

# Fair and Efficient Packet Scheduling Using Elastic Round Robin

Salil S. Kanhere, *Student Member, IEEE*, Harish Sethu, *Member, IEEE*, and Alpa B. Parekh

**Abstract**—Parallel systems are increasingly being used in multiuser environments with the interconnection network shared by several users at the same time. Fairness is an intuitively desirable property in the allocation of bandwidth available on a link among traffic flows of different users that share the link. Strict fairness in traffic scheduling can improve the isolation between users, offer a more predictable performance and improve performance by eliminating some bottlenecks. This paper presents a simple, fair, efficient, and easily implementable scheduling discipline, called *Elastic Round Robin (ERR)*, designed to satisfy the unique needs of wormhole switching, which is popular in interconnection networks of parallel systems. In spite of the constraints of wormhole switching imposed on the design, ERR is also suitable for use in Internet routers and has better fairness and performance characteristics than previously known scheduling algorithms of comparable efficiency, including Deficit Round Robin and Surplus Round Robin. In this paper, we prove that ERR is efficient, with a per-packet work complexity of  $O(1)$ . We analytically derive the relative fairness bound of ERR, a popular metric used to measure fairness. We also derive the bound on the start-up latency experienced by a new flow that arrives at an ERR scheduler. Finally, this paper presents simulation results comparing the fairness and performance characteristics of ERR with other scheduling disciplines of comparable efficiency.

**Index Terms**—Fair queuing, Deficit Round Robin, Surplus Round Robin, relative fairness bound, quality of service, wormhole networks.

## 1 INTRODUCTION

IN interconnection networks of parallel systems, packets belonging to different traffic flows often share links in their respective paths toward their destinations. Fairness is an intuitively desirable property in the allocation of bandwidth available on a link among multiple traffic flows that share the link. Fairness in packet scheduling becomes especially desirable with the increasing use of parallel systems in multiuser environments with the interconnection network shared by several users at the same time. Fair allocation of bandwidth at links within a network is a necessary requirement for providing protection to flows, i.e., for ensuring that the performance is not affected when another possibly misbehaving flow tries to send packets at a rate faster than its fair share. In multiuser environments, the protection guaranteed by fair scheduling of packets improves the isolation between users, a quality strongly desired by customers of parallel systems [1]. Isolation offers a more predictable performance to user applications, which also facilitates repeatability of performance results necessary for reliable benchmarking of systems and applications without taking all the users off the system. Strict fairness is also desirable for good performance since unfair treatment of some traffic flows in the network can easily lead to unnecessary bottlenecks.

Most switch architectures designed for interconnection networks of parallel systems, however, eliminate only the worst kinds of unfairness, such as starvation, where packets belonging to one traffic flow may not be scheduled for an indefinite period of time. In the most commonly implemented scheduling discipline, First-Come-First-Served (FCFS), packets are scheduled in the order of their arrival times. The advantage of this scheme is its simplicity since the scheduler is not even required to distinguish packets by the flows to which they belong. The difficulty with FCFS is that it does not provide adequate protection from a sudden bursty source sending packets at a rate higher than its fair share for brief periods of time. Such a source can significantly increase the mean delay of packets belonging to flows from other sources. An alternate technique is Packet-Based Round Robin (PBRR) in which packets are queued separately, based on the flows to which they belong, and the scheduler transmits one packet from each queue in a round-robin fashion [2]. The PBRR scheduler, however, is not fair since flows sending longer packets can use up an unfairly high fraction of the available bandwidth.

One of the difficulties in designing a fair scheduling algorithm for interconnection networks of parallel systems is the unique restriction on the scheduler imposed by wormhole switching [3], a popular technique used in these networks. Wormhole switching and its variations are widely used in a variety of parallel systems and more recently in system area networks [4], [5], [6], [7], [8], [9]. Wormhole switching is distinguished by the fact that the granularity of flow control in the network can be smaller than a packet. This unit of flow control is called a *flit*. In order to not add to the per-flit overhead, only the head flit (the first flit) of a packet contains information necessary to route the packet through the network. A switch in the

- S.S. Kanhere and H. Sethu are with the Department of Electrical and Computer Engineering, Drexel University, 3141 Chestnut Street, Philadelphia, PA 19104. E-mail: {salil, sethu}@ece.drexel.edu.
- A.B. Parekh is with Lockheed Martin Global Telecommunications, Clarksburg, MD 20871. E-mail: alpa.parekh@lmco.com.

Manuscript received 28 June 2000; accepted 12 July 2001.

For information on obtaining reprints of this article, please send e-mail to: [tpds@computer.org](mailto:tpds@computer.org), and reference IEEECS Log Number 112362.

network reads the information in the head flit and directs it to the next switch or the end-system device in its path. The rest of the flits of the packet follow the path of the head flit.

Consider a packet scheduler at an output link in a wormhole switch serving several queues, each corresponding to a different flow and consisting of flits ready to be forwarded to the output link. We define a *queue* as a logical entity containing a sequence of flits that have to be served in a FIFO order. Note that, depending on the buffering architecture of the switch, a queue may not be the same thing as a buffer since a single buffer can implement multiple logical queues [10], [11]. In wormhole switching, flits from different flows cannot be multiplexed on the same link unless each flit is appropriately marked to distinguish it from flits belonging to other flows. In wormhole networks with virtual channels [12], each flit is marked by the virtual channel it belongs to and, therefore, it is possible to time-multiplex flits from different channels on the same physical link. If each flow can be assigned a separate virtual channel, one may use the Flit-Based Round Robin (FBRR) scheduler which visits the flow queues in round-robin fashion, and transmits one flit from each queue. This scheme is very fair among the flows in terms of the number of flits scheduled from each flow over any interval of time. However, this scheme for achieving fairness is prohibitively expensive since it can only be used if there are as many virtual channels implemented as there are flows (which can be in hundreds or thousands). In addition, by serving packets flit-by-flit, the FBRR scheme uniformly increases the delay of packets in all the flows [13]. A packet-by-packet scheduler, therefore, is more desirable since it can achieve a better average delay.

Wormhole switches typically use a flit-by-flit credit-based flow control protocol and, therefore, downstream congestion can thwart the progress of the packet currently being served for an unpredictable length of time. Since, as explained earlier, it may not always be possible to time-multiplex the transmission of packets from different flows, one cannot always begin forwarding packets from another flow until all flits belonging to the packet currently being served are forwarded. Thus, during the time that a packet is in the middle of its transmission, packets from other flows may be blocked without access to the output link even while there are no flits being transmitted over the link. Therefore, the relevant measure of the use of a resource, in this case the output link, is the length of time a flow occupies the link. In wormhole networks, therefore, fairness should be based on the length of time each flow occupies a link and not on the number of flits sent by each flow over the link. This length of time depends on the downstream congestion, which can be hard to predict without complex feedback mechanisms. In wormhole networks, unlike in Internet routers and many other networks, this length of time cannot be accurately estimated from knowledge of the length of the packet being transmitted. The actual length of time that a packet takes to be dequeued, thus, may not be known until the last flit of the packet is dequeued. A scheduling discipline for wormhole networks, therefore, should be able to make a decision on starting the transmission of a packet without knowledge of the length

of time it will take to transmit the entire packet. In addition, the algorithm also cannot assume an upper bound on this length of time. In other words, the unique requirements of wormhole switching require that a scheduler perform its operations without *any* assumptions on how long it will take to transmit a packet.

In traditional scheduling literature, it is typically assumed that the length of time it takes to transmit a packet is directly proportional to the size of the packet. Therefore, the problem of designing a fair scheduler for wormhole networks is equivalent to the problem of designing a fair scheduler in the traditional sense, but without the scheduler making *any* assumptions on the size of a packet before beginning the transmission. Based on this equivalence, we present our paper as a solution to the latter problem. It should be noted that in many real interconnection networks, as well as in Internet routers, packet headers do carry a field with the packet length in it and, therefore, the problem in such cases is not a lack of knowledge of the packet length. However, when a scheduler uses the size of a packet to make its decisions, it cannot be readily adapted to the unique requirements of wormhole switching.

Over the last decade, a variety of algorithms that seek to achieve fairness in bandwidth allocation have been proposed and implemented in Internet routers [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. A number of these scheduling disciplines are discussed in Section 2. Unfortunately, most fair scheduling disciplines proposed for Internet routers are either too expensive to implement in high-speed hardware switches because of the work complexity of per-packet processing or cannot be easily adapted to the unique requirements of wormhole networks described above. For example, most timestamp-based schedulers, such as Weighted Fair Queuing [24], have a work complexity of  $O(\log n)$  with respect to the number of flows. On the other hand, more efficient schedulers, such as Deficit Round Robin (DRR) [22] and Surplus Round Robin (SRR) [18], [19], [20] require knowledge of the upper bound on packet lengths to achieve a work complexity of  $O(1)$ , rendering them difficult to adapt to wormhole networks. In this paper, we present *Elastic Round Robin (ERR)*, a new, simple, fair, efficient, and low-latency packet scheduling discipline that can be used in both Internet routers and wormhole switches. Besides being efficient, ERR has better fairness and performance characteristics than previously known scheduling algorithms of comparable efficiency, including Deficit Round Robin and Surplus Round Robin.

