_2 p_1$  are used to maintain active and pending sets. Bit  $a_k$  is 1 if class  $F_k$  is active, and 0 otherwise. Similarly, bit  $p_k$  is 1 if class  $F_k$  is pending, and 0 otherwise. Since the number of flow classes  $n$  is small, these bitmaps are quite compact, and for reasonable word sizes  $W$ , can be maintained in a single word of memory each. The SCHEDULE operation can then be implemented as follows.

**Choosing a flow class** The inter-class scheduler chooses the pending flow class with the earliest deadline, where the deadline for a class is the start of its next scheduling interval. The alignment property guarantees that a deadline for class  $F_k$  is also a deadline for all classes  $F_{k'}$  where  $k' < k$ . Therefore, the flow class that is selected is simply the lowest-numbered class that is pending. This class can be identified by locating the rightmost bit  $p_k$  of  $p_n \dots p_2 p_1$  that is set to 1. For small values of  $n$ , this can be done in  $O(1)$  by a standard piece of combinational logic called a *priority encoder*. Figure 3 shows an example of a priority encoder being used to choose the flow class to schedule next. Note that  $F_3$  is chosen and not  $F_1$  because  $F_1$  is not pending.

**Advancing the clock** If none of the classes are pending, the clock time  $t_C$  must be advanced to the earliest time at which some class becomes pending. The alignment property guarantees that the start of a scheduling interval for class  $F_k$  is also the start of a scheduling interval for all classes  $F_{k'}$  where  $k' < k$ . Therefore, the earliest time at which some class becomes pending is the start of the next scheduling interval of the lowest-numbered class that is active. This class can be identified by locating the rightmost bit  $a_k$  that is set to 1, again using a priority encoder. The new clock time

is the smallest multiple of  $2^k$  greater than the current clock time  $t_C$ , which can be computed by adding  $2^k$  to  $t_C$  and then setting  $k$  least significant bits to 0, as illustrated in Figure 4.

**Updating the pending set** If  $k$  is the largest number such that  $2^k$  divides the new clock time  $t_C$ , the alignment property guarantees that all active flow classes  $F_{k'}$  where  $k' < k$  become pending at  $t_C$ . To identify  $k$ , it is sufficient to find the least significant bit of  $t_C$  that is set to 1, again using a priority encoder. The  $k$  least significant bits  $p_k \dots p_1$  of  $p$  are set to 1 using an appropriate mask. Finally, since only active classes are considered, it is necessary to perform an AND operation with  $a = a_n \dots a_2 a_1$ , as is illustrated in Figure 4.

The ACTIVATE and DEACTIVATE operations involve nothing more than setting a bit in a bitmap. To conclude, the small number of flow classes in practice allows Stratified Round Robin to be implemented efficiently at low cost. Aligning scheduling intervals along multiples of  $2^k$  further reduces this complexity, resulting in an implementation that requires just a few bit operations more than a  $O(1)$  round-robin scheme like DRR. With commercial implementations of DRR in existence, there is good reason to believe that Stratified Round Robin can be implemented in practice as well.

## 5. ANALYTICAL RESULTS

This section analyzes the properties of Stratified Round Robin. In particular we are interested in the fairness and delay properties of Stratified Round Robin. The analysis in this section assumes that scheduling intervals are aligned with each other as explained. While this is not critical, it does simplify the analysis to a large extent.

LEMMA 1. For  $k \geq 1$ ,

$$\sum_{j=1}^{j=k} N_j 2^{k-j} \leq 2^k \quad (6)$$

where  $N_j$  is the cardinality of  $F_j$ .

PROOF. See appendix  $\square$

The following theorem is critical to the correct operation of Stratified Round Robin.

