Proceedings of the International Parallel and Distributed Processing Symposium
Cancun, Mexico, May 1–5, 2000

Fair and Efficient Packet Scheduling in Wormhole Networks

Salil S. Kanhere, Alpa B. Parekh and Harish Sethu
Department of ECE, Drexel University
3141 Chestnut Street, Philadelphia, PA 19104-2875.
{*salil, alpa, sethu*}*@ece.drexel.edu*

Abstract

Most switch architectures for parallel systems are designed to eliminate only the worst kinds of unfairness such as starvation scenarios in which packets belonging to one traffic flow may not make forward progress for an indefinite period of time. However, stricter fairness can lead to a more predictable and better performance, in addition to improving isolation between traffic belonging to different users. This paper presents a new easily implementable scheduling discipline, called Elastic Round Robin (ERR), *for the unique requirements of wormhole switching, popular in interconnection networks for parallel systems. Despite the constraints of wormhole switching imposed on the design, our scheduling discipline is at least as efficient as other scheduling disciplines, and more fair than scheduling disciplines of comparable efficiency proposed for any other kind of network, including the Internet. We prove that the work complexity of ERR is O(1) with respect to the number of flows. We analytically prove the fairness properties of ERR, and show that its relative fairness measure has an upper bound of $3m$, where m is the size of the largest packet that actually arrives during an execution of ERR. Finally, we present simulation results comparing the fairness and performance characteristics of ERR with other scheduling disciplines of comparable efficiency.*

1. Introduction

Almost all interconnection networks, both direct and indirect, are constructed out of switches connected together in a certain topology. Wormhole switching is a popular switching technique used in the implementation of switches in interconnection networks for parallel systems, and more recently, in system area networks [1–3, 10, 16, 18]. This paper considers the problem of fair and efficient scheduling of packets in wormhole networks for parallel systems. Even though the solution presented in this paper is designed for the unique requirements of wormhole switches, it may also be applied to wide-area networks such as the Internet.

Most switch architectures for parallel systems are designed to 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. However, stricter fairness is essential for good performance, since unfair treatment of some traffic flows in the network can easily lead to unnecessary bottlenecks. Fairness is also essential for good performance with compilers that take into account the predicted communication delays in the network. In addition, the increasing use of multi-user environments in parallel systems [16], with the interconnection network shared by several users at the same time, provides further motivation for fairness and isolation within the network between traffic generated by different users. This isolation between users in a multi-user system is especially required for repeatable performance necessary for benchmark applications. Fair scheduling disciplines in switch architectures are an essential first step in protecting one user's traffic from another and thus providing a firewall. Finally, fair scheduling algorithms have especially gained in importance with the increasing demand for customer-specific differentiated services in the Internet [14], or in parallel systems that are used, for example, as video-servers.

Implementing fair scheduling disciplines in wormhole switches poses certain unique challenges, which we describe in the following paragraphs. Wormhole switching is distinguished by the fact that the granularity of flow control in a wormhole network can be smaller than a packet [5, 8]. 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 network reads the information in the head flit and directs it to the next switch or end-system device on its path. The rest of the flits of the packet in a wormhole switch follow the path of the head flit.

Consider a wormhole switch with several input and output queues, with packets in the input queues ready for transmission to one of the output queues. 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 [7, 11, 19]. Consider a certain packet *P*, with a head flit that arrives at an input queue of a wormhole switch and is ready to be routed to an output queue. Once the head flit has been routed to the output queue, no flits from any other packet can be routed to this output queue, until all of the remaining flits of packet *P* are routed. In wormhole switches with virtual channels [4], one typically has as many output queues as there are virtual channels associated with each of the output links. Since each flit is marked by the virtual channel it belongs to, in scheduling flits to the output link from these output queues, it is not necessary to schedule all flits belonging to a packet before a flit from another virtual channel is scheduled. However, even in such switches, in scheduling entry into the output queues from the various input queues, all flits of a packet have to be scheduled before a flit from another packet enters the same output queue. Now, downstream congestion can thwart further progress of flits belonging to packet *P* for an unpredictable amount of time. A packet of length L bytes, scheduled for forwarding to an output queue feeding a link of capacity C bytes/second, may take more than L/C seconds for transmission. In other words, the length of a packet cannot singly determine the length of time it takes to dequeue a packet while it blocks other flows from access to the output queue.

In wormhole networks, as in any other kind of networks, the resource that should be fairly allocated is, for example, access to the output queue or link. No matter how many flits a flow sends during the time that it blocks access to the output link for other flows, the fairness in such a case should be over the length of time each flow is allowed to block other flows from accessing the output link. This length of time, in wormhole networks, is not dependent upon the length of a packet that is being transmitted but instead on the downstream congestion which can be hard to predict without complex feedback mechanisms. Therefore, scheduling of packets in wormhole networks should be based on the actual length of time that a packet takes to be dequeued, which quantity 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 dequeuing a packet without knowledge of how long it will take to dequeue the packet. Such a requirement on the scheduling discipline is also essential in networks where packet delimiters are the only indication of the beginning and the end of packets, with no length fields in the packet headers. The Elastic Round Robin (ERR) scheduling discipline presented here is designed to address this unique requirement.

