Efficient Fair Queuing using Deficit Round Robin

M. Shreedhar
Microsoft Corporation*

George Varghese
Washington University in St. Louis.

Abstract

Fair queuing is a technique that allows each flow passing through a network device to have a fair share of network resources. Previous schemes for fair queuing that achieved nearly perfect fairness were expensive to implement: specifically, the work required to process a packet in these schemes was $O(log(n))$, where n is the number of active flows. This is expensive at high speeds. On the other hand, cheaper approximations of fair queuing that have been reported in the literature exhibit unfair behavior. In this paper, we describe a new approximation of fair queuing, that we call *Deficit Round Robin*. Our scheme achieves nearly perfect fairness in terms of throughput, requires only $O(1)$ work to process a packet, and is simple enough to implement in hardware. Deficit Round Robin is also applicable to other scheduling problems where servicing cannot be broken up into smaller units, and to distributed queues.

1 Introduction

When there is contention for resources, it is important for resources to be allocated or scheduled fairly. We need firewalls between contending users, so that the "fair" allocation is followed strictly. For example, in an operating system, CPU scheduling of user processes controls the use of CPU resources by processes, and insulates well-behaved users from ill–behaved users. Unfortunately, in most computer networks there are no such firewalls; most networks are susceptible to badly–behaving sources. A rogue source that sends at an uncontrolled rate can seize a large fraction of the buffers at an intermediate router; this can result in dropped packets for other sources sending at more moderate rates! A solution to this problem is needed to *isolate* the effects of bad behavior to users that are behaving badly.

An isolation mechanism called Fair Queuing [DKS89] has been proposed, and has been proved [GM90] to have nearly perfect isolation and fairness. Unfortunately, Fair Queuing (FQ) appears to be expensive to implement. Specifically, FQ requires $O(log(n))$ work per packet to implement fair queuing, where n is the number of packet streams that are concurrently active at the gateway or router. With a large number of active packet streams, FQ is hard to implement[1] at high speeds. Some attempts have been made to improve the efficiency of FQ; however such attempts either do not avoid the $O(log(n))$ bottleneck or are unfair.

In this paper we shall define an isolation mechanism that achieves nearly perfect fairness (in terms of throughput), and which takes $O(1)$ processing work per packet. Our scheme is simple (and therefore inexpensive) to implement at high speeds at a router or gateway. Further we provide analytical results that do not depend on assumptions about traffic distributions; we do so by providing worst-case results across sequences of inputs. Such *amortized* [CLR90] and *competitive* [ST85] analyses have been a major influence in the analysis of sequential algorithms because they finesse the need to make assumptions about probability distributions of inputs.

Flows: Our intent is to provide firewalls between different packet streams. We formalize the intuitive notion of a packet stream using the more precise notion of a *flow* [Zha91]. A flow has two properties:

- A flow is a stream of packets which traverse the same route from the source to the destination and that re-

*Work done while at Washington University

[1] alternately, while hardware architectures could be devised to implement FQ, this will probably drive up the cost of the router.

quire the same grade of service at each router or gateway in the path.

- In addition, every packet can be uniquely assigned to a flow using prespecified fields in the packet header.

The notion of a flow is quite general and applies to datagram networks (e.g., IP, OSI) and Virtual Circuit networks like X.25 and ATM. For example, a flow could be identified by a Virtual Circuit Identifier (VCI) in a virtual circuit network like X.25 or ATM. On the other hand, in a datagram network, a flow could be identified by packets with the same source–destination addresses.[2] While the source and destination addresses are used for routing, we could discriminate flows at a finer granularity by also using port numbers (which identify the transport layer session) to determine the flow of a packet. For example, this level of discrimination allows a file transfer connection between source A and destination B to receive a larger share of the bandwidth than a virtual terminal connection between A and B.

As in all FQ variants, our solution can be used to provide fair service to the various flows that thread a router, regardless of the way a flow is defined.

Organization: The rest of the paper is organized as follows. In the next section, we review the relevant previous work. A new technique for avoiding the unfairness of round-robin scheduling called *deficit round-robin* is described in Section 3. Round-robin scheduling [Nag87] can be unfair if different flows use different packet sizes; our scheme avoids this problem by keeping state, per flow, that measures the "deficit" or past unfairness. We analyze the behavior of our scheme using both analysis and simulation in Sections 4-6. Basic deficit round-robin provides throughput in terms of fairness but provides no latency bounds. In Section 7, we describe how to augment our scheme to provide latency bounds.

2 Previous Work

Existing Routers: Most routers use first-come-first-serve (FCFS) service on output links. In FCFS, the order of arrival completely determines the allocation of packets to output buffers. The presumption is that congestion control is implemented by the source. In feedback schemes for congestion control, connections are supposed to reduce their sending rate when they sense congestion. However, a rogue flow can keep increasing its share of the bandwidth and cause other (well–behaved) flows to reduce their share. With FCFS queuing, if a rogue connection sends packets at a high rate, it can capture an arbitrary fraction of the outgoing bandwidth. This is what we want to prevent by building firewalls between flows.

Typically routers try to enforce some amount of fairness by giving fair access to traffic coming on different input links. However, this crude form of resource allocation can produce exponentially bad fairness properties as shown below.

In Figure 1 for example, assume that all four flows $F1$ - $F4$ wish to flow through link L to the right of node D, and that all flows always have data to send. If node D does not discriminates flows, node D can only provide fair treatment by alternately serving traffic arriving on its input links. Thus flow $F4$ gets half the bandwidth of link L and all other flows combined get the remaining half. A similar analysis at C shows that $F3$ gets half the bandwidth on the link from C to D. Thus without discriminating flows, $F4$ gets 1/2 the bandwidth of link L, $F3$ gets 1/4 of the bandwidth, $F2$ gets 1/8 of the bandwidth, and $F1$ gets 1/8 of the bandwidth. In other words, the portion allocated to a flow can drop exponentially with the number of hops that the flow must traverse. This is sometimes called the *parking lot* problem because of its similarity to a crowded parking lot with one exit.

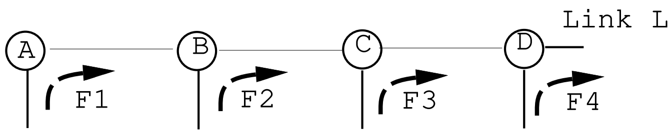

Figure 1: The parking lot problem.