**THEOREM 1 (CORRECTNESS).** *Let  $f \in F_k$  be a backlogged flow. Stratified Round Robin assigns  $f$  exactly one slot in every scheduling interval of class  $F_k$ .*

PROOF. (by induction on  $k$ ) For  $k = 1$ , Lemma 1 implies  $N_1 \leq 2$ . If  $N_1 = 0$ , there is nothing to prove. Consider any scheduling interval of  $F_1$ , which is of length 2. At the beginning of the interval, all backlogged flows  $f \in F_1$  are pending. Since  $F_1$  is the lowest pending class, all pending flows  $f \in F_1$  are assigned a slot first. Since there are at most two such flows, each backlogged flow  $f \in F_1$  is guaranteed to be assigned a slot. After being assigned a slot, a flow is no longer pending and is not assigned another slot until  $F_1$ 's next scheduling interval.

Assume the inductive hypothesis holds for  $k - 1$  ( $k > 1$ ). We prove that every backlogged flow  $f \in F_k$  is assigned one slot in every scheduling interval of  $F_k$ . If  $N_k = 0$ , there is nothing to prove. Consider any scheduling interval of  $F_k$ , which is of length  $2^k$ . Due to the alignment of scheduling intervals, for every  $j$  ( $1 \leq j < k$ ), this scheduling interval coincides with  $2^{k-j}$  scheduling intervals of  $F_j$ . By the inductive hypothesis, each backlogged flow  $f \in F_j$  is assigned exactly  $2^{k-j}$  slots during this interval. There are at most  $N_j$  such flows. The number of unassigned slots remaining is

$$2^k - \sum_{j=1}^{j=k-1} N_j 2^{k-j} \geq N_k$$

by Lemma 1. Every backlogged flow  $f \in F_k$  is pending until it is assigned a slot during these  $2^k$  slots. Stratified Round Robin assigns a slot to a flow belonging to the lowest pending flow class. Therefore, no flow belonging to a flow class  $F_j$  where  $j > k$  is assigned a slot until every pending flow  $f \in F_k$  is assigned a slot first. Since there are at most  $N_k$  flows in  $F_k$  and there are at least  $N_k$  slots available, every backlogged flow in  $F_k$  is guaranteed to be assigned a slot. After being assigned one slot, a flow is no longer pending and is not assigned another slot until the next scheduling interval of  $F_k$ . Hence proved.  $\square$

It is clear that Stratified Round Robin idles only when there is no packet to send, i.e., it is work conserving. The following sections analyze the fairness and delay properties of Stratified Round Robin.

## 5.1 Golestani fairness

The fairness measure of Golestani [17] essentially requires that the difference between the normalized service received by any two backlogged flows  $f_i$  and  $f_j$ , over any time period  $(t_1, t_2)$ , be bounded by a small constant. This section analyzes Stratified Round Robin with respect to the fairness measure of Golestani.

Before considering Golestani fairness, the following lemma is required. Lemma 2 shows that Stratified Round Robin assigns a flow  $f_i \in F_k$  slots in proportion to its approximate weight  $2^{-k}$ . Specifically, it shows that in any  $x$  contiguous slots, the number of slots assigned to  $f_i$  deviates from its "expected" value of  $2^{-k}x$  by at most 2.

**LEMMA 2.** *Let  $f_i \in F_k$  be a backlogged flow. The number of slots  $x_i$  assigned to  $f_i$  in any time period  $(t_1, t_2)$  is given by*

$$2^{-k}x - 2 < x_i < 2^{-k}x + 2$$

where  $x$  is the number of contiguous slots that completely contain  $(t_1, t_2)$ .

PROOF. See appendix  $\square$

Corollary 1 uses Lemma 2 to bound the amount of service that a flow  $f_i$  can receive in some time period.

**COROLLARY 1.** *Let  $f_i \in F_k$  be a backlogged flow. The amount of service  $S_i(t_1, t_2)$  received by  $f_i$  in any time period  $(t_1, t_2)$  is given by*

$$x w_i L_M - 5L_M < S_i(t_1, t_2) < x w_i L_M + 5L_M$$

where  $x$  is the number of contiguous slot that completely contain  $(t_1, t_2)$ .

PROOF. See appendix.  $\square$

Finally, Theorem 2 gives a bound for Stratified Round Robin with respect to the fairness measure of Golestani.