Section 3 presents the ERR scheduling algorithm along with the rationale behind it. Section 4 presents analytical results on the efficiency, fairness, and the performance characteristics of ERR. We consider a scheduler efficient if the order of the work complexity of enqueueing and dequeuing a packet, with respect to the number of flows, is  $O(1)$ . We prove that the work complexity of ERR is  $O(1)$ , equal to or better than other scheduling disciplines. We measure fairness using a well-known and widely used metric, known as the relative fairness bound [17]. We prove that the relative fairness bound of ERR is  $3m$ , where  $m$  is the size of the largest packet that *actually* arrives during the execution of ERR. Finally, Section 4 also presents bounds on

the start-up latency of ERR, the delay experienced by the first packet of a new flow. A low start-up latency is critical to obtaining low latencies with control packets. The results in this section present an analytical proof that the ERR algorithm has better fairness properties, as well as better performance characteristics, than other fair scheduling disciplines of comparable efficiency, such as DRR and SRR.

Section 5 presents several simulation results on the fairness and performance characteristics of this algorithm in comparison with other algorithms. Section 6 concludes the paper, with a tabulated summary of the properties of ERR in comparison to other fair scheduling algorithms. The concluding section also includes a brief discussion of other applications of ERR.

## 2 BACKGROUND AND PREVIOUS WORK

Traditionally, a *flow* is defined as a sequence of packets generated by the same source and headed toward the same destination via the same path in the network. It is assumed that packets belonging to different flows are queued separately while they await transmission. A *scheduler* dequeues packets from these queues and forwards them for transmission. A flow is said to be *active* during a period if its queue is nonempty throughout this period. A flow is *inactive* when its queue is empty. Note that, even though we use the above traditional definition of a flow to present our results in this paper, a flow can also be more broadly defined as any distinct sequence of packets queued separately at the scheduler and competing with other sequences of packets for service by the scheduler. For example, in parallel systems, a flow may also be defined as the set of all packets belonging to the same user, with packets of these flows queued at the scheduler accordingly.

A precise definition of fairness is essential before further discussion of fair scheduling of flows. The classic notion of fairness in the allocation of a resource among multiple requesting entities with equal rights to the resource, but unequal demands, is as follows [24]:

- The resource is allocated in order of increasing demand.
- No requesting entity gets a share of the resource larger than its demand.
- Requesting entities with unsatisfied demands get equal shares of the resource.

The Generalized Processor Sharing (GPS) algorithm is an unimplementable but ideal scheduling discipline, which satisfies the above notion of absolute fairness [14], [16]. The GPS scheduler visits each active flow's queue in a round-robin fashion and serves an infinitesimally small amount of data from each queue in such a way that during any finite interval of time, it can visit each queue at least once. Consider a set of  $n$  flows denoted by  $1, 2, \dots, n$  demanding bandwidths  $b_1, b_2, \dots, b_n$  on a link of total bandwidth  $B$ . Without loss of generality, assume  $b_1 \leq b_2 \leq \dots \leq b_n$ . The GPS scheduler first allocates  $B/n$  of the bandwidth to each of the active flows. If this is more than the bandwidth demanded by flow 1, the unused bandwidth,  $B/n - b_1$ , is divided equally among the remaining  $n - 1$  flows. If the

total bandwidth allocated thus far to flow 2 is more than  $b_2$ , the unused excess bandwidth is again divided equally, this time among the remaining  $n - 2$  flows. The allocation process of the GPS scheduler continues in this fashion until each flow has received no more than its demand and, if the demand was not satisfied, no less than any other flow with higher demand.

Over the last decade, a variety of algorithms that seek to achieve fairness in bandwidth allocation as per the above definition have been proposed and implemented in Internet routers. Algorithms, such as *Weighted Fair Queuing (WFQ)* [14], [15], try to emulate the ideal GPS scheduler by time-stamping each arriving packet with a *finish number*, the expected completion time of the packet if it were scheduled by the GPS scheduler. The WFQ scheduler then serves the packets in the increasing order of the finish numbers. WFQ is not very efficient in the method it uses to compute the timestamps. In addition, WFQ suffers from the cost associated with sorting among the timestamps, incurring a work complexity of  $O(\log n)$ , where  $n$  is the number of flows. Another scheduling discipline, *Worst-case Fair Weighted Fair Queuing (WF<sup>2</sup>Q)* [21], improves upon the emulation of GPS and makes the implementation easier. In fact, it can be shown that no packet-by-packet scheduler can be more fair than WF<sup>2</sup>Q. However, WF<sup>2</sup>Q suffers from the same work complexity as WFQ.

A simpler implementation is achieved by *Self-Clocked Fair Queuing (SCFQ)* [17], which uses the finish number of the packet currently being transmitted in the computation of finish numbers for arriving packets. SCFQ is slightly less fair than WFQ and also has a larger delay bound. *Start-time Fair Queuing (SFQ)* [23], is a variant of SCFQ which uses the starting time of the packet currently in service to compute the timestamp of the arriving packet. SFQ has better fairness properties than SCFQ and lower worst-case delays. Flows, instead of packets, are timestamped in another approach of similar complexity, called *Time-Shift Scheduling* [25]. In this scheme, the scheduler's real-time clock is periodically adjusted to achieve the effect of bounding the difference between any pair of flow timestamps, thus ensuring fair scheduling of flows. The complexity of timestamp computations is further reduced in two recently proposed timestamp-based algorithms, *Frame-Based Fair Queuing (FFQ)* and *Starting Potential-Based Fair Queuing (SPFQ)* [26]. FFQ, however, uses a framing approach in which the fairness depends on the frame size chosen in the implementation. SPFQ has better fairness properties at a cost of more complexity than FFQ.

None of the scheduling disciplines described above, however, avoid the  $O(\log n)$  work complexity associated with sorting among the timestamps. *Deficit Round Robin (DRR)* [22], a less fair but more efficient scheduling discipline with an  $O(1)$  per-packet work complexity, was proposed by Shreedhar and Varghese in 1996. DRR is not a timestamp-based algorithm and, therefore, avoids the associated computational complexity. DRR serves active flows in a strict round-robin order and succeeds in eliminating the unfairness of pure packet-based round-robin by maintaining a *deficit counter (DC)* to keep an account of past unfairness. A *quantum* is assigned to each of the flows

and when a flow is picked for service, its DC is incremented by the quantum value for that flow. A packet is served from a flow only if the packet size at the head of the flow queue is less than the sum of DC and the quantum value; otherwise, the scheduler begins serving the next flow in the round-robin sequence. When a packet is transmitted, the DC corresponding to that flow is decremented by the size of the transmitted packet.

In DRR, in order that the per-packet work complexity is  $O(1)$ , one has to make sure that the quantum value chosen is no smaller than the size of the largest packet that may potentially arrive at the scheduler [22]. Otherwise the per-packet work complexity increases to  $O(n)$  since one may encounter a situation in which, even after visiting each of the  $n$  flows and examining the respective DC values, no packet is eligible for transmission. A per-packet work complexity of  $O(1)$  is ensured if we make sure that at least one packet is transmitted from each active flow during each round. This is ensured if the quantum is no smaller than the size of the largest possible packet since this guarantees that the packet size at the head of each queue at the start of its service opportunity will always be less than the sum of the DC value and the quantum value of the flow. In order to achieve a per-packet work-complexity of  $O(1)$ , therefore, the DRR scheduler requires knowledge of the upper bound on the size of a packet. Thus, DRR is not ideally suitable for wormhole networks since it requires the knowledge of the size of a packet before making a decision on transmitting it and, in addition requires an upper bound on the size of a packet.

In [18], [19], a fair scheduler similar to DRR was proposed. This algorithm, later known as *Surplus Round Robin (SRR)*, has also been used in other contexts, such as in [20]. SRR is a modified version of DRR in which the scheduler continues serving a flow as long as the DC value of the flow is positive. When the DC becomes negative, the scheduler begins serving the next flow in the round-robin sequence. Thus, while DRR never allows a flow to overdraw its account but rewards an under-served flow in the next round, SRR allows a flow to overdraw its account but penalizes the flow accordingly in the next round. DRR keeps an account of each flow's deficit in service, while SRR keeps an account of the surplus service received by each flow. SRR does not require the scheduler to know the length of a packet before scheduling it. However, it does require the use of a fixed quantum assigned to each flow per round. As in DRR, in order to ensure an  $O(1)$  per-packet work complexity, the quantum value has to be no smaller than the size of the largest packet that may potentially arrive at the scheduler. SRR, like DRR, cannot be readily adapted for use in wormhole switching since it also requires knowledge of the upper bound on packet sizes.