Nagle's solution: In Figure 1, the problem arose because the router allocated bandwidth based on input links. Thus at router D, $F4$ is offered the same bandwidth as flows $F1$, $F2$ and $F3$ combined. It is unfair to allocate bandwidth based on topology. A better idea is to distinguish flows at a router and treat them separately.

Nagle [Nag87] proposed an approximate solution to this problem for datagram networks by having routers discriminate flows, and then providing round-robin service to flows for every output link. Nagle proposed identifying flows using source-destination addresses, and using separate output queues for each flow; the queues are serviced in round-robin fashion. This prevents a source from arbitrarily increasing its share of the bandwidth. When a source sends packets too quickly, it merely increases the length of its own queue. An ill–behaved source's packets will get dropped repeatedly.

[2] Note that a flow might not always traverse the same path in datagram networks, since the routing tables can change during the lifetime of a connection. Since the probability of such an event is low we shall assume that it traverses the same path during a session.

Despite its merits, there is a flaw in this scheme. It ignores packet lengths. The hope is that the average packet size over the duration of a flow is the same for all flows; in this case each flow gets an equal share of the output link bandwidth. However, in the worst case, a flow can get $\frac{Max}{Min}$ times the bandwidth of another flow, where Max is the maximum packet size and Min is the minimum packet size.

Fair Queuing: Demers, Keshav and Shenker devised an ideal algorithm called *bit–by–bit round robin – (BR)* which solves the flaw in Nagle's solution. In the BR scheme, each flow sends one bit at a time in round robin fashion. Since it is impossible to implement such a system, they suggest approximately *simulating* BR. To so, they calculate the time when a packet would have left the router using the BR algorithm. The packet is then inserted into a queue of packets sorted on departure times. Unfortunately, it is expensive to insert into a sorted queue. The best known algorithms for inserting into a sorted queue require $O(log(n))$ time, where n is the number of flows. While the BR guarantees fairness [GM90], the packet processing cost makes it hard to implement cheaply at high speeds.

A naive FQ server would require $O(log(m))$, where m is the number of packets in the router. However Keshav [Kes91] shows that only one entry per flow need be inserted into a sorted queue. This still results in $O(log(n))$ overhead. Keshav's other implementation ideas [Kes91] take at least $O(log(n))$ time in the worst case.

Stochastic Fair Queuing (SFQ): SFQ was proposed by McKenney [McK91] to address the inefficiencies of Nagle's algorithm. McKenney uses hashing to map packets to corresponding queues. Normally, one would use hashing with chaining to map the flow ID in a packet to the corresponding queue. One would also require one queue for every possible flow through the router. McKenney, however, suggests that the number of queues be considerably less than the number of possible flows. All flows that happen to hash into the same bucket are treated equivalently. This simplifies the hash computation (hash computation is now guaranteed to take $O(1)$ time), and allows the use of a smaller number of queues. The disadvantage is that flows that collide with other flows will be treated unfairly. The fairness guarantees are probabilistic; hence the name *stochastic fair queuing*. However, if the size of the hash index is sufficiently larger than the number of active flows through the router, the probability of unfairness will be small. Notice that the number of queues need only be a small multiple of the number of *active flows* (as opposed to the number of *possible* flows, as required by Nagle's scheme).

Queues are serviced in round robin fashion, without considering packet lengths. When there are no free buffers to store a packet, the packet at the end of the longest queue is dropped. McKenney shows how to implement this buffer stealing scheme in $O(1)$ time using bucket sorting techniques. Notice that buffer stealing allows better buffer utilization as buffers are essentially shared by all flows. The major contributions of McKenney's scheme are the buffer stealing algorithm, and the idea of using hashing and ignoring collisions. However, his scheme does nothing about the inherent unfairness of Nagle's round-robin scheme.

Other Relevant Work: Golestani introduced [Gol94] a fair queuing scheme, called self-clocked fair queuing. This scheme uses a virtual time function which makes computation of the departure times simpler than in ordinary Fair Queuing. However, his approach retains the $O(log(n))$ sorting bottleneck.

Van Jacobson and Sally Floyd have proposed a resource allocation scheme called Class-based queuing that has been implemented. In the context of that scheme, and independent of our work, Sally Floyd has proposed a queuing algorithm [Flo93a, Flo93b] that is similar to our Deficit Round Robin scheme described below. Her work does not have our theorems about throughput properties of various flows; however, it does have interesting results on delay bounds and also considers the more general case of multiple priority classes. A recent paper [SA94] has (independently) proposed a similar idea to our scheme: in the context of a specific LAN protocol (DQDB) they propose keeping track of remainders across rounds. Their algorithm is, however, mixed in with a number of other features needed for DQDB. We believe that we have cleanly abstracted the problem; thus our results are simpler and applicable to a variety of contexts.

A paper by Parekh and Gallagher [PG93] showed that fair queuing could be used together with a leaky bucket admission policy to provide delay guarantees. This showed that FQ provides more than isolation; it also provides end–to–end latency bounds. While it increased the attractiveness of FQ, it provided no solution for the high overhead of FQ.

3 Deficit Round Robin

Ordinary round-robin servicing of queues can be done in constant time. The major problem, however, is the unfairness caused by possibly different packet sizes used by different flows. We now show how this flaw can be removed, while still requiring only constant time. Since our scheme is a simple modification of round-robin servicing, we call our scheme *deficit round-robin.*

We use stochastic fair queuing to assign flows to queues. To service the queues, we use round-robin servicing with a quantum of service assigned to each queue; the only difference from traditional round-robin is that *if a queue was not able to send a packet in the previous round because its packet size was too large, the remainder from the previous quantum is added to the quantum for the next round.* Thus deficits are kept track off; queues that were shortchanged in a round are compensated in the next round.

In the next few sections, we will describe and precisely prove the properties of deficit round-robin schemes. We start by defining the figures of merit used to evaluate different schemes. In defining the figures of merit we make two assumptions:

- We use McKenney's idea of stochastic queuing to bound the number of queues required. Thus when combining deficit round-robin with hashing, there are two orthogonal issues which affect performance. To clearly separate these issues, we will assume during the analysis of deficit round-robin that flows are mapped uniquely into different queues. We incorporate the effect of hashing later.
- In calculating throughput, we assume that each flow is always *backlogged* — i.e., always has a packet to send. We return to the issue of fairness for non-backlogged flows in Section 6.