**THEOREM 2 (GOLESTANI FAIRNESS).** *In any time period  $(t_1, t_2)$  during which flows  $f_i$  and  $f_j$  are backlogged,*

$$\left| \frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j} \right| \leq 5L_M \left( \frac{1}{r_i} + \frac{1}{r_j} \right)$$

PROOF. Applying Corollary 1 to  $f_i$ ,

$$x w_i L_M - 5L_M < S_i(t_1, t_2) < x w_i L_M + 5L_M$$

Dividing by  $r_i$ ,

$$\frac{x w_i L_M}{r_i} - \frac{5L_M}{r_i} < \frac{S_i(t_1, t_2)}{r_i} < \frac{x w_i L_M}{r_i} + \frac{5L_M}{r_i}$$

Since  $\frac{w_i}{r_i} = \frac{1}{R}$ ,

$$\frac{x L_M}{R} - \frac{5L_M}{r_i} < \frac{S_i(t_1, t_2)}{r_i} < \frac{x L_M}{R} + \frac{5L_M}{r_i} \quad (7)$$

Similarly, for  $f_j$ ,

$$\frac{x L_M}{R} - \frac{5L_M}{r_j} < \frac{S_j(t_1, t_2)}{r_j} < \frac{x L_M}{R} + \frac{5L_M}{r_j} \quad (8)$$

Taking the difference of equations 7 and 8 yields

$$\left| \frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j} \right| \leq 5L_M \left( \frac{1}{r_i} + \frac{1}{r_j} \right)$$

$\square$



## 5.2 Bennet-Zhang fairness

The fairness measure of Bennet-Zhang (also called worst-case fairness) is a more refined notion of fairness. Rather than comparing the relative amounts of service received by two flows  $f_i$  and  $f_j$ , it compares the service received by a single flow  $f_i$  to the service it would receive in the ideal case, i.e., when  $f_i$  has exclusive access to an output link of bandwidth  $r_i$ <sup>5</sup>.

Suppose a packet belonging to flow  $f_i$  arrives, creating a total backlog of  $q_i$  in  $f_i$ 's queue. The fairness measure of Bennet-Zhang requires a bound on the maximum time  $\delta_i$  that elapses before the packet is transmitted, thereby "draining" the backlog of  $q_i$ . In particular, it is desired to bound how much  $\delta_i$  is in excess of  $\frac{q_i}{r_i}$ , which is the amount of time it would take to clear a backlog of  $q_i$  with an output link of bandwidth  $r_i$ . Theorem 3 gives a bound for  $\delta_i$ .

THEOREM 3 (BENNET-ZHANG FAIRNESS).

$$\delta_i < \frac{q_i}{r_i} + \frac{5L_M}{r_i} + 5(N-1)\frac{L_M}{R} \quad (9)$$

PROOF. Let  $t_1$  be the time a packet belonging to flow  $f_i$  arrives, creating a total backlog of  $q_i$  at  $t_1$ . Let  $t_2$  be the time the backlog of  $q_i$  is drained. By Theorem 2, the amount of service received by a flow  $f_j$  ( $j \neq i$ ) during  $(t_1, t_2)$  is

$$S_j(t_1, t_2) < r_j \frac{q_i}{r_i} + r_j \frac{5L_M}{r_i} + 5L_M$$

Since real time elapses only when a flow is serviced, the elapsed time is simply the total service received by all flows divided by the bandwidth of the output link  $R$ . Therefore,

$$\begin{aligned} \delta_i &= \frac{q_i}{R} + \frac{1}{R} \sum_{j \neq i} S_j(t_1, t_2) \\ &< \frac{q_i}{R} + \frac{q_i}{r_i} \sum_{j \neq i} \frac{r_j}{R} + \frac{5L_M}{r_i} \sum_{j \neq i} \frac{r_j}{R} + \frac{5L_M}{R} \sum_{j \neq i} 1 \\ &< \frac{q_i}{r_i} \sum_{j=1}^{j=N} \frac{r_j}{R} + \frac{5L_M}{r_i} \sum_{j=1}^{j=N} \frac{r_j}{R} + 5(N-1) \frac{L_M}{R} \\ &\leq \frac{q_i}{r_i} + \frac{5L_M}{r_i} + 5(N-1) \frac{L_M}{R} \end{aligned}$$

since  $\sum_{j=1}^{j=N} r_j \leq R$ .  $\square$

Stratified Round Robin provides a delay bound that is proportional to the number of flows  $N$ . This is not surprising as recent lower bounds [31] suggest that a GPS-relative delay proportional to  $N$  is impossible to avoid without incurring at least  $O(\log N)$  complexity. The following section shows that in the special case of a single packet, Stratified Round Robin achieves a strictly rate-proportional delay bound.