### 3 ELASTIC ROUND ROBIN

Even though ERR was designed for wormhole networks, it can be used in a wide variety of contexts whenever there is a shared resource that needs to be allocated fairly among multiple requesting entities. In some of these contexts, its unique properties relevant to wormhole switching are critical and, in some others, its advantages derive from its

```

Initialize: (Invoked when the scheduler is initialized)
RoundRobinVisitCount = 0;
PreviousMaxSC = 0;
for (i = 0; i < n; i = i + 1)
    SCi = 0;

Enqueue: (Invoked when a packet arrives)
i = QueueInWhichPacketArrives;
if (ExistsInActiveList(i) == FALSE) then
    AddToActiveList(i);
    Increment SizeOfActiveList;
    SCi = 0;
end if;

Dequeue:
while (TRUE) do
    if (RoundRobinVisitCount == 0) then
        PreviousMaxSC = MaxSC;
        RoundRobinVisitCount = SizeOfActiveList;
        MaxSC = 0;
    end if;
    i = HeadOfActiveList;
    RemoveHeadOfActiveList;
    Ai = 1 + PreviousMaxSC - SCi;
    Senti = 0;
    do
        TransmitPacketFromQueue(i);
        Increase Senti by LengthInFlitsOfTransmittedPacket;
    while (Senti < Ai);
    SCi = Senti - Ai;
    if (SCi > MaxSC) then
        MaxSC = SCi;
    end if;
    if (QueueIsEmpty == FALSE) then
        AddQueueToActiveList(i);
    else
        SCi = 0;
        Decrement SizeOfActiveList;
    end if;
    Decrement RoundRobinVisitCount;
end while;

```

Fig. 1. Pseudo-code for ERR.

simplicity, better fairness and better performance characteristics. Because of the wide applicability of our solution and so that this work may be readily understood and used in a broader variety of contexts, we present this algorithm as a solution to the following abstraction of the problem.

Consider  $n$  flows, each with an associated queue with packets in it. The scheduler dequeues packets from these queues according to a scheduling discipline and forwards them for transmission over an output link. As in traditional scheduling problems, we allow that the length of time it takes to dequeue a packet is proportional to the size of the packet—however, to apply this work to wormhole networks, it is required that the scheduling algorithm not make any assumptions about the length of a packet prior to completely transmitting the packet.

We now proceed to describe the ERR scheduler which meets the above requirement. A pseudo-code implementation of the ERR scheduling algorithm is shown in Fig. 1, consisting of *Initialize*, *Enqueue*, and *Dequeue* routines. The *Enqueue* routine is called whenever a new packet arrives at a flow. The *Dequeue* routine is the heart of the algorithm which schedules packets from the queues corresponding to different flows. In this paper, we use a flit as the smallest

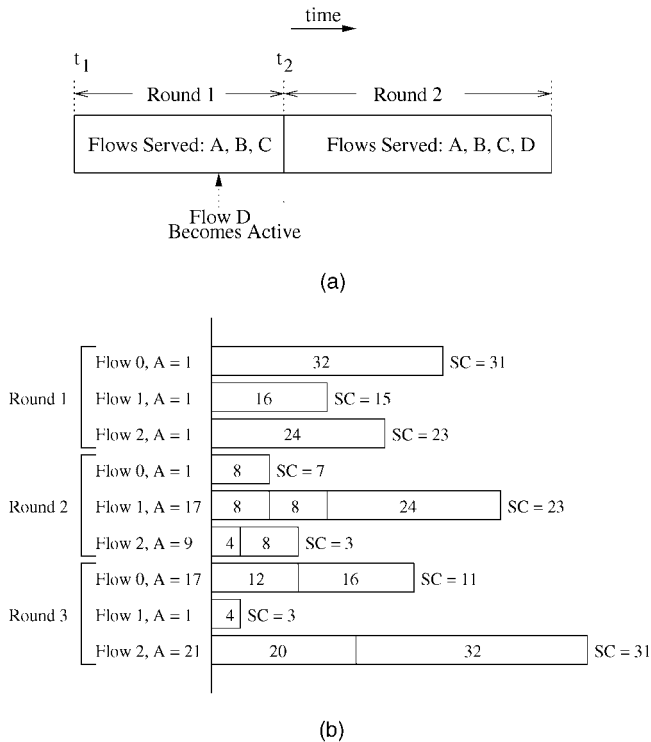


Fig. 2. (a) Definition of a Round and (b) an illustration of three rounds in an ERR execution.

piece of a packet that can be independently scheduled and we measure the length of a packet in flits.

We maintain a linked list, called the *ActiveList*, of flows which are active. A flow whose queue was previously empty and therefore not in the *ActiveList* is added to the tail of the list whenever a new packet belonging to the flow arrives. The ERR scheduler serves the flow  $i$  at the head of this list. After serving flow  $i$ , if the queue of flow  $i$  becomes empty, it is removed from the list. On the other hand, if the queue of flow  $i$  is not empty after it has received its round-robin service opportunity, flow  $i$  is added back to the tail end of the list.

Consider the instant of time,  $t_1$ , when the scheduler is first initialized. We define *Round 1* as one round-robin iteration starting at time  $t_1$  and consisting of visits to all the flows that were in the *ActiveList* at time  $t_1$ . We illustrate this definition of a round using Fig. 2a. Assume that flows  $A$ ,  $B$ , and  $C$  are the only flows active at the beginning of *Round 1*. The visits of the scheduler to flows  $A$ ,  $B$ , and  $C$ , comprise *Round 1*. Let flow  $D$  become active after the time instant  $t_1$ , but before the completion of *Round 1*. Let the time instant  $t_2$  mark the completion of *Round 1*. The scheduler does not visit flow  $D$  in *Round 1* since  $D$  was not in the *ActiveList* at the start of *Round 1*. *Round 2* is now defined as consisting of the visits to all of the flows that are in the *ActiveList* at time  $t_2$ . Assuming that flows  $A$ ,  $B$ , and  $C$  are still active at time  $t_2$ , *Round 2* will consist of visits to the flows  $A$ ,  $B$ ,  $C$ , and  $D$ . In general, we define round  $i$  recursively as the set of visits to all the flows in the *ActiveList* at the instant round  $(i - 1)$  is completed. In order that the scheduler knows the number of flows it has to visit in any given round, we introduce the quantity *RoundRobinVisitCount* which denotes the number of flows that are in the *ActiveList* at the start of a round.

*RoundRobinVisitCount* is decremented by one after each flow is served and, when it eventually equals zero, it implies the end of a round.

In each round, the scheduling algorithm determines the number of flits that a flow is allowed to send. We call this quantity the *allowance* for the flow during that round. The allowance assigned to flow  $i$  during round  $r$  is denoted by  $A_i(r)$ . This allowance, however, is not a rigid one and is actually *elastic* in that a flow may be allowed to send more flits in a round than its allowance. Let  $Sent_i(r)$  be the number of flits that are transmitted from the queue of flow  $i$  in round  $r$ . The ERR scheduler will begin serving the next packet from the queue if the total number of flits transmitted by the flow so far in the current round is less than its allowance. The ERR scheduler, thus, makes the scheduling decision without any knowledge about the packet length. Note that the last packet transmitted by a flow may cause it to exceed its allowance, as can happen when the allowance is smaller than the size of the packet at the head of the corresponding queue. When a flow ends up sending more than its allowance, it is interpreted as having obtained more than its fair share of the bandwidth. The scheduler records this unfairness in the *Surplus Count* (SC) associated with each flow. The surplus count, during any round, is the number of flits the flow sent in addition to its allowance. Let  $SC_i(r)$  denote the surplus count of flow  $i$  in round  $r$ . Then, after serving flow  $i$  in round  $r$ , the scheduler computes  $SC_i(r)$  as follows:

$$SC_i(r) = Sent_i(r) - A_i(r). \quad (1)$$

Let  $MaxSC(r)$  denote the largest surplus count among all the flows served during round  $r$ . This quantity is used to recursively compute the allowances for each of the flows in the next round, using the following equation:

$$A_i(r) = 1 + MaxSC(r - 1) - SC_i(r - 1). \quad (2)$$

Note that, for the flow with the largest surplus count in the previous round, the new allowance is 1. This is ensured by the addition of 1 in (2) so that the scheduler will transmit at least one packet from this flow during the next round.

The allowance given to each of the flows in a given round is not fixed and is computed depending on the behavior of the flows in the previous round. After the ERR scheduler serves flow  $i$ , if the queue of flow  $i$  is empty, its surplus count is reset to zero and it is removed from the *ActiveList*. Otherwise, if flow  $i$  has packets in its queue that are ready for transmission, it is added back at the tail end of the list.