Figures of Merit: Currently, there is no uniform figure of merit defined for Fair Queuing algorithms. We define two measures: *FairnessIndex* (that measures the fairness of the queuing discipline) and *WorkQuotient* (that measures the time complexity of the queuing algorithm). Similar fairness measures have been defined before, but no definition of work has been proposed. It is important to have measures that are not specific to deficit round robin, so that they can be applied to other forms of fair queuing.

To define the work measure, we assume the following model of a router We assume that packets sent by flows arrive to an Enqueue Process that queues a packet to an output link for a router. We assume there is a Dequeue Process at each output link (although the figure shows a single Dequeue Process) that is active whenever there are packets queued for the output link; whenever a packet is transmitted, this process picks the next packet (if any) and begins to transmit it. Thus the work to process a packet involves two parts: enqueuing and dequeuing.

Definition 3.1 *Work is defined as the maximum of the time complexities in enqueuing or dequeuing a packet from the router.*

For example, if a fair queuing algorithm takes $O(log(n))$ time to enqueue a packet and $O(1)$ time to dequeue a packet, we say that the *Work* of the algorithm is $O(log(n))$. To define the throughput fairness measure, we assume a heavy traffic model. Thus all n flows have a continuous stream of arbitrary sized packets arriving to the router, and all these flows wish to leave the router on the same outgoing link. In other words, there is always a backlog for each flow, and the backlog consists of arbitrary sized packets.

Assume that we start sending packets on the outgoing link at time 0. Let $sent_{i,t}$ be the total number of bytes sent by flow i by time t; let $sent_t$ be the total number of bytes sent by all n flows by time t. Intuitively, we will define a fairness quotient for flow i that is the worst-case ratio (across all possible input packet size distributions) of the bytes sent by flow i to the bytes sent by all flows. This merely expresses the worst-case "share" obtained by flow i. While we can define such a quotient after any amount of time t, it is more natural to take the limit as t tends to infinity.

Definition 3.2 $FQ_i = Max(\lim_{t\to\infty} \frac{sent_{i,t}}{sent_t})$, *where the maximum is taken across all possible input packet size distributions for all flows.*

Next, we assume there is some quantity f_i, settable by a manager, which expresses the ideal share to be obtained by flow i. Thus the "ideal" fairness quotient for flow i is $\frac{f_i}{\sum_{j=1}^{n} f_j}$. In the simplest case, all the f_i are equal and the ideal fairness quotient is $1/n$. Finally, we measure how far a fair queuing implementation departs from the ideal by measuring the ratio of actual fairness quotient achieved to the ideal fairness quotient. We call this the fairness index.

Definition 3.3 *The fairness index for a flow i in a fair queuing implementation is:*

$$FairnessIndex_i = \frac{FQ_i \cdot \sum_{j=1}^{n} f_j}{f_i},$$

Algorithm: We propose an algorithm called *Deficit Round Robin* (Figure 2, Figure 3) for servicing queues in a router (or a gateway). We will assume that the quantities f_i, that indicate the share given to flow i, [3] are specified by a quantity called $Quantum_i$ for each flow (for reasons that will be apparent below). Also, since the algorithm works in rounds, we measure time in terms of rounds.

[3] more precisely, this is the share given to queue i and to all flows that hash into this queue. However, we will ignore this distinction until we incorporate the effects of hashing.

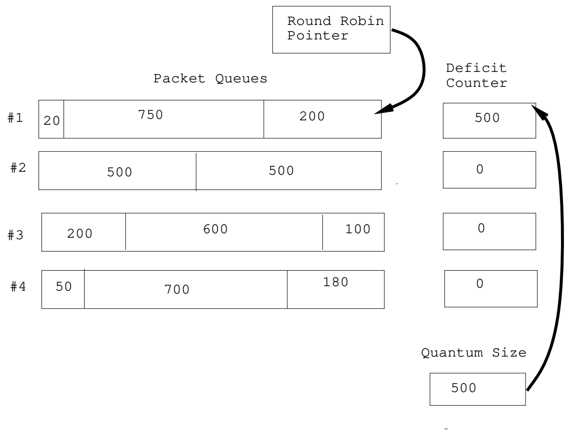

Figure 2: Deficit Round Robin: At the start, all the $DeficitCounter$ variables are initialized to zero. The round robin pointer points to the top of the active list. When the first queue is serviced the $Quantum$ value of 500 is added to the $DeficitCounter$ value. The remainder after servicing the queue is left in the $DeficitCounter$ variable.

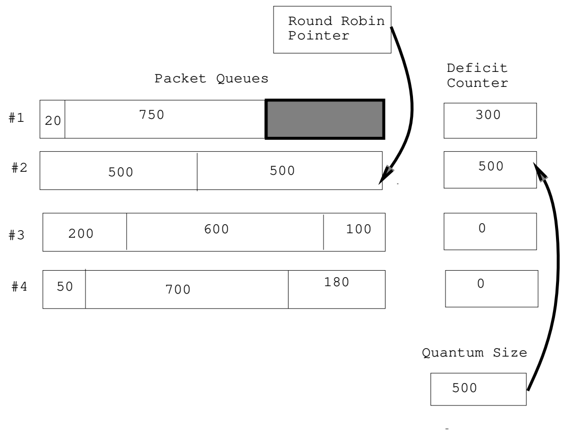

Figure 3: Deficit Round Robin (2): After sending out a packet of size 200, the queue had 300 bytes of its quantum left. It could not use it the current round, since the next packet in the queue is 750 bytes. Therefore, the amount 300 will carry over to the next round when it can send packets of size totaling 300 (deficit from previous round) + 500 (quantum).

Packets coming in on different flows are stored in different queues. Let the number of bytes sent out for queue i in round k be $bytes_{i,k}$. Each queue i is allowed to send out packets in the first round subject to the restriction that $bytes_{i,1} \leq Quantum_i$. If there are no more packets in queue i after the queue has been serviced, a state variable called $DeficitCounter_i$ is reset to 0. Otherwise, the remaining amount $(Quantum_i - bytes_{i,k})$ is stored in the state variable $DeficitCounter_i$. In subsequent rounds, the amount of bandwidth usable by this flow is the sum of $DeficitCounter_i$ of the previous round added to $Quantum_i$. Pseudocode for this algorithm is shown in Figure 4.