## 5.3 Single packet delay bound

This section analyzes the single packet delay bound for Stratified Round Robin, i.e., the maximum time that elapses between the time it reaches the head of its queue, and the time it is completely transmitted over the output link. This is similar to the fairness measure of Bennet-Zhang, except

<sup>5</sup>This case is identical to the Generalized Processor Sharing model.

that it deals with a backlog of a single packet. A bound is derived by considering the maximum number of slots that can go by before a flow is assigned a slot, and the maximum real time that elapses in any slot.

LEMMA 3. Let  $\tau$  be the maximum real time that elapses in any one slot. Then

$$\tau < \frac{3L_M}{R} \quad (10)$$

PROOF. See appendix.  $\square$

THEOREM 4 (SINGLE PACKET DELAY). For every flow  $f_i$ , let  $\Delta_i$  be the maximum delay experienced by a packet at the head of  $f_i$ 's queue. Then

$$\Delta_i < \frac{12L_M}{r_i} \quad (11)$$

PROOF. Suppose  $f_i \in F_k$ . A packet reaches the head of  $f_i$ 's queue if either  $f_i$  was not backlogged, or the preceding packet was transmitted when  $f_i$  was serviced in the current slot. In either case, this packet is guaranteed to be transmitted in the next slot assigned to  $f_i$ . By Theorem 1,  $f_i$  is assigned a slot in every scheduling interval of  $F_k$ . Therefore the maximum number of slots that can go by before  $f_i$  is assigned a slot again is twice the length of  $F_k$ 's scheduling interval or  $2^{k+1}$  slots. By Lemma 3, the maximum amount of real time that elapses in any slot is  $\frac{3L_M}{R}$ . Therefore

$$\begin{aligned} \Delta_i &\leq 2^{k+1} \tau \\ &< \frac{4R}{r_i} \frac{3L_M}{R} \\ &= \frac{12L_M}{r_i} \end{aligned}$$

$\square$

Theorem 4 gives a strictly rate-proportional upper bound for  $\Delta_i$  that is independent of the number of flows  $N$ . This means that Stratified Round Robin provides a guarantee that a flow  $f_i$  is served within a small constant amount of time that depends only on its reserved bandwidth and is independent of the number of flows  $N$ . To our knowledge, this is the first scheme of this complexity that has this property<sup>6</sup>. Schedulers of comparable complexity such as DRR and SRR can delay even a single packet by an amount that is proportional to  $N$ . An  $O(\log N)$  scheduler like WF<sup>2</sup>Q [2] has an optimal Bennet-Zhang bound of  $\frac{L_M}{r_i}$ . Stratified Round Robin has a bound that is within a multiplicative factor of this for the single packet case.

Theorem 4 implies that Stratified Round Robin has good asymptotic properties, but at the expense of large constant factors (5 for fairness and 12 for delay). It is likely that a tighter analysis could reduce these constants (this is the subject of current work). However, while the large constants are

<sup>6</sup>An interesting application of this is in the area of proportional-share processor scheduling [4] [29], where processor time must be allocated among  $N$  competing processes. Stratified Round Robin not only allocates processor time to each process in proportion to its share, but also provides a bounded response time for each process. A process is guaranteed to be scheduled within a small constant amount of time, independent of the other processes in the system.