Fig. 2b illustrates the first three rounds in an execution of the ERR scheduling discipline. In this figure, at the beginning of the first of these rounds, the surplus counts for all three flows and the *MaxSC* are all initialized to 0. Thus, from (2), the allowance during round 1 is equal to 1 for all the flows. The sizes of the packets actually sent by the flow during this round are shown by the horizontal bars and the new allowances for the next round are again computed using (1) and (2). It is easily observed from the figure that, in general, flows which receive very little service in a round are given an opportunity to receive proportionately more service in the next round.

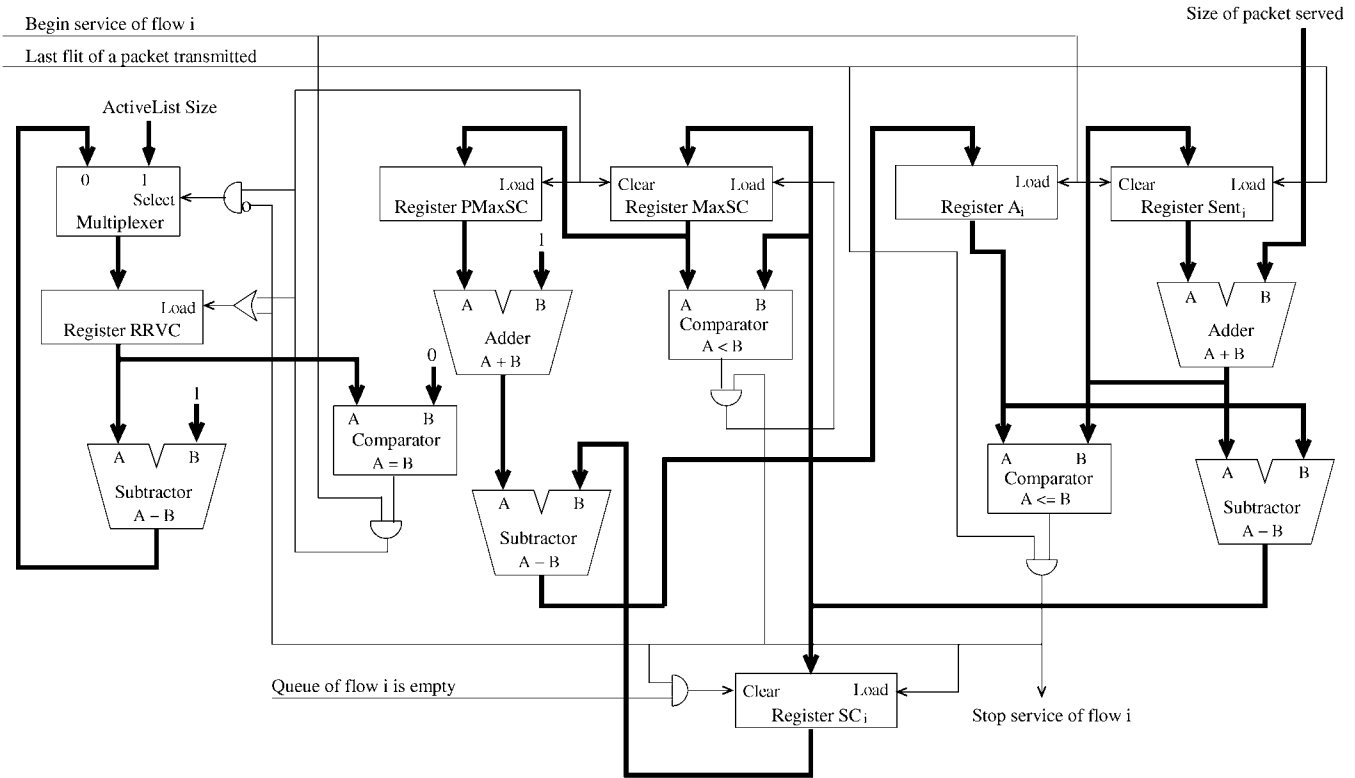


Fig. 3. A block diagram illustration of ERR.

Fig. 3 shows a block diagram of a portion of ERR to illustrate the various operations used to determine when the scheduler should stop service of one flow and begin service for another. In the diagram, *PreviousMaxSC* and *RoundRobinVisitCount* are abbreviated as *PMaxSC* and *RRVC*. The thin lines in the figure indicate single-bit signals, while the thick dark lines indicate multibit buses carrying quantities, such as packet sizes and values of various counters.

*Weighted ERR.* In the interests of clarity of presentation, in this section, we have discussed the ERR scheduler assuming that all the traffic flows have been given equal weights, i.e., equal rights to the bandwidth of the output link. One may, however, want some flows to receive a greater share of the bandwidth than some other flows. Let the weight associated with flow  $i$  be  $w_i$ , indicating its relative share of the bandwidth. The computation of the surplus count for the weighted ERR scheduler is identical to that of the unweighted version. However the allowance calculated by the Weighted ERR scheduler uses the following modified version of (2):

$$A_i(r) = w_i(1 + MaxSC(r - 1)) - SC_i(r - 1). \quad (3)$$

## 4 ANALYTICAL RESULTS

In this section, we analytically derive the work complexity, fairness, and delay properties of ERR.

### 4.1 Work Complexity

Consider an execution of the ERR scheduling discipline over  $n$  flows. We define the work complexity of the ERR scheduler as the order of the time complexity with respect to  $n$  of enqueueing and then dequeuing a packet for transmission.

**Theorem 1.** *The work complexity of an ERR scheduler is  $O(1)$ .*

**Proof.** We prove the theorem by showing that enqueueing and dequeuing a packet are each of time complexity  $O(1)$ .

The time complexity of enqueueing a packet is the same as the time complexity of the *Enqueue* procedure in Fig. 1, which is executed whenever a new packet arrives at a flow. Determining the flow at which the packet arrives is an  $O(1)$  operation. The flow at which the new packet arrives is added to the *ActiveList* if it is not already in the list. This addition of an item to the tail of a linked list data structure is also an  $O(1)$  operation.

We now consider the time complexity of dequeuing a packet. During each service opportunity, the ERR scheduler transmits at least one packet. Thus, the time complexity of dequeuing a packet is equal to or less than the time complexity of all the operations performed during each service opportunity. Each execution of the set of operations inside the while loop of the *Dequeue* procedure in Fig. 1, represents all operations performed during each service opportunity given to a flow. These operations include determining the next flow to be served, removing this flow from the head of the *ActiveList* and possibly adding it back at the tail. All of these operations on a linked list data structure can be executed in  $O(1)$  time. Additionally, each service opportunity includes updating the values of surplus count and allowance corresponding to the flow being served and updating the values of *MaxSC*, *PreviousMaxSC*, *SizeOfActiveList* and *RoundRobinVisitCount*. All of these can be done in constant time, as represented by the constant number of operations in the dequeue procedure in Fig. 1.  $\square$

## 4.2 Fairness

The fairness of a scheduling discipline is best measured in comparison to the GPS scheduling algorithm. This quantity, known as the *Absolute Fairness Bound* of a scheduler  $S$ , is defined as the upper bound on the difference between service received by a flow under  $S$  and that under GPS over all possible intervals of time. This bound is often difficult to derive analytically and, therefore, another fairness metric first proposed in [17] is more commonly employed. This metric, known as the *Relative Fairness Bound (RFB)*, is defined as the maximum difference in the service received by any two flows over all possible intervals of time. The following provides a more rigorous definition. In the following, a flow is considered *active* during an interval of time if, during this interval, its queue is never empty of packets awaiting transmission.

**Definition 1.** Let  $Sent_i(t_1, t_2)$  be the number of flits transmitted by flow  $i$  during the time interval between  $t_1$  and  $t_2$ . Given an interval  $(t_1, t_2)$ , we define the Relative Fairness,  $RF(t_1, t_2)$  for this interval as the maximum value of  $|Sent_i(t_1, t_2) - Sent_j(t_1, t_2)|$  over all pairs of flows  $i$  and  $j$  that are active during this interval. Define the relative fairness bound (RFB) as the maximum of  $RF(t_1, t_2)$  over all possible time intervals  $(t_1, t_2)$ .

**Definition 2.** Define  $m$  as the size in flits of the largest packet that is actually served during the execution of a scheduling algorithm.

**Definition 3.** Define  $M$  as the size in flits of the largest packet that may potentially arrive during the execution of a scheduling algorithm. Note that  $M \geq m$ .

**Lemma 1.** For any flow  $i$  and round  $r$  in the execution of an ERR scheduling discipline,  $0 \leq SC_i(r) \leq m - 1$ .

**Proof.** The lower bound on  $SC_i(r)$  in the expression of the lemma is obvious since the ERR algorithm always schedules at least as many flits as  $A_i(r)$  during round  $r$ . The only exception is when the queue for flow  $i$  becomes empty in round  $r$ , in which case the surplus count of the flow is reset to 0.