To avoid examining empty queues, we keep an auxiliary list *ActiveList* which is a list of indices of queues that contain at least one packet. Whenever a packet arrives to a previously empty queue i, i is added to the end of *ActiveList*. Whenever index i is at the head of *ActiveList*, the algorithm services up to $Quantum_i + DeficitCounter_i$ worth of bytes from queue i; if at the end of this service opportunity, queue i still has packets to send, the index i is moved to the end of *ActiveList*; otherwise, $DeficitCounter_i$ is set to zero and index i is removed from *ActiveList*.

In the simplest case $Quantum_i = Quantum_j$ for all flows i, j. Exactly as in weighted fair queuing, however, each flow i can ask for a larger relative bandwidth allocation and the system manager can convert it into an equivalent value of $Quantum_i$. Clearly if $Quantum_i = 2Quantum_j$, the manager intends that flow i get twice the bandwidth of flow j when both i and j are active.

4 Analytical Results

We begin with a lemma that is true for all executions of the DRR algorithm (not just for the heavy traffic scenario which is used to evaluate fairness):

Lemma 4.1 *For all i and at the end of a round in every execution of the DRR algorithm:* $0 \leq DeficitCounter_i < Max$.

Proof: Initially, $DeficitCounter_i = 0 \Longrightarrow DeficitCounter_i < Quantum_i$. Notice that the $DeficitCounter_i$ variable only changes value when queue i is serviced. During a round, when the servicing of queue i completes there are two possibilities:

- If a packet is left in the queue for flow i, then it must be of size strictly greater than $DeficitCounter_i$. Also, by definition, the size of any packet is no more than Max; thus $DeficitCounter_i$ is strictly less than Max. Also, the code guarantees that $DeficitCounter_i \geq 0$.
- If no packets are left in the queue, the algorithm resets $DeficitCounter_i$ to zero.

□

Next we consider the case where only flow i always has a backlog (i.e., the other flows may or may not be backlogged or even active), and show that the difference between the ideal and actual allocation to flow i is always bounded by the size of a maximum-size packet. While this result will

```
Consider any output link for a given router.
Queue_i is the ith queue, which stores packets
with flow id i. Queues are numbered 0 to (n − 1),
n is the maximum number of output link queues.

Enqueue(), Dequeue() are standard Queue operators.
We use a list of active flows, ActiveList, with
standard operations like InsertActiveList, which adds
a flow index to the end of the active list.
FreeBuffer() frees a buffer from the flow with the
longest queue using using McKenney's buffer stealing.
Quantum_i is the quantum allocated to Queue_i.
DeficitCounter_i contains the bytes that Queue_i did not
use in the previous round.

Initialization:
  for(i = 0; i < n; i = i + 1)
      DeficitCounter_i = 0;

Enqueuing module: on arrival of packet p
  i = ExtractFlow(p)
  if (ExistsInActiveList(i) == FALSE) then
     InsertActiveList(i); (*add i to active list*)
     DeficitCounter_i = 0;
  if no free buffers left then
     FreeBuffer(); (* using buffer stealing *)
  Enqueue(i, p); (* enqueue packet p to queue i*)

Dequeuing module:
  While(TRUE) do
    If ActiveList is not empty then
      Remove head of ActiveList, say flow i
      DeficitCounter_i = Quantum_i + DeficitCounter_i;
      while((DeficitCounter_i > 0) and
            (Queue_i not empty)) do
        PacketSize = Size(Head(Queue_i));
        if(PacketSize ≤ DeficitCounter_i) then
          Send(Dequeue(Queue_i));
          DeficitCounter_i = DeficitCounter_i
             −PacketSize;
        Else break; (*skip while loop *)
      if (Empty(Queue_i)) then
        DeficitCounter_i = 0;
      Else InsertActiveList(i);
```

Figure 4: Code for Deficit Round Robin

imply that the fairness index of a flow is 1, it has even stronger implications. This is because it implies that even during arbitrarily short intervals, the discrepancy between what a flow gets and what it should get is bounded by *Max*.

The router services the queues in a round robin manner according to the DRR algorithm defined earlier. A round is one round-robin iteration over the n queues. We assume that rounds are numbered starting from 1, and the start of Round 1 can be considered the end of a hypothetical Round 0.

Definition 4.1 *A flow is* **backlogged** *in an execution if the queue for flow i is never empty at any point during the execution.*

Theorem 4.2 *Consider any execution of the DRR scheme in which flow i is backlogged. After any K rounds the difference between $K \cdot Quantum_i$ (i.e., the bytes that flow i should have sent) and the bytes that flow i actually sends is bounded by Max.*

Proof: We start with some definitions. Let $DeficitCounter_{i,k}$ be the value of $DeficitCounter_i$ for flow i at the end of round k. Let $bytes_{i,k}$ be the bytes sent by flow i in round k. Let $sent_{i,k}$ be the bytes sent by flow i in rounds 1 through k. Clearly, $sent_{i,K} = \sum_{k=1}^{K} bytes_{i,k}$.

Initially, we have: for all i, $DeficitCounter_{i,0} = bytes_{i,0} = 0$. The main observation (which follows immediately from the protocol) is: $bytes_{i,k} + DeficitCounter_{i,k} = Quantum_i + DeficitCounter_{i,k-1}$. We use the assumption that flow i always has a backlog in the above equation. Thus in round k, the total allocation to flow i is $Quantum_i + DeficitCounter_{i,k-1}$. Thus if flow i sends $bytes_{i,k}$, then the remainder will be stored in $DeficitCounter_{i,k}$, because queue i never empties during the execution. This equation reduces to:

$$bytes_{i,k} = Quantum_i + DeficitCounter_{i,k-1} - DeficitCounter_{i,k}.$$

Summing the last equation over K rounds of servicing of flow i we get a telescoping series. Since $sent_{i,K} = \sum_{k=1}^{K} bytes_{i,k}$ and $DeficitCounter_{i,0} = 0$, we get:

$$sent_{i,K} = K \cdot Quantum_i - DeficitCounter_{i,K}.$$

The ideal bytes allocated to a flow i after K rounds is $K \cdot Quantum_i$. Subtracting, it is easy to see that the difference is $\leq Max$ (using Lemma 4.1). □

The following corollary shows that two backlogged flows that have the same quantum receive almost the same service. In Section 6 we show that a similar result holds even if the flows are not initialized in the same way, and if the two flows are compared in the middle of a round. The corollary follows directly from Theorem 4.2.