Besides scheduling packets from input queues to output queues in wormhole switches, the ERR algorithm can actually also be used for achieving low average delay in the fair scheduling of packets to the output link from output queues belonging to various virtual channels. The ERR algorithm may also be implemented in Internet routers for fair scheduling of various flows of traffic with each flow corresponding to a source-destination pair. Because of this wider applicability of our solution, and so that this work can be readily understood and used in a broad 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 on an output link or to another queue. 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, we require that the scheduling algorithm not know the length of a packet until it has completely dequeued the packet. For wormhole networks, references to the length of the packet in the algorithm may be replaced by length of time it takes to dequeue the packet. In this paper, we use a flit as the smallest piece of a packet that can be independently scheduled, and we measure the length of a packet in terms of flits.

Despite the constraints of wormhole switching imposed on the design, the ERR algorithm is at least as efficient as other scheduling disciplines, and more fair than scheduling disciplines of comparably high efficiency proposed for any other kind of network, including the Internet. We measure efficiency by the order of the work complexity associated with the enqueuing and dequeuing operations, with respect to n, the number of flows. We prove that this work complexity is O(1), equal to or better than other scheduling disciplines proposed. We measure fairness using a well-known and widely used relative fairness measure first proposed in [9]. The relative fairness measure of our ERR algorithm, achieves an upper bound of $3m$, where m is the size of the largest packet that *actually* arrives during the execution of ERR. The closest comparison that can be made is to the Deficit Round Robin (DRR) scheduler [17], which has an O(1) work complexity and a relative fairness measure of $Max + 2m$, where Max is the size of the largest packet that may *potentially* arrive during the lifetime of the execution of the scheduling discipline. DRR, however, for reasons described earlier, is not suitable for wormhole networks since it requires the knowledge of the size of a packet before dequeuing it.

Section 2 describes relevant previous research in the design of scheduling algorithms. Section 3 presents the Elastic Round Robin scheduling algorithm, and explains the rationale behind it. Section 4 analytically proves the order of work complexity and the fairness properties of this scheduling discipline. Section 5 presents some simulation results on the fairness and performance characteristics of this algorithm in comparison with other algorithms of similar efficiency. Section 6 concludes the paper.

2. Background and Previous Work

Consider several flows, with flits belonging to packets waiting in the respective queues to be forwarded to another queue or an output link. One scheduling technique would be to use a pure *Flit-Based Round Robin* (FBRR) scheme, in which the scheduler visits each flow's queue in a round-robin fashion, and transmits one flit from each queue. This scheme is only possible in wormhole networks when each flit is tagged with a flow id, such as when each flow represents a virtual channel [4]. This scheme is very fair among the flows in terms of the number of flits scheduled from each of the virtual channels during any time interval. However, it cannot be used in other contexts such as for scheduling packets from input queues to output queues in a wormhole switch as described in Section 1. An alternate technique would be *Packet-Based Round Robin* (PBRR), in which the scheduler visits each of the queues in a round-robin fashion, and transmits an entire packet from a queue before beginning transmission from another queue. These techniques and a number of its variations have been analyzed in [15] for their performance characteristics, but not for their fairness properties. The PBRR scheduling discipline, for example, is not fair among the flows when the packet sizes in the different flows are not equal. Consequently, flows sending longer packets use up an unfairly high fraction of the available transmission bandwidth.

Most wormhole switches used today for system area networks, employ the First-Come-First-Served (FCFS), FBRR or PBRR scheduling disciplines in the various functional units of a switch. In FCFS, as the name implies, the packets are scheduled in the order of their arrival times. The difficulty with FCFS is that it does not provide adequate protection from a bursty source that may suddenly send 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 a flow from another source. As long as a source is demanding bandwidth within its rightful share, the delays experienced by packets from this source should not be affected by other traffic in the network.

The classic notion of fairness in the allocation of a resource among n requesting entities with equal rights to the resource but unequal demands, is as follows:

- 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) is an unimplementable but ideal scheduling discipline, which satisfies the above notion of absolute fairness [6]. The GPS scheduler visits each queue in a round-robin fashion, and serves an infinitesimally small amount of data from each queue, in such a way that in any finite time interval, it can visit every queue at least once. The fairness of a scheduling discipline can be measured in comparison to GPS [9].

Algorithms such as Fair Queuing and Weighted Fair Queuing (WFQ) [6, 12], try to emulate the ideal GPS scheme by time-stamping each arriving packet with the *finish number*, which is the expected completion time that a packet would have had if it were scheduled by the GPS scheduler. The WFQ then serves the packets in the increasing order of the finish numbers. A different pioneering approach was the *Virtual Clock* scheduling discipline [20], which tries to emulate time-division multiplexing instead of GPS. In the virtual clock method, the scheduler timestamps the arriving packets with the completion times under time-division multiplexing, and then serves packets in order of these completion times. Both of these approaches suffer the cost associated with sorting among the timestamps, incurring a work complexity of at least O($\log n$).