The ERR algorithm never begins dequeuing a new packet in a flow after the number of flits sent in a round  $r$  is equal to or more than the allowance  $A_i(r)$ . Thus, the lowest value of  $(Sent_i(r) - A_i(r))$  at which a new packet transmission may begin is 1 and this will be the last packet transmitted by the flow during this round. Since the size of this packet can be no greater than  $m$ , from (1), the upper bound in the expression of the lemma is proven.  $\square$

The following corollary follows directly from Lemma 1.

**Corollary 1.** In any round  $r$ ,  $0 \leq MaxSC(r) \leq m - 1$ .

**Theorem 2.** Given  $n$  consecutive rounds starting from round  $k$ , during which flow  $i$  is active, the bounds on the total number of flits,  $N$ , transmitted by flow  $i$  are given by

$$n + \sum_{r=k-1}^{k+n-2} MaxSC(r) - (m - 1) \leq N \leq n + \sum_{r=k-1}^{k+n-2} MaxSC(r) + (m - 1).$$

**Proof.** Substituting for  $A_i(r)$  using (2) into (1), we get,

$$Sent_i(r) = 1 + MaxSC(r - 1) - SC_i(r - 1) + SC_i(r). \quad (4)$$

Summing the LHS in (4) above for  $r = k$  to  $r = k + n - 1$ , we get  $N$ , the total number of flits sent during the  $n$  consecutive rounds under consideration. Equating this to the summation of the RHS in (4) for  $r = k$  to  $r = k + n - 1$ ,

$$N = n + \sum_{r=k-1}^{k+n-2} MaxSC(r) + SC_i(k + n - 1) - SC_i(k - 1). \quad (5)$$

Using Lemma 1,  $0 \leq SC_i(k + n - 1) \leq m - 1$ , and  $0 \leq SC_i(k - 1) \leq m - 1$ . The result of the theorem is readily obtained by substituting for these bounds on  $SC_i(k - 1)$  and on  $SC_i(k + n - 1)$  in (5).  $\square$

We now proceed to prove the bound on the relative fairness of the ERR scheduling discipline. Note that the relative fairness bound, RFB, is defined taking into consideration all possible intervals of time  $(t_1, t_2)$ . In the following, we prove that a tight upper bound can be obtained considering only a subset of all possible time intervals. This subset is the set of all time intervals bounded by time instants that coincide with the beginning or the end of the service opportunity of flows.

**Definition 4.** Let  $\mathbf{T}$  be the set of all time instants during an execution of the ERR algorithm. Define  $\mathbf{T}_s$  as the set of all time instants at which the scheduler ends serving one flow and begins serving another. Define  $F(t)$ , for  $t \notin \mathbf{T}_s$ , as the flow which is being served at time instant  $t$ . For  $t \in \mathbf{T}_s$ , we define  $F(t)$  as the flow just about to begin service.

The following lemma allows us to prove an upper bound on the relative fairness, stated in Theorem 3, considering only the time intervals  $(t_1, t_2)$ , where  $t_1, t_2 \in \mathbf{T}_s$ .

**Lemma 2.**  $RFB = \max_{t_1, t_2 \in \mathbf{T}_s} RF(t_1, t_2)$ .

**Proof.** This lemma is proven if, for any  $t_1, t_2 \in \mathbf{T}$ , we can find  $t'_1, t'_2 \in \mathbf{T}_s$ , such that  $RF(t'_1, t'_2) \geq RF(t_1, t_2)$ .

Consider any two active flows  $i$  and  $j$  during the time interval between  $t_1$  and  $t_2$ , where  $t_1, t_2 \in \mathbf{T}$ . Without loss of generality, assume that, during this time interval, more flits have been scheduled from flow  $i$  than from flow  $j$ . By appropriately choosing  $t'_1$  as the time instant at either the beginning or the end of the service opportunity given to  $F(t_1)$  at time  $t_1$ , one may verify that  $RF(t'_1, t_2) \geq RF(t_1, t_2)$ . Similarly, an appropriate choice of  $t'_2$ , as either the beginning or the ending instant of the service opportunity given to  $F(t_2)$  at time  $t_2$ , can lead to  $RF(t'_1, t'_2) \geq RF(t_1, t_2)$ .  $\square$

**Theorem 3.** For any execution of the ERR scheduling discipline,  $RFB < 3m$ .

**Proof.** By the statement of Lemma 2, we need to only consider all time intervals bounded by time instants that coincide with the starting or ending of service to a flow. We therefore prove the statement of the theorem using the time interval between instants  $t_1$  and  $t_2$ , where both  $t_1$  and  $t_2$  belong to  $\mathbf{T}_s$ .

Consider any two flows  $i$  and  $j$  that are active in the time interval between  $t_1$  and  $t_2$ . From the algorithm in Fig. 1, it follows that after flow  $i$  receives service, it is added to the tail end of the *ActiveList*. The ERR scheduler then visits flow  $j$ , which is served before flow  $i$  receives service again. Thus, in between any two consecutive service opportunities given to flow  $i$ , flow  $j$  receives exactly one service opportunity. Hence, if  $n_i$  and  $n_j$  denote the total round-robin opportunities received by flows  $i$  and  $j$ , respectively, in the time interval  $(t_1, t_2)$ , then  $|n_i - n_j| \leq 1$ .

Let  $r(t)$  denote the round in progress at time instant  $t$ . Also note that the time instant  $t_1$  may be such that the service opportunity received by one of the two flows in round  $r(t_1)$  may not be a part of interval  $(t_1, t_2)$ . Thus, the first time that the scheduler visits this flow in the interval under consideration would be in the round following  $r(t_1)$ . Consequently, if  $r_i$  and  $r_j$  denote the rounds in which flows  $i$  and  $j$  receive service for the first time in the interval  $(t_1, t_2)$ , respectively, then  $|r_i - r_j| \leq 1$ .

Without loss of generality, we can assume that in the interval  $(t_1, t_2)$ , flow  $i$  starts receiving service before flow  $j$ . Thus,

$$r_j \leq r_i + 1 \quad \text{and} \quad n_i \leq n_j + 1. \quad (6)$$

From Theorem 2, for flow  $i$ ,

$$Sent_i(t_1, t_2) \leq n_i + \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) + (m-1). \quad (7)$$

For flow  $j$ ,

$$n_j + \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k) - (m-1) \leq Sent_j(t_1, t_2). \quad (8)$$

Combining (7) and (8) and using (6), we get

$$Sent_i(t_1, t_2) - Sent_j(t_1, t_2) \leq 1 + 2(m-1) + \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) - \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k). \quad (9)$$

Let us now consider the quantity  $D$  given by

$$D = \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) - \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k).$$

We now compute  $D$  for each of the four possible cases.

Case 1 ( $r_i = r_j$ ,  $n_i = n_j$ ):

$$D = 0.$$

Case 2 ( $r_i = r_j$ ,  $n_i = n_j + 1$ ):

$$D = MaxSC(r_i + n_i - 2).$$

Case 3 ( $r_i = r_j - 1$ ,  $n_i = n_j$ ):

$$D = MaxSC(r_i - 1) - MaxSC(r_i + n_i - 1).$$

Case 4 ( $r_i = r_j - 1$ ,  $n_i = n_j + 1$ ):

$$D = MaxSC(r_i - 1).$$

Using Corollary 1, it is readily verified that, in each of the above four cases,  $D < m$ . Substituting in (9), the statement of the theorem is proven.  $\square$

Note that for the Weighted ERR scheduler, the relative fairness over an interval  $(t_1, t_2)$  is defined as

$$RF(t_1, t_2) = \max_{i,j} \left| \frac{Sent_i(t_1, t_2)}{w_i} - \frac{Sent_j(t_1, t_2)}{w_j} \right|, \quad (10)$$

where  $i$  and  $j$  are flows that are active during the interval  $(t_1, t_2)$  [17], [24].

It can be verified that Theorem 3 can also be proven for the Weighted ERR scheduler using (3) in place of (2) in the proof above.

In comparison to a relative fairness bound of  $3m$  for ERR, both DRR and SRR have a relative fairness bound of  $M + 2m$ , where  $M$  is the size of the largest packet that may *potentially* arrive during the lifetime of the execution of the scheduling discipline. Recall that  $m$  is the size of the largest packet that *actually* arrives during the execution of the scheduler. In most networks, including the Internet, the vast majority of the packets in the traffic are of much smaller size than the maximum possible size of a packet [27]. The value of  $m$  in the expression for the relative fairness, especially over short intervals of time, is likely to be much smaller than  $M$ . The fairness achieved by ERR, thus, is always equal to or better than that achieved by DRR or SRR.