Corollary 4.3 *The maximum difference in the total bytes sent by any two backlogged flows (that have the same quantum size) after any number of rounds is less than Max.*

Finally, we consider the heavy traffic scenario in which there are n continuously active incoming flows. Recall that we assume that there are packets available to be sent out from all flow queues at all times. The packets are of arbitrary sizes in the range between Min and Max. Using this scenario, we show that the fairness index of all flows is 1.

Theorem 4.4 *For any flow i, $FairnessIndex_i = 1$.*

Proof: We know from Theorem 4.2 that the maximum difference between a flow's ideal and actual allocations is Max. Thus for any flow i and any round K:

$$K \cdot Quantum_i - Max \leq sent_{i,K} \leq K \cdot Quantum_i$$

Summing over all n flows we get:

$$K \sum_{i=1}^{n} Quantum_i - n \cdot Max \leq sent_K \leq K \sum_{i=1}^{n} Quantum_i.$$

Thus:

$$\frac{K \cdot Quantum_i - Max}{K \sum_{i=1}^{n} Quantum_i} \leq \frac{sent_{i,K}}{sent_K} \leq \frac{K \cdot Quantum_i}{K \sum_{i=1}^{n} Quantum_i - n \cdot Max}$$

Now in the limit as time t tends to infinity, so does the number of rounds K. If we take the limit of the previous equation as K goes to infinity we get (see definition of FQ given in Definition 3.2):

$$\frac{Quantum_i}{\sum_{i=1}^{n} Quantum_i} \leq FQ_i \leq \frac{Quantum_i}{\sum_{i=1}^{n} Quantum_i},$$ which implies that:

$$FQ_i = \frac{Quantum_i}{\sum_{i=1}^{n} Quantum_i}.$$

Finally, since $f_i = Quantum_i$ (recall that in DRR the share allocated to each flow is expressed using the quantum allocated to each flow), we get:

$$FairnessIndex_i = \frac{FQ_i \cdot \sum_{j=1}^{n} f_j}{f_i} = 1. \qquad \square$$

Having dealt with the fairness properties of DRR, we analyze the worst-case packet processing work. It is easy to see that the size of the various *Quantum* variables in the algorithm determines the number of packets that can be serviced from a queue in a round. This means that the latency for a packet (at low loads) and the throughput of the router (at high loads) is dependent on the value of the *Quantum* variables.

Theorem 4.5 *The Work for Deficit Round Robin is O(1), if for all i, $Quantum_i \geq Max$.*

Proof: Enqueuing a packet requires finding the queue used by the flow ($O(1)$ time complexity using hashing since we ignore collisions), appending the packet to the tail of the queue, and possibly stealing a buffer ($O(1)$ time using the technique in [McK91]). Dequeuing a packet requires determining the next queue to service by examining the head of *ActiveList*, and then doing a constant number of operations (per packet sent from the queue) in order to update the deficit counter and *ActiveList*. If $Quantum \geq Max$, we are guaranteed to send at least one packet every time we visit a queue, and thus the worst-case time complexity is $O(1)$. $\square$

In the previous analysis, we showed that if we ignored collisions introduced by hashing, Deficit Round Robin (DRR) can achieve a *FairnessIndex* that is close to 1 and a $Work = O(1)$. If we combine DRR with hashing, *FairnessIndex* must be adjusted by the average number of collisions. The average number of other flows that collide with a flow is $\frac{n}{Q}$[CLR90], where n is the number of flows and Q is the number of queues. For example, if we have 1000 concurrent flows and 10,000 queues (a factor of 10, which is achievable with modest amounts of memory) the average number of collisions is 0.1. If B is the bandwidth allocated to a flow, the effective bandwidth in such a situation becomes $\frac{B}{1+\frac{n}{Q}}$. Thus assuming a suitable choice of quantum size and a reasonable number of queues, our overall scheme achieves a *FairnessIndex* that is close to 1 and a $Work = O(1)$.

We compare the *FairnessIndex* and *Work* of the fair queuing algorithms that have been proposed until now in Figure 1. Deficit Round Robin is the only algorithm that provides a *FairnessIndex* of 1 and O(1) *Work*.

5 Simulation Results

We wish to answer the following questions about the performance of DRR.

- We would like to experimentally confirm that DRR provides isolation superior to FCFS as the theory indicates, especially in the backlogged case.
- The theoretical analysis of DRR is for a single router (i.e., 1 hop). How are the results affected in a multiple hop network?
- The theoretical analysis of DRR showed fairness under the assumption that all flows were backlogged. Is the fairness provided by DRR still good when the flows arrive at different (not backlogged) rates and with different distributions? Is the fairness sensitive to packet size distributions and arrival distributions?

Table 1: Performance of Fair Queuing Algorithms

Fair Queuing Algorithm	**Fairness Index**	**Work Complexity**
Fair Queuing ([Nag87])	$\frac{Max}{Min}$	O(1) Expected
Fair Queuing ([DKS89])	1	O(log(n))
Stochastic Fair Queuing	$\frac{Max}{Min}$	O(1)
Deficit Round Robin	1 (Expected)	O(1)

Since we have multiple effects, we have devised experiments to isolate each of these effects. However, there are other parameters (such as number of packet buffers and Flow Index size) that also impact performance. We first did parametric experiments to determine these values before investigating our main questions. For lack of space we only present a few experiments and refer the reader to a forthcoming report for more details.

5.1 Default Simulation Settings

Unless otherwise specified, the default for all the later experiments is as specified here. We measure the throughput in terms of delivered bits in a simulation interval, typically 2000 seconds.[4]

In the single router case (see Figure 5), there are one or more hosts. Each host has twenty flows, each of which generates packets at a Poisson average rate of 10 packets/second. The packet sizes are randomly selected between 0 and Max packet size (which is 4500 bits). Ill–behaved flows send packets at thrice the rate at which the other flows send packets (i.e., 30 packets/second). Each host in such an experiment is configured to have one ill–behaved flow.

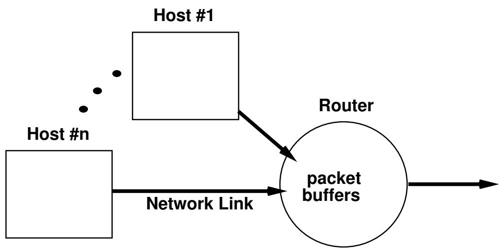

Figure 5: Single router configuration.