A slightly different approach is Deficit Round Robin (DRR) [17], which achieves an O(1) time-complexity because it serves active flows in a strict round-robin order. It succeeds in eliminating the unfairness of pure packet-based round-robin by keeping a deficit counter to measure the past unfairness. A *Quantum* is assigned to each of the queues and a packet is served from a queue only if the packet size at the head of the queue is less than the sum of the deficit counter and the *Quantum* value. Consequently, this requires the scheduler to have a-priori knowledge of the packet lengths arriving at the queues. In the following, we present the ERR scheduling discipline which does not have this requirement and in addition, has better fairness properties.

3. Elastic Round Robin

A pseudo-code implementation of the scheduling algorithm is shown in Figure 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.

We define a flow as *active* when a packet belonging to this flow is in the middle of being dequeued by the scheduler, or when the queue corresponding to the flow is not empty. We maintain a linked list, called the *ActiveList*, of

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

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

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

Figure 1. Pseudo-Code for ERR

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 Figure 2. Assume that flows A, B and C are the only flows active at the beginning of *Round 1*. The visits of the scheduler to the 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.

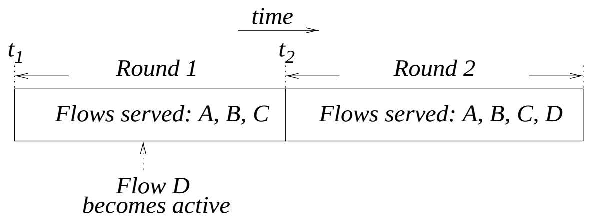

Figure 2. Definition of a Round

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

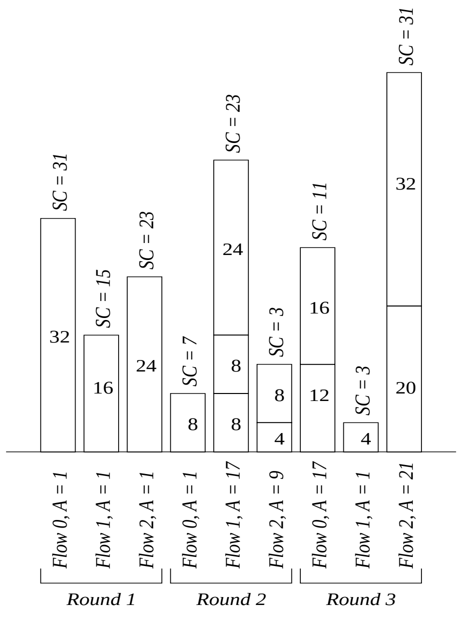

Figure 3. An illustration of 3 rounds in an ERR execution

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 shown,

$$SC_i(r) = Sent_i(r) - A_i(r) \tag{1}$$

Let *MaxSC*(r) denote the largest surplus count among all the flows served during round r. This quantity is used, as follows, to recursively compute the allowances for each of the flows in the next round.

$$A_i(r) = 1 + MaxSC(r-1) - SC_i(r-1) \tag{2}$$

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.

Figure 3 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 the 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 vertical 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.

4. Analytical Results

In this section, we analyze the work complexity of ERR, and also the fairness using a measure based on a popular metric proposed in [9].

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 enqueuing 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 enqueuing and dequeuing a packet are each of time complexity O(1).

The time complexity of enqueuing a packet is the same as the time complexity of the *Enqueue* procedure in Figure 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 Figure 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 also 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 Figure 1. □

4.2. Fairness

In our fairness analysis, we use a fairness measure based on the relative fairness metric first proposed in [9]. Our metric is identical to the one used in [17].

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 *Fairness Measure*, FM(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 FM as the maximum of FM(t_1, t_2) for 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 *Max* as the size in flits of the largest packet that may potentially arrive during the execution of a scheduling algorithm. Note that *Max* $\geq m$.

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 the allowance 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 proved. $\square$

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,

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

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

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

$$Sent_i(r) = 1 + MaxSC(r-1) - SC_i(r-1) + SC_i(r) \quad (3)$$

Summing the LHS in (3) 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 (3) for $r = k$ to $r = k+n-1$,

$$\begin{aligned} N &= n + \sum_{r=k-1}^{k+n-2} MaxSC(r) \\ &\quad + SC_i(k+n-1) - SC_i(k-1) \end{aligned} \quad (4)$$

Using Lemma 1, $0 \leq SC_i(k+n-1) \leq m-1$, and $0 \leq SC_i(k-1) \leq m-1$. The theorem is proved by substituting for these bounds on $SC_i(k-1)$ and on $SC_i(k+n-1)$ in Equation (4