### 4.3 Start-Up Latency

The performance of scheduling disciplines can be compared in terms of at least two quantities: The time it takes for the first packet of a flow to be served completely and the average queuing delay of packets in the flows. We denote the first quantity as the *Start-Up Latency Bound* and define it as the maximum length of time between the instant the first packet of a new flow arrives in its queue and the instant the last flit of this packet is scheduled. A good bound on the start-up latency guarantees that the scheduler will serve a new flow or a flow that has just become active within an acceptable and finite amount of time. This bound is particularly relevant to scheduling of control packets, which typically do not arrive as part of a traffic stream and which have a low tolerance for delays. We define the *queuing delay* of a packet as the length of time between the instant it is placed in the queue for scheduling and the instant its last flit is scheduled. In this section, we analyze the start-up latency of ERR in comparison with other scheduling algorithms. A simulation study of queuing delays is presented in Section 5.

As in the rest of this section, we present our analysis assuming equal weights for all the flows. Therefore, given  $n$  active flows, the share of the bandwidth allowed to each flow is  $r/n$ , where  $r$  is the capacity of the output link. The following theorem proves the start-up latency bound for ERR.

**Theorem 4.** *During an execution of the ERR scheduling discipline serving  $n$  active flows at a link of maximum rate  $r$ , the start-up latency,  $S_{ERR}$ , of a newly active flow has an upper bound given by*

$$S_{ERR} \leq \frac{(2m-1)n + m}{r}.$$

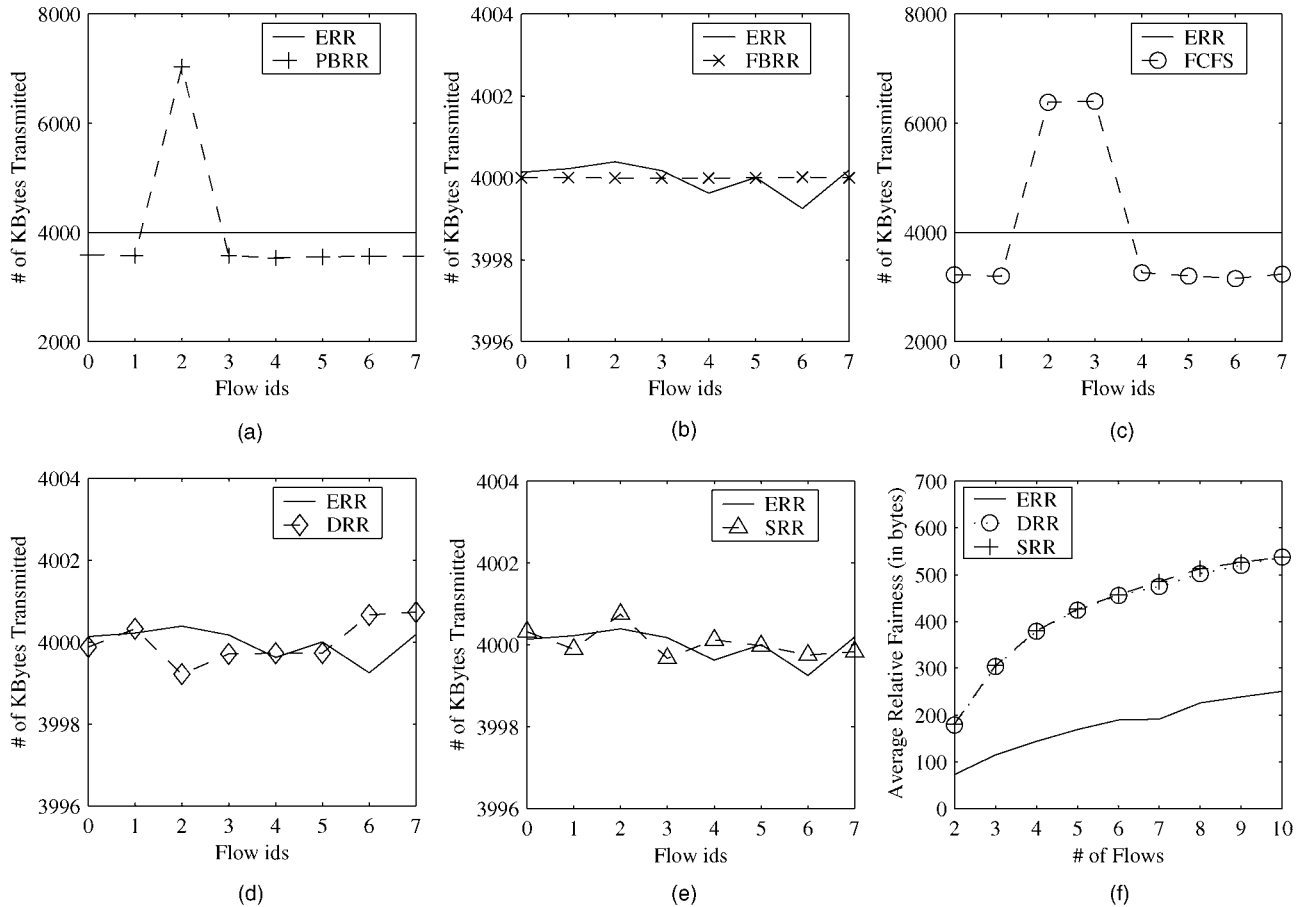


Fig. 4. Simulation results on fairness comparisons.

**Proof.** When the first packet of a newly active flow arrives, the flow is placed at the tail of the *ActiveList* and is served after all the  $n$  previously active flows are served. Using Lemma 1 and Corollary 1 in (4), we get

$$Sent_i(r) \leq 2m - 1.$$

Therefore, a maximum of  $(2m - 1)n$  flits may be served before the scheduler begins serving the newly active flow. Noting that the maximum size of the first packet of the newly active flow is  $m$ , the statement of the theorem is readily proven.  $\square$

The start-up latency bound of ERR is lower than that of DRR, which has a start-up latency,  $S_{DRR}$ , bounded above as follows:

$$S_{DRR} \leq \frac{(M + m - 1)n + m}{r}.$$

It can be readily proven that the start-up latency bound of SRR is equal to that of DRR.

## 5 SIMULATION RESULTS

In this section, we present simulation results on the performance and the fairness properties of the ERR scheduler. The ERR algorithm is compared with other packet-by-packet scheduling algorithms of equivalent work complexity

such as DRR, SRR, FCFS, FBRR, and PBRR. DRR and SRR are the scheduling disciplines that come closest to ERR in terms of being both fair and efficient and, therefore, a majority of our focus is on ERR in comparison with DRR and SRR.

We compare the fairness of scheduling disciplines by plotting for a given interval, during which all the flows are active, the number of bytes scheduled from each of the different flows. Figs. 4a, 4b, 4c, 4d, 4e, and 4f show the results of our simulation experiments on fairness.

For the results in Figs. 4a, 4b, 4c, 4d, and 4e, we simulate eight flows with flow ids from 0 to 7. We collect results for a period of four million cycles, during which we ensure that all the flows are active. The arrival rate in terms of packets per second into the queue corresponding to flow 3 is twice the rate of other flows. Also, the packet lengths are uniformly distributed between one and 64 flits for all the flows, except flow 2. Packets belonging to flow 2 have lengths uniformly distributed between 1 and 128 flits. Note that, in this experiment, the maximum possible packet size,  $M$ , is equal to 128, while the largest packet that actually arrives is also 128 since the number of cycles in the simulation experiment is large. We assume a flit size of 8 bytes and that the scheduler dequeues one flit from one of the queues in each cycle.

Fig. 4a demonstrates that ERR is fair in terms of throughput achieved by the different flows, while PBRR is