In Figure 6, we show the typical settings in a multiple hop topology. There are hosts connected at each hop (a router) and each host behaves as as described in the previous section. In multiple hop routes, where there are more than twenty flows through a router, we use large buffer sizes (around 500 packet buffers) to factor out any effects due to lack of buffers. In multiple hop topologies, all outgoing links are set at 1Mbps.

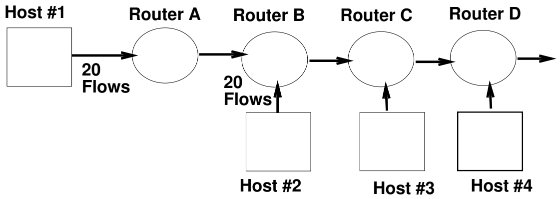

Figure 6: Multiple router configuration.

5.2 Comparison of DRR and FCFS

To show how DRR performs with respect to FCFS, we performed the following two experiments. In Figure 7, we use a single router configuration and one host with twenty flows sending packets at the default rate through the router. The only exception is that Flow 10 is a misbehaving flow. We use Poisson packet arrivals and random packet sizes. All parameters used are the default settings. The figures shows the bandwidth offered to different flows using the FCFS and DRR queuing disciplines. We find that in FCFS the ill–behaved flow grabs an arbitrary share of bandwidth, while in DRR there is nearly perfect fairness.

We further contrast FCFS scheduling with DRR scheduling by examining the throughput offered to each flow at different stages in a multiple hop topology. The experimental setup is the default multiple hop topology described in Section 5.1. The throughput offered is measured at Router D (see Figure 8). This time we have a number of misbehaving flows. The figure shows that DRR behaves well in multihop topologies.

[4]Throughput is typically measured in bits/second. However, it is easy to convert our results into bits/second by dividing by the simulation time.

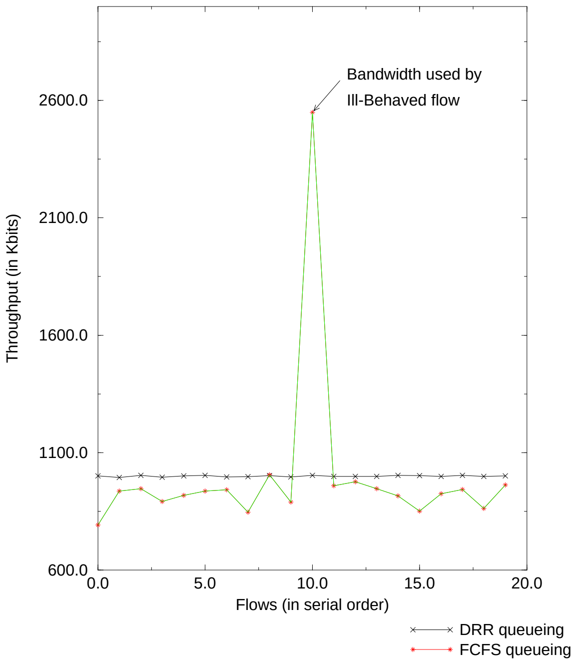

Figure 7: This is a plot of the bandwidth offered to flows using FCFS queuing and DRR. In FCFS, the ill–behaved flow (flow 10) obtains an arbitrary share of the bandwidth. The isolation property of DRR is clearly illustrated.

5.3 Independence with respect to Packet Sizes

We investigate the effect of different packet size on the fairness properties. The packet sizes in a train of packets can be modeled as random, constant or bimodal. We use a single router configuration with one host that has 20 flows. In the first experiment, we use random packet sizes. In the next two experiments, instead of using the random packet sizes, we first use a constant packet size of 100 bits, and then a bimodal size varying between 100 and 4500 bits.

Figure 9 shows the lack of any particular pattern in response to the usage of different packet sizes in the packet traffic into the router. The difference in bandwidth offered to a flow while using the three different packet size distributions is negligible. The maximum deviation from this figure while using constant, random and bi-modal cases turned out to be 0.3%, 0.4699% and 0.32% respectively. Thus DRR seems fairly insensitive to packet size distributions.

This property becomes clearer when the DRR algorithm is contrasted with the performance of the SFQ Algorithm. While using the SFQ algorithm (simulation results not shown here), flows sending larger packets consistently get higher throughput than the flows sending random sized packets; while all flows get equal bandwidth while using DRR.

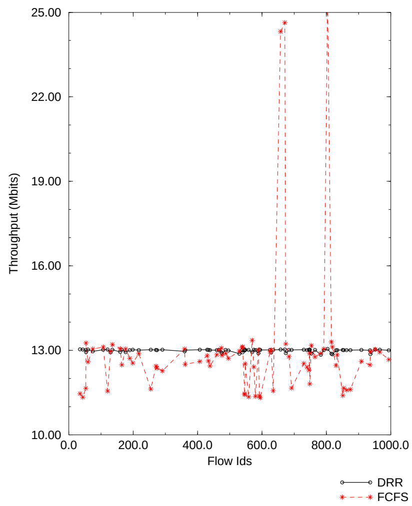

Figure 8: The bandwidth allocated to different flows at Router D using FCFS and DRR to schedule the departure of packets.

5.4 Independence with respect to Traffic Patterns

We show that DRR's performance is independent of the traffic distribution of the packets entering the router. We used two models of traffic generators: exponential and constant interarrival times respectively. and collected data on the number of bytes sent by each of the flows. We then examined the results in the bandwidth obtained. The other parameters (e.g. number of buffers in the router) are kept constant at sufficiently high values in this simulation.

The experiment used a single router configuration with default settings. The outgoing link bandwidth was set to 10Kbps. Therefore, if there are 20 input flows each sending at rates higher than 0.5Kbps, there is contention for the outgoing link bandwidth. We found almost equal bandwidth allocation for all flows. The maximum deviation from the average throughput offered to all flows in the Constant traffic sources case is 0.3869%. In the Poisson case it is bounded by 0.3391% from the average throughput. Thus, DRR appears to work well regardless of the input traffic distributions.