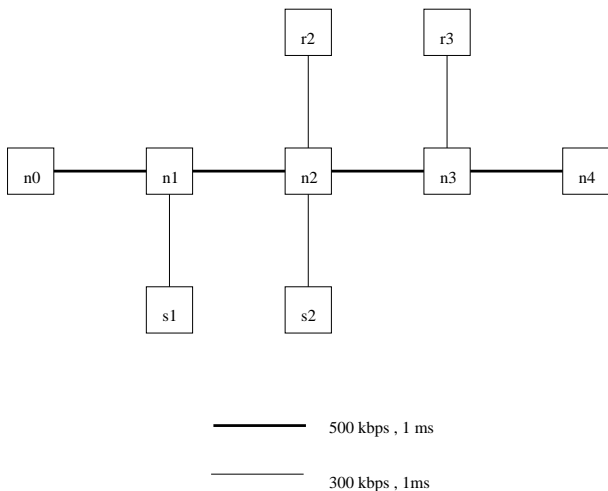


Figure 5: Simulation setup.

not ideal for providing hard QoS guarantees, they are likely to be the key reason for achieving the low complexity of Stratified Round Robin. This tradeoff is important because there are lower bounds on the asymptotic performance of low complexity schedulers [31]. For example, in the case of Stratified Round Robin, the Bennet-Zhang fairness measure is still proportional to  $N$  in the general case of a backlog of packets. A challenge that remains is to design low complexity schedulers that achieve optimal asymptotic properties subject to these lower bounds, while simultaneously having small constants that are important for providing strong QoS guarantees.

## 6. SIMULATION RESULTS

We simulated the operation of Stratified Round Robin using the  $ns-2$  network simulator. The simulation setup is shown in Figure 5. There are 7 CBR flows from  $n_0$  to  $n_4$  with reserved rates ranging from 10 Kbps to 70 Kbps increasing in steps of 10 Kbps. In addition, to congest the intermediate links in the network, there are three rogue CBR flows each between  $n_0$  and  $n_4$ ,  $s_1$  and  $r_2$ , and  $s_2$  and  $r_3$ . Each of these flows transmits at 100 Kbps each. For the purposes of the simulation, we measure the average and worst-case delays of a single control flow between  $n_0$  and  $n_4$ , whose reserved rate is varied from 5 Kbps to 100 Kbps in steps of 5 Kbps. All delays are measured end-to-end.

Stratified Round Robin is compared against two other well-known schedulers: Deficit Round Robin and Weighted Fair Queuing. DRR is chosen because it is a round-robin scheduler with comparable implementation complexity as Stratified Round Robin. WFQ is chosen because it is an example of a time-stamp scheduler with provably good delay properties. Essentially, the purpose of the simulation is to demonstrate that Stratified Round Robin makes it possible to get WFQ-like delay properties with DRR-like implementation complexity.

Figure 6 shows the average and worst-case delay properties of the three schedulers respectively, as a function of reserved bandwidth. As expected, both average and worst-case delays of DRR are relatively insensitive to the reserved bandwidth of the flow. This is because once a flow is ser-

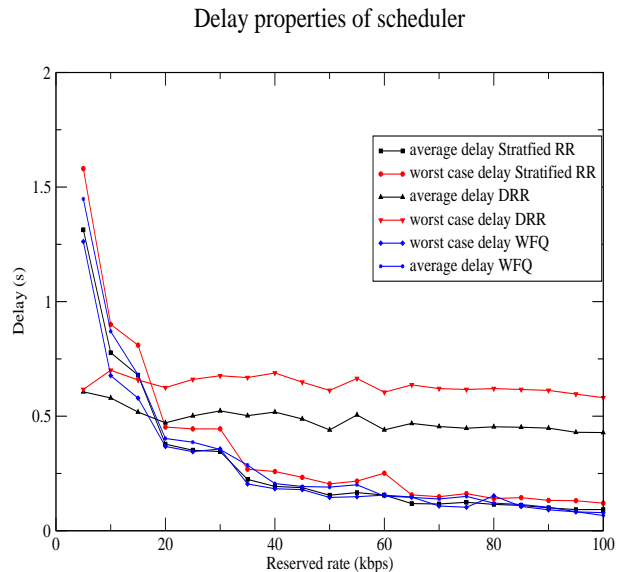


Figure 6: Average and worst-case delays of DRR, WFQ and Stratified Round Robin.

vice, under DRR, it must wait until all other flows are serviced before it is serviced again. In the case of WFQ, the delays are inversely proportional to the reserved rate of the flow, i.e., the higher the reserved rate, the lower the average and worst-case delays. This is the case with Stratified Round Robin as well. It can be seen from Figure 6 that both average and worst-case delays of Stratified Round Robin closely mirror that of WFQ. This is despite the fact that the implementation complexity of Stratified Round Robin is only slightly more than that of DRR, as shown in Section 4.