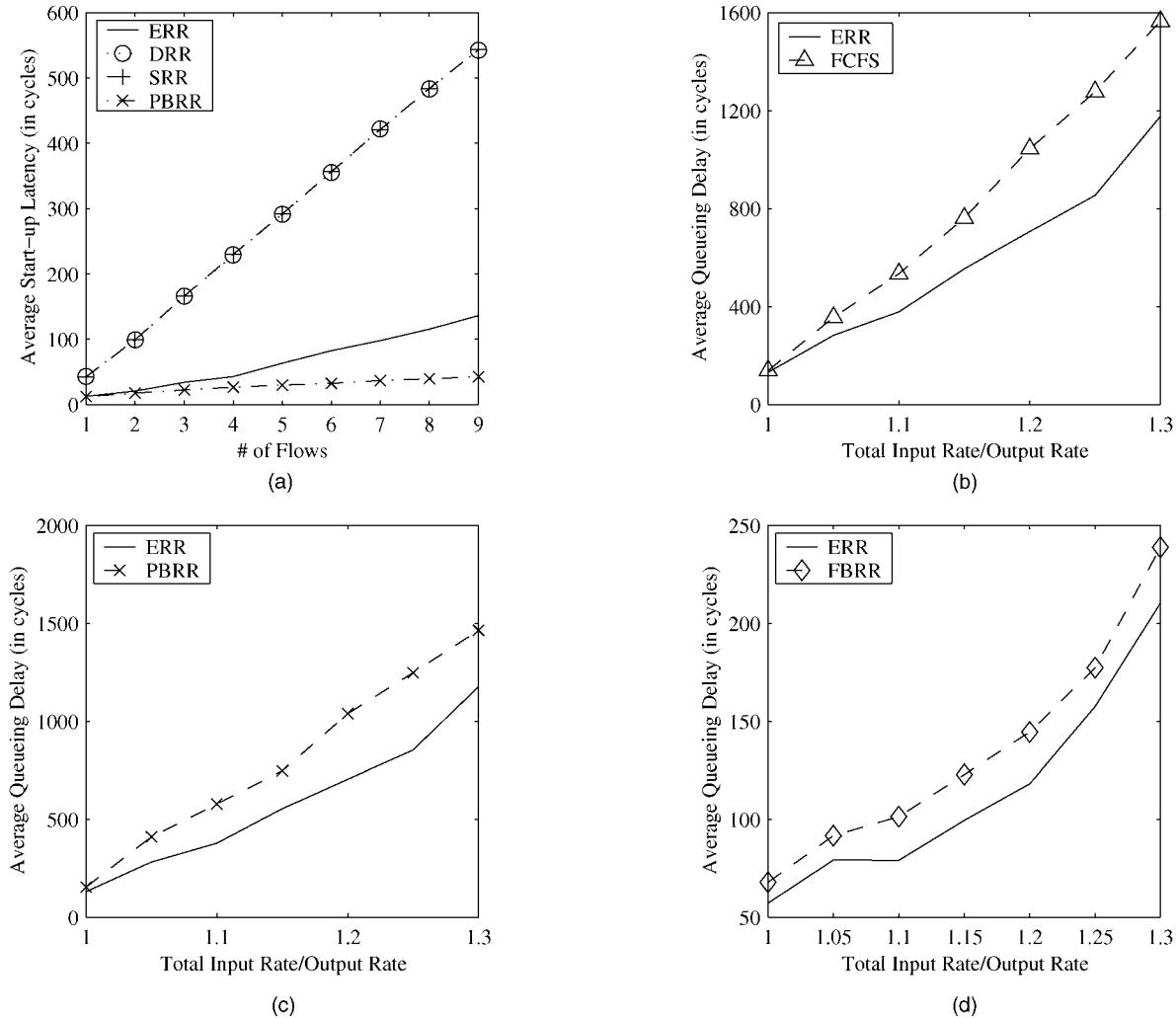


Fig. 5. Simulation results on performance comparisons.

not. As shown in the figure, with PBRR, the flow sending larger packets (flow 2) gains an unfair fraction of the bandwidth.

Fig. 4b shows that ERR, however, is not as fair as FBRR. This is expected since, with a scheduling granularity of one flit, FBRR is the fairest algorithm in terms of throughput achieved by the flows. Note that, as stated by Theorem 3 for ERR, the maximum difference between the number of bytes served from different flows is less than  $3 \times 128 \times 8$  bytes or 3 KBytes.

Fig. 4c compares FCFS with ERR. As expected, in FCFS, flows which send at twice the rate or which send packets of twice the lengths manage to steal approximately twice the bandwidth. ERR, on the other hand, maintains fairness among the flows independent of packet lengths or injection rates.

Fig. 4d compares ERR with DRR and shows that the two scheduling disciplines, for uniformly distributed packet lengths, are comparable in fairness. Fig. 4e similarly shows that ERR and SRR are comparable in fairness.

Fig. 4f shows the result of a simulation in which packet lengths in all the flows are exponentially distributed between one to 64 flits, with the parameter  $\lambda = 0.2$ . Recall

that the relative fairness bound for both DRR and SRR is  $M + 2m$ , whereas that for ERR is  $3m$ . This difference in fairness between the scheduling disciplines is best highlighted by a packet length distribution in which the larger size packets are less likely to appear than smaller size packets, such as when the lengths are exponentially distributed. We compute the average relative fairness achieved by the ERR, DRR, and SRR scheduling disciplines over 10,000 randomly chosen intervals during a period of four million cycles. Fig. 4f demonstrates that when larger size packets are less likely than smaller size packets, which is often the case in many real networks including interconnection networks for parallel systems and the Internet [27], ERR achieves better fairness than DRR or SRR.

Figs. 5a, 5b, 5c, and 5d show the results of our next set of experiments on the performance of the ERR scheduler. In Fig. 5a, we compare the start-up latencies of ERR, DRR, SRR, and PBRR. We do not plot results for FCFS in this figure since, at an FCFS scheduler, the start-up latency is determined by the number of packets that are already waiting in the queue and is thus bounded only by the size

TABLE 1  
Comparison of Fair Scheduling Algorithms

Scheduling Discipline	Complexity	Fairness	Start-up Latency Bound	Applicable to Wormhole Networks
GPS [16]	-	0	$\frac{mn}{r} + \frac{m}{r}$	-
Packet-Based Round Robin [2]	O(1)	$\infty$	$\frac{mn}{r} + \frac{m}{r}$	✓
First-Come-First-Served	O(1)	$\infty$	$\infty$	✓
Weighted Fair Queueing [14]	O(log n)	O(n)	$\frac{mn}{r} + \frac{m}{r}$	-
Self-Clocked Fair Queueing [17]	O(log n)	2m	$\frac{2mn}{r} + \frac{m}{r}$	-
Worst-Case Fair Weighted Fair Queueing [21]	O(log n)	2m	$\frac{mn}{r} + \frac{m}{r}$	-
Deficit Round Robin [22]	O(1)	M + 2m	$\frac{(M + m - 1)n}{r} + \frac{m}{r}$	-
Surplus Round Robin [18-20]	O(1)	M + 2m	$\frac{(M + m - 1)n}{r} + \frac{m}{r}$	-
Elastic Round Robin	O(1)	3m	$\frac{(2m - 1)n}{r} + \frac{m}{r}$	✓

of the buffers implementing the queues. Recall that the latency bound for ERR is  $\frac{(2m-1)n+m}{r}$ , whereas that for DRR or SRR is  $\frac{(M+m-1)n+m}{r}$ . To best illustrate this difference, we use packet lengths that are exponentially distributed in the range between 1 to 64 flits, with the parameter  $\lambda = 0.2$ . The start-up latency depends on the number of active flows before the arrival of the first packet of the new flow, and therefore, we produce simulation results using different numbers of flows ranging from  $n = 1$  to 9. While these flows are kept active throughout the duration of this simulation, we simulate another flow that alternates between active and inactive periods. The beginning and the end of active periods is chosen randomly. For each active period of the flow, we measure the start-up latency as the number of cycles between the arrival of the first packet of the active period of the flow and the scheduling of the last flit of this packet. For each of the different numbers of flows, Fig. 5a plots the average start-up latency over 1,000 active periods and shows that ERR has better start-up latencies than both DRR and SRR. Note that, even though PBRR has lower start-up latencies, its relative

fairness bound is infinity and cannot be used for achieving fairness.

Figs. 5b and 5c show the average queuing delay of packets arriving at an ERR scheduler in comparison to FCFS and PBRR. In comparing the queuing delays experienced by packets with different scheduling disciplines, it only makes sense to consider flows that are active. If the sum of the rates at which the packets are arriving in the flows is greater than the maximum possible output rate, the delays will eventually reach infinity because of persistent congestion, rendering a meaningful comparison impossible. On the other hand, if there is no congestion at all, queuing delays are all nearly zero. For an appropriate and realistic comparison, therefore, we create transient periods of congestion during which the sum of the input rates is higher than the output bandwidth. In our simulations, these transient periods of congestion last 10,000 cycles, after which we halt all injection of packets into the queues and continue simulation until all the queues are empty. The figures are plotted for the average queuing delay of a packet against the intensity of the transient congestion (measured by the ratio of the sum of the input rates to the maximum possible output rate). We use four flows in the simulations in Figs. 5b, and 5c and, as before, packet arrival rate in the queue for flow 3 is twice that of other flows. Also, as before, the

packet lengths are uniformly distributed from one to 64 flits, except for flow 2, in which the packet lengths are uniformly distributed between one and 128 flits.

Fig. 5b shows that ERR has a better average queuing delay than FCFS when packet sizes and input rates of the flows can be different. A well-known result in queuing theory states that, for any given flow, if a scheduling discipline achieves better average delay than FCFS, it comes at the expense of increasing the delays of some other flows [28]. The better overall average delay of ERR is achieved at the expense of flows sending at twice the rate or flows sending larger packets. FCFS, on the other hand, would have given these flows higher bandwidths and, therefore, lower delays, while increasing the delays of all other flows. Similarly, ERR has a much better average queuing delay than PBRR, as shown in Fig. 5c.