6 Non-backlogged Analysis

The analysis in Section 4 showed that DRR provides almost the same throughput to any two backlogged flows that have the same quantum size (Corollary 4.3). First, note that Corollary 4.3 compares flows under two assumptions: i) the execution begins with a deficit counter of 0 for all flows ii) the comparison is done at the end of a round. However, it

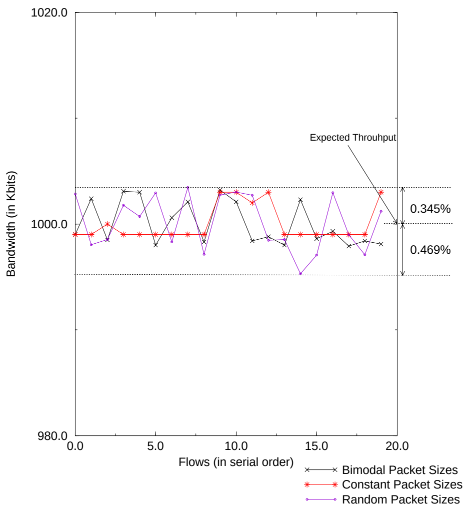

Figure 9: The bandwidth offered to different flows with exponential interpacket times and constant, bimodal and random packet sizes.

is easy to remove these two assumptions. i) We know from Lemma 4.1 that the maximum value of the deficit counter at the start of a round is Max ii) If we examine a flow i in the middle of a round before it has been serviced, the maximum discrepancy (compared to examining a flow at the end of a round), is $Quantum_i$. Thus it is easy to extend Corollary 4.3 to show that the maximum difference in the bytes sent by two flows that have the same quantum size Q during any execution (*regardless* of how executions begin or end) is $2Max + Q$. We use this value in the conjecture we state below.

It is natural to ask whether this result changes if either of the two flows are not backlogged at any point. Our simulations indicate that it does not, but we would like a theoretical result. Is it possible that a flow loses its quantum size periodically because the queue for the flow is empty periodically? Intuitively, if a flow is not backlogged at any instant t, it means that the flow has sent all that it can possibly sent by time t. Thus if the queue for a flow is periodically empty and the flow misses round-robin opportunities, these are opportunities that the flow could not use anyway. Of course, a flow could be very bursty and have nothing arrive for a large period (during which it naturally gets no service) and then have a large amount of data arrive during a small amount of time. But in that case, other FQ schemes like bit-by-bit round-robin (BR) will not give the flow much better treatment.

Thus it is hard to compare the performance of two non-backlogged flows because one flow could be much more bursty and hence have much worse performance. We might consider comparing two flows that have similar probability distributions for their arrival processes. We can, however, avoid the need for postulating a priori probability distributions, by comparing the performance of DRR to a fair scheme like BR for an *arbitrary* execution. This notion of *competitive analysis* was introduced by Sleator and Tarjan in a seminal paper [ST85].

Consider a set of n flows. Consider an arrival pattern F which specifies the interarrival times of packets to flows and identifies the size of each arriving packet and which flow it belongs to. Consider any arbitrary arrival patterns F. Assume that F is applied for some time t to two routers R and R' which are identical except that R uses DRR while R' uses BR. This results in two executions, say $E(F,t)$ and $E'(F,t)$. We state the following conjecture:

Conjecture 6.1 *For any F and any t and any flow i, let $sent_{i,t}$ be the bytes sent in execution $E(F,t)$, and let $sent'_{i,t}$ be defined similarly. Then for all F, t, and i:*

$$|sent_{i,t} - sent'_{i,t}| \leq 2Max + Quantum_i.$$

If it can be proved, this is an extremely strong result. It states that the difference in throughput allocated to any flow i under the DRR and BR disciplines is bounded by a constant factor *regardless* of the arrival pattern of packets for flow i or the other flows.

7 Latency Requirements

Consider a packet p for flow i that arrives at a router. Assume that the packet is queued for an output link instantly and there are no other packets for flow i at the router. Let s be the size of packet p in bits. If we use bit–by–bit round-robin then the packet will be delayed by s round-robin rounds, Assuming that there are no more than n active flows at any time, this leads to a latency bound of $n * s/B$, where B is the bandwidth of the output line in bits/sec. In other words, a small packet can only be delayed by an amount proportional to its own size by every other flow. The DKS (Demers–Keshav– Shenker) approximation only adds a small error factor to this latency bound.

The original motivation in both the work of Nagle and DKS was the notion of isolation. Isolation is essentially a throughput issue: we wish to give each flow a fair share of the overall throughput. In terms of isolation, the proofs given in the previous section indicate that Deficit Round Robin is competitive with Fair Queuing. However, the additional latency properties of BR and DKS have attracted considerable interest. In particular, Parekh and Gallager [PG93] have calculated bounds for end-to-end delay assuming the use of DKS fair queuing at routers and token bucket traffic shaping at sources.

At first glance, Deficit Round Robin (DRR) fails to provide strong latency bounds. In the example of the arrival of packet p given above, the latency bound provided by DRR is $\frac{\sum_i^n Quantum_i}{B}$. In other words, a small packet can be delayed by a quantum's worth by every other flow. Thus in the case where all the quanta are equal to Max (which is needed to make the work $O(1)$), the ratio of the delay bounds for DRR and BR is Max/Min.

However, the real motivation behind providing latency bounds is to allow real-time traffic to have predictable and dependable performance. Since most traffic will consist of a mix of best-effort and real-time traffic, the simplest solution is to reserve a portion of the bandwidth for real-time traffic and use a separate Fair Queuing algorithm for the real-time traffic while continuing to use DRR for the best-effort traffic. This allows efficient packet processing for best-effort traffic; at the same time it allows the use of other fair queuing schemes that provide delay bounds for real-time traffic at reasonable cost.

As a simple example of combining fair queuing schemes, consider the following modification of Deficit Round Robin called DRR+. In DRR+ there are two classes of flows: *latency critical* and *best-effort*. A latency critical flow must contract to send no more than x bytes in some period T. If a latency critical flow f meets its contract, whenever a packet for flow f arrives to an empty flow f queue, *the flow f is placed at the head of the round-robin list.*

Suppose for instance that each latency critical flow guarantees to send at most a single packet (of size at most s) every T seconds. Assume that T is large enough to service one packet of every latency critical flow as well as one quantum's worth for every other flow. Then if all latency critical flows meet their contract, it appears that each latency critical flow is at most delayed by $((n' * s) + Max)/B$, where n' is the number of latency critical flows. In other words, a latency critical flow is delayed by one small packet from every other latency critical flow, as well as an error term of one maximum size packet (the error term is inevitable in all schemes unless the router preempts large packets). In this simple case, the final bound appears better than the DKS bound because a latency critical flow is only delayed by other latency critical flows.