## 7. CONCLUSIONS

In this paper, we proposed a new fair queuing scheme called Stratified Round Robin. The key idea is to group flows of roughly similar bandwidth requirements into a single flow class. Within a flow class, a weighted round robin scheme with deficit is employed. Since all flows within the class have approximately the same weight, unfairness can be bounded. The flow classes are defined by partitioning the total bandwidth into exponentially increasing intervals. This permits deadline-based scheduling to be efficiently implemented over flow classes, as the number of flow classes grows only logarithmically, and for practical scenarios, is a small constant.

We describe a simple hardware implementation of Stratified Round Robin that can be realized with nothing more sophisticated than a priority encoder. We emphasize ease of implementation as one of the key merits of our scheme. Stratified Round Robin requires only a few bit operations more than a simple scheme like DRR, but provides much stronger fairness and delay guarantees.

Finally, we show that Stratified Round Robin has good fairness and delay properties. Although its worst-case fairness is proportional to  $N$ , for the special case of a single packet, Stratified Round Robin achieves a delay bound that is strictly rate-proportional and independent of the number of flows. This property appears to be unique to Stratified

Round Robin among all schedulers of comparable complexity. Through simulations, we demonstrate that it has good average-case behavior as well, and provides an accurate approximation of Weighted Fair Queuing.

To conclude, Stratified Round Robin is a low complexity scheduler with bandwidth fairness and certain delay bounds. In addition, it admits a simple and efficient hardware implementation. Consequently, we believe that Stratified Round Robin is a good candidate for practical deployment in high-speed routers.

## 8. ACKNOWLEDGMENTS

We gratefully acknowledge the support of AT&T Labs-Research. We also wish to thank the anonymous reviewers, our shepherd Prof. Hui Zhang, and Prof. George Varghese, whose constructive comments led to a much improved paper.