Fig. 5d shows the average queuing delay of packets obtained with ERR and FBRR. Note that in ERR, once a packet begins transmission, the entire packet is completely scheduled before another packet is allowed to begin transmission. This reduces the average delay of packets in comparison to FBRR, which, by serving packets flit-by-flit, uniformly increases the delays experienced by all the flows. In general, for this reason, packet-by-packet round-robin schedulers have the potential to improve average delays in comparison to FBRR. This additional advantage of ERR over FBRR is shown in Fig. 5d.

DRR, SRR, and ERR have finite and close relative fairness bounds and, therefore, the differences between these disciplines in terms of the exact time a packet gets scheduled, is bounded and finite. Therefore, the differences between the average queuing delays with these two disciplines remain small in comparison to the large queuing delays experienced during congestion. Therefore, the queuing delays among ERR, DRR, and SRR are not compared.

## 6 SUMMARY AND CONCLUSION

In this paper, we have presented a novel scheduling discipline called *Elastic Round Robin* (ERR), which is simple, fair, and efficient with a low start-up latency. In addition, it satisfies the unique requirement imposed by wormhole switching on fair scheduling disciplines. We have shown that the work complexity of ERR is  $O(1)$  and, therefore, can be easily implemented in networks with large numbers of flows. In comparison to other scheduling disciplines of similar efficiency, such as Deficit Round Robin (DRR) and Surplus Round Robin (SRR), ERR has better fairness properties, as well as a lower start-up latency bound. Table 1 summarizes the work complexity, fairness, and the start-up latency of several scheduling algorithms. In the column on work complexity,  $n$  is the total number of active flows. In the column on start-up latency,  $n$  is the number of active flows at the instant before the start of the new flow. The peak rate of the output link is denoted by  $r$ . Among scheduling disciplines of comparable efficiency, DRR and SRR come closest to ERR in fairness. However, neither DRR nor SRR is ideally suitable for use in wormhole networks, where the length of time a packet occupies the link is not known before a decision to transmit the packet is

made. On the other hand, ERR can be readily used in wormhole networks, in addition to being perfectly suitable for achieving fair scheduling in Internet routers.

Finally, it is worthwhile to note that ERR can be used in a wide variety of contexts whenever there is a shared resource that needs to be allocated fairly among multiple requesting entities. For example, one may define a flow as the stream of packets belonging to the same virtual channel, in which case, ERR can be used to achieve fairness among virtual channels in the forwarding of flits to the output link, while also achieving lower delays as shown in Fig. 5d. ERR can also be used in the forwarding of packets from the input buffers to the output buffers of switching elements in networks. ERR can also be a solution in token ring networks, where the bandwidth of the ring has to be shared among multiple sources. Similarly, ERR can be used to efficiently arbitrate access to a busy shared bus. The lower start-up latency of ERR among similarly efficient algorithms is especially useful here in improving the latency of short control messages. Finally, ERR is particularly relevant to the problem of job scheduling in operating systems, where multiple processes are competing for limited CPU cycles.

## ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation CAREER Award CCR-9984161 and US Air Force Contract F30602-00-2-0501.

## REFERENCES

- [1] H. Sethu, C.B. Stunkel, and R.F. Stucke, "IBM RS/6000 SP Large System Interconnection Network Topologies," *Proc. Int'l Conf. Parallel Processing*, Aug. 1998.
- [2] J. Nagle, "On Packet Switches with Infinite Storage," *IEEE Trans. Comm.*, vol. 35, no. 4, Apr. 1987.
- [3] W.J. Dally and C.L. Seitz, "The Torus Routing Chip," *J. Distributed Computing*, vol. 1, no. 3, pp. 187-196, Oct. 1986.
- [4] C.B. Stunkel, "The SP2 High-Performance Switch," *IBM Systems J.*, vol. 34, no. 2, pp. 185-204, Feb. 1995.
- [5] J. Beecroft, M. Homewood, and M. McLaren, "Meiko CS-2 Interconnect Elan-Elite Design," *Parallel Computing*, vol. 20, no. 10-11, pp. 1627-1638, Nov. 1994.
- [6] Intel Corporation, *Paragon XP/S Product Overview*. 1991.
- [7] Cray Research, Inc., *Cray T3D System Architecture*. 1993.
- [8] ANSI, Inc., *High-Performance Parallel Interface-6400 Mb/s Physical Layer (HIPPI-6400-PH)*. June 1999.
- [9] N.J. Boden et al., "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, pp. 29-35, Feb. 1995.
- [10] Y. Tamir and G.L. Frazier, "Dynamically Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 725-737, June 1992.
- [11] J. Ding and L.N. Bhuyan, "Evaluation of Multi-Queue Buffered Multistage Interconnection Networks under Uniform and Non-Uniform Traffic Patterns," *Int'l J. Systems Science*, vol. 28, no. 11, 1997.
- [12] W.J. Dally, "Virtual Channel Flow Control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 3, pp. 194-205, Mar. 1992.
- [13] H. Sethu, H. Shi, S.S. Kanhere, and A.B. Parekh, "A Round-Robin Scheduling Strategy for Reduced Delays in Wormhole Switches with Virtual Lanes," *Proc. Int'l Conf. Comm. in Computing*, June 2000.
- [14] A. Demers, S. Keshav, and S. Shenker, "Design and Analysis of a Fair Queuing Algorithm," *Proc. ACM SIGCOMM*, pp. 1-12, Sept. 1989.
- [15] S. Keshav, "On the Efficient Implementation of Fair Queuing," *J. Internetworking Research and Experience*, vol. 2, no. 3, pp. 3-26, Sept. 1990.

- [16] A.K. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control—The Single Node Case," *Proc. IEEE INFOCOM*, pp. 915-924, May 1992.
- [17] S.J. Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," *Proc. IEEE INFOCOM*, pp. 636-646, June 1994.
- [18] S. Floyd and V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks," *IEEE Trans. Networking*, vol. 3, no. 4, pp. 365-386, Aug. 1995.
- [19] S. Floyd, "Notes on Class-Based-Queueing and Guaranteed Service," Unpublished Notes: <http://www.aciri.org/floyd/cbq.html>, July 1995.
- [20] G. Parulkar, H. Adishesu, and G. Varghese, "A Reliable and Scalable Striping Protocol," *Proc. ACM SIGCOMM*, pp. 131-141, Aug. 1996.
- [21] J.C.R. Bennett and H. Zhang, "WF<sup>2</sup>Q: Worst-Case Fair Weighted Fair Queueing," *Proc. IEEE INFOCOM*, pp. 120-128, Mar. 1996.
- [22] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round-Robin," *IEEE Trans. Networking*, vol. 4, no. 3, pp. 375-385, June 1996.
- [23] P. Goyal, H.M. Vin, and H. Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *IEEE Trans. Networking*, vol. 5, no. 5, pp. 690-704, Oct. 1997.
- [24] S. Keshav, *An Engineering Approach to Computer Networks*. Reading, Mass.: Addison-Wesley, 1997.
- [25] J.A. Cobb, M.G. Gouda, and A. El-Nahas, "Time-Shift Scheduling-Fair Scheduling of Flows in High-Speed Networks," *IEEE Trans. Networking*, vol. 6, no. 3, pp. 274-285, June 1998.
- [26] D. Stiliadis and A. Varma, "Efficient Fair Queueing Algorithms for Packet-Switched Networks," *IEEE Trans. Networking*, vol. 6, no. 2, pp. 175-185, Apr. 1998.
- [27] K. Thompson, G.J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10-23, Nov./Dec. 1997.
- [28] L. Kleinrock, *Queueing Systems, Volume 2: Computer Applications*. New York: Wiley Interscience, 1975.



**Salil S. Kanhere** (S '00) received the BE degree in electrical engineering from the University of Bombay, Bombay, India, in June 1998. He is currently pursuing the PhD degree in the Department of Electrical and Computer Engineering at Drexel University, Philadelphia, Pennsylvania. His research interests include design, analysis, and implementation of scheduling disciplines in packet switched networks. He is a student member of the IEEE.



**Harish Sethu** (M '99) obtained the BTech degree in electronics and communication engineering from Indian Institute of Technology (IIT), Chennai, in 1988. He received the PhD degree in electrical engineering from Lehigh University in 1992. He worked as an Advisory Development Engineer/Scientist at IBM Corporation for six years, during which he contributed to the hardware, software, and system-level design of three generations of the RS/6000 SP family of high-performance parallel computers. He joined the Department of Electrical and Computer Engineering at Drexel University in 1998 as an assistant professor. He has been awarded three US patents. He is also a recipient of the US National Science Foundation CAREER award. His current research interests include quality-of-service in computer networks and the architecture of switches and routers. He is a member of the IEEE.



**Alpa B. Parekh** received the BE degree in electrical engineering from the University of Bombay, Bombay, India, in June 1998 and the MS degree in electrical engineering from Drexel University, Philadelphia, PA in June 2000. She currently works as a software engineer in the Networks and Protocol Development Department at Lockheed Martin Global Telecommunications in Clarksburg, MD.

► For more information on this or any computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.