In the simple case, it is easy to police the contract for latency critical flows. A single bit that is part of the state of such a flow is cleared whenever a timer expires and is set whenever a packet arrives; the timer is reset for T time units when a packet arrives. Finally, if a packet arrives and the bit is set, the flow has violated its contract; an effective (but user-friendly) countermeasure is to place the flow ID of a deviant flow at the end of the round-robin list. This effectively moves the flow from the class of latency critical flows to the class of best effort flows.

DRR+ is a simple example of combining DRR with other fair queuing algorithms to handle latency critical flows. By using other schemes for the latency critical flows, we can provide better bounds while allowing more general traffic shaping rules.

8 Conclusions

We have described a new scheme, Deficit Round Robin (DRR), that provides near-perfect isolation at very low implementation cost. As far as we know, this is the first fair queuing solution that provides near-perfect throughput fairness with $O(1)$ packet processing. DRR should be attractive to use while implementing Fair Queuing at gateways and routers.

We have described theorems that describe the behavior of DRR in backlogged traffic scenarios. We have not completely understood its behavior in non–backlogged cases, though we have conjectured that its throughput differs from the behavior of bit-by-bit round robin by at most a constant additive factor. Our simulations support this conjecture and indicate that DRR works as well in non–backlogged cases.

The *Quantum* size required for keeping the work O(1) is high (at least equal to Max). We feel that while Fair Queuing using DRR is general enough for any kind of network, it is best suited for datagram networks. In ATM networks, packets are fixed size cells; therefore Nagle's solution (simple round robin) will work as well as DRR. However, if connections in an ATM network require weighted fair queuing with arbitrary weights, DRR will be useful.

DRR can be combined with other FQ algorithms such that DRR is used to service only the best-effort traffic. We described a trivial combination algorithm called DRR+ that offers good latency bounds to *LatencyCritical* flows as long as they meet their contracts. However, even if the source meets the contract, the contract may be violated due to "bunching" effects at intermediate routers. Thus other combinations need to be investigated. Recall that DRR requires having the quantum size be at least a maximum size packet in order for the packet processing work to be low; this does affect delay bounds.

We believe that DRR should be easy to implement using existing technology. It only requires a few instructions beyond the simplest queuing algorithm (FCFS), and this addition should be a small percentage of the instructions needed for routing packets. The memory needs are also modest; 6K size memory should give a small number of collisions for about 100 concurrent flows. This is a small amount of extra memory compared to the buffer memory used in many routers. Note that the buffer size requirements should be identical to the buffering for FCFS because in DRR buffers are shared between queues using McKenney's buffer stealing algorithm.

Lastly, we note that deficit round robin schemes can be applied to other scheduling contexts in which jobs must be serviced as whole units. In other words, jobs cannot be served in several "time slices" as in a typical operating system. This is true for packet scheduling because packets cannot be interleaved on the output lines, but it is true for other contexts as well. Note also that DRR is applicable to distributed queues because it needs only local information to implement. For instance, the current 802.5 token ring uses token holding timers that limit the number of bits a station can send at each token opportunity. If a station's packets do not fit exactly into the allowed number of bits, the remainder is not kept track off; this allows the possibility of unfairness. The unfairness can easily be removed by keeping a deficit counter at each ring node and by a small modification to the token ring protocol.

Another application is load balancing or, as it is sometimes termed, striping. Consider a router that has traffic arriving on a high speed line that needs to be sent out to a destination over multiple slower speed lines. If the router sends packets from the fast link in a round robin fashion across the slower links, then the load may not balance perfectly if the packet sizes are highly variable. For example, if packets alternate between large and small packets, then round robin across two lines can cause the second line to be underutilized. But load balancing is almost the inverse of fair queuing. It is is not hard to see that deficit round robin solves the problem; we send up to to a quantum limit per output line but we keep track of deficits. This should produce nearly perfect load balancing; as usual it can be extended to weighted load balancing. In [APV95], we show how to obtain perfect load balancing and yet guarantee FIFO delivery. Our load balancing scheme [APV95] appears to be a novel solution to a very old problem.

For these reasons, we believe that deficit round robin scheduling is a general and useful tool. We hope our readers will use it in other contexts.

Acknowledgments: A number of people listened patiently to our ideas and provided us with valuable feedback. Prominent among them are: Dave Clark, Sally Floyd, Andy Fingerhut, Srinivasan Keshav, Paul McKenney, and Lixia Zhang. We thank Andy Fingerhut and S. Keshav for their careful reading of the paper. We thank R. Gopalakrishnan and Apostolos Dalianis for valuable discussions.

References

[APV95] H. Adiseshu, G. Parulkar, and G. Varghese. *Reliable FIFO Load Balancing over Multiple FIFO Channels.* Washington University Technical Report 95-11, available by FTP.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Proc. Sigcomm '89*, 19(4):1–12, September 1989.

[Flo93a] S. Floyd. Notes on guaranteed service in resource management. Unpublished note. 1993.

[Flo93b] S. Floyd. Personal communication. 1993.

[GM90] A. Greenberg and N. Madras. How fair is fair queueing. In *Proc. Performance '90*, 1990.

[Gol94] S. Golestani. A self clocked fair queueing scheme for broadband applications. In *Proc. IEEE Infocomm '94*, 1994.

[JR86] R. Jain and S. Routhier. Packet trains measurement and a new model for computer netwoek traffic. *IEEE Journal on Selected Areas in Communications*, Sept. 1986.

[Kes91] S. Keshav. On the efficient implementation of fair queueing. In *Internetworking: Research and Experience Vol.2, 157-173*, September 1991.

[McK91] P. McKenney. Stochastic fairness queueing. In *Internetworking: Research and Experience Vol.2, 113-131*, January 1991.

[Nag87] John Nagle. On packet switches with infinite storage. *IEEE Trans. on Comm.*, COM-35(4), April 1987.

[PG93] A. K. Parekh and R. G. Gallagher. A generalized processor sharing approach to flow control in integrated services networks. In *Proc. IEEE Infocomm '93.*

[SA94] D. Saha and M. Saksena and S. Mukherjee and S. Tripathi. On Guaranteed Delivery of Time-Critical Messages in DQDB. In *Proc. IEEE Infocomm '94.*

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.

[Zha91] L. Zhang. Virtual clock: A new traffic control algorithm for packet switched networks. *ACM Trans. on Computer Systems*, 9(2):101–125, May 1991.