## 9. REFERENCES

- [1] BENNET, J., AND ZHANG, H. Hierarchical packet fair queuing algorithms. In *ACM SIGCOMM '96* (1996).
- [2] BENNET, J., AND ZHANG, H. WF<sup>2</sup>Q : Worst case fair weighted fair queuing. In *IEEE INFOCOM '96* (1996).
- [3] BLAKE, S. ET AL. An architecture for differentiated services. *RFC 2475* (1998).
- [4] CHAN, W., AND NIEH, J. Group ratio round-robin: An O(1) proportional share scheduler. Tech. Rep. CUCS-012-03, Department of Computer Science, Columbia University, April 2003.
- [5] CHEUNG, S., AND PENCEA, C. BSFQ: Bin sort fair queuing. In *IEEE INFOCOM'02* (2002).
- [6] CHUANXIONG, G. SRR, an O(1) time complexity packet scheduler for flows in multi-service packet networks. In *ACM SIGCOMM '01* (2001).
- [7] CLARK, D. The design philosophy of darpa internet protocols. In *ACM SIGCOMM '88* (1998).
- [8] CLARK, D., AND FANG, W. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking* 6 (August 1998).
- [9] CLARK, D., SHENKER, S., AND ZHANG, L. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *ACM SIGCOMM '92* (1992).
- [10] DEMERS, A., KESHAV, S., AND SHENKER, S. Analysis and simulation of a fair queuing algorithm. In *ACM SIGCOMM '89* (1989).
- [11] FIGUEIRA, N., AND PASQUALE, J. Leave-in-time: A new service discipline for real-time communications in a packet-switching network. In *ACM SIGCOMM '95* (1995).
- [12] FIGUEIRA, N., AND PASQUALE, J. An upper bound on the delay for the virtual clock service discipline. *IEEE/ACM Transactions on Networking* 3 (August 1995).
- [13] FIGUEIRA, N., AND PASQUALE, J. A schedulability condition for deadline-based service disciplines. *IEEE/ACM Transactions on Networking* 5 (April 1997).
- [14] FLOYD, S., AND FALL, K. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking* 7 (August 1999).
- [15] FLOYD, S., AND JACOBSON, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1 (August 1993).
- [16] FLOYD, S., AND JACOBSON, V. Link-share and resource management models for packet networks. *IEEE/ACM Transactions on Networking* 3 (August 1995).
- [17] GOLESTANI, S. A self-clocked fair queueing scheme for broadband applications. In *IEEE INFOCOM'94* (1994).
- [18] GOYAL, P., AND VIN, H. Generalized guaranteed rate scheduling algorithms: A framework. *IEEE/ACM Transactions on Networking* 5 (August 1997).
- [19] LENZINI, L., MINGOZZI, E., AND STEA, G. Aliquem: a novel DRR implementation to achieve better latency and fairness at O(1) complexity. In *IWQoS'02* (2002).
- [20] MCKENNEY, P. Stochastic fair queueing. *Internetworking: Research and Experience* 2 (January 1991).
- [21] PAREKH, A., AND GALLAGER, R. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking* 1 (1993).
- [22] REXFORD, J., GREENBERG, A., AND BONOMI, F. Hardware-efficient fair queueing architectures for high-speed networks. In *IEEE INFOCOM '96* (1996).
- [23] SAHA, D., MUKHERJEE, S., AND TRIPATHI, S. Carry-over round robin: A simple cell scheduling mechanism for ATM networks. *IEEE/ACM Transactions on Networking* 6 (1998).
- [24] SHREEDHAR, M., AND VARGHESE, G. Efficient fair queuing using deficit round robin. In *ACM SIGCOMM '95* (1995).
- [25] STEPHENS, D., BENNET, J., AND ZHANG, H. Implementing scheduling algorithms in high speed networks. *IEEE Journal on Selected Areas in Communications: Special Issue on Next-generation IP Switches and Routers* 17 (June 1999).
- [26] STILIADIS, D., AND VARMA, A. Efficient fair queueing algorithms for packet switched networks. *IEEE/ACM Transactions on Networking* 6 (April 1998).
- [27] STILIADIS, D., AND VARMA, A. Rate proportional servers: A design methodology for fair queueing algorithms. *IEEE/ACM Transactions on Networking* 6 (April 1998).
- [28] SURI, S., VARGHESE, G., AND CHANDRANMENON, G. Leap forward virtual clock: An  $O(\log \log N)$  queuing scheme with guaranteed delays and throughput fairness. In *IEEE INFOCOM '97* (1997).
- [29] WALDSPURGER, C. *Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management*. PhD thesis, Massachusetts Institute of Technology, 1995.
- [30] WWW.CISCO.COM. Cisco GSR.
- [31] XU, J., AND LIPTON, R. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. In *ACM SIGCOMM '02* (2002).
- [32] ZHANG, L. *A New Architecture for Packet Switching Network Protocols*. PhD thesis, Massachusetts

Institute of Technology, 1989.

- [33] ZHANG, L. Virtual clock: A new traffic control scheme for packet switching networks. In *ACM SIGCOMM '90* (1990).
- [34] ZHANG, L., DEERING, S., ESTRIN, D., SHENKER, S., AND ZAPPALA, D. RSVP: A new resource reservation protocol. *IEEE Network* (1993).

## APPENDIX

LEMMA 1. For  $k \geq 1$ ,  $\sum_{j=1}^{j=k} N_j 2^{k-j} \leq 2^k$  where  $N_j$  is the cardinality of  $F_j$ .

PROOF. For  $k \geq 1$ ,

$$\begin{aligned} \sum_{j=1}^{j=k} N_j 2^{k-j} &= 2^k \left( \sum_{j=1}^{j=k} \frac{N_j}{2^j} \right) \\ &\leq 2^k \left( \sum_{j=1}^{j=k} \left( \sum_{f_i \in F_j} \frac{r_i}{R} \right) \right) \\ &\leq 2^k \sum_{i=1}^{i=N} \frac{r_i}{R} \\ &= \frac{2^k}{R} \sum_{i=1}^{i=N} r_i \\ &\leq \frac{2^k}{R} R \\ &= 2^k \end{aligned}$$

Hence proved.  $\square$

LEMMA 2. Let  $f_i \in F_k$  be a backlogged flow. The number of slots  $x_i$  assigned to  $f_i$  in any time period  $(t_1, t_2)$  is given by

$$2^{-k}x - 2 < x_i < 2^{-k}x + 2$$

where  $x$  is the minimum number of contiguous slots that completely contain  $(t_1, t_2)$ .

PROOF. If  $f_i$  is allocated exactly  $x_i$  slots, then  $x_i - 2$  is a strict lower bound for the number of scheduling intervals of  $F_k$  contained in the  $x$  slots. Therefore  $x > (x_i - 2)2^k$ , which implies  $x_i < 2^{-k}x + 2$ . Similarly,  $x_i + 2$  is a strict upper bound for the number of scheduling intervals of  $F_k$  contained in the  $x$  slots. Therefore  $x < (x_i + 2)2^k$ , which implies  $x_i > 2^{-k}x - 2$ .  $\square$

COROLLARY 1. Let  $f_i \in F_k$  be a backlogged flow. The amount of service  $S_i(t_1, t_2)$  received by  $f_i$  in any time period  $(t_1, t_2)$  is given by

$$x w_i L_M - 5L_M < S_i(t_1, t_2) < x w_i L_M + 5L_M$$

where  $x$  is the minimum number of contiguous slots that completely contain  $(t_1, t_2)$ .

PROOF. Flow  $f_i$  is given a credit of  $c_i$  for each of the  $x_i$  slots assigned to it during the time period  $(t_1, t_2)$ . It can carry over a deficit of at most  $L_M$  to the next slot.

Therefore,

$$\begin{aligned} S_i(t_1, t_2) &\geq x_i c_i - L_M \\ &> (2^{-k}x - 2)c_i - L_M \\ &= 2^{-k}x c_i - 2c_i - L_M \\ &> 2^{-k}x 2^k w_i L_M - 4L_M - L_M \\ &= x w_i L_M - 5L_M \end{aligned}$$

Similarly,  $f_i$  can carry over a deficit of at most  $L_M$  from the preceding slot. Therefore

$$\begin{aligned} S_i(t_1, t_2) &\leq x_i c_i + L_M \\ &< (2^{-k}x + 2)c_i + L_M \\ &= 2^{-k}x c_i + 2c_i + L_M \\ &< 2^{-k}x 2^k w_i L_M + 4L_M + L_M \\ &= x w_i L_M + 5L_M \end{aligned}$$

$\square$

The reader may be concerned that while computing the bounds in Lemma 2 and Corollary 1, a slightly larger period of  $x$  slots is considered instead of the actual period  $(t_1, t_2)$ . It is easy to verify that both the upper and lower bounds hold for the actual period as well.

LEMMA 3. Let  $\tau$  be the maximum real time that elapses in any slot. Then

$$\tau < \frac{3L_M}{R}$$

PROOF. If a slot is not assigned to any flow, no packets are sent on the output link and so no real time elapses. If a slot is assigned to flow  $f_i \in F_k$ , the amount of service received by  $f_i$  in the slot is at most  $c_i + d_i$ , where  $c_i = 2^k w_i L_M$  is the amount of credit given to  $f_i$  in each slot, and  $d_i$  is the amount of deficit carried over by  $f_i$  from its previous slot. Since  $c_i < 2L_M$  and  $d_i < L_M$ , the amount of service received by  $f_i$  is at most  $3L_M$ . Since  $f_i$  is given the full bandwidth  $R$  of the output link during this slot, the amount of real time that elapses is bounded by  $\frac{3L_M}{R}$ .  $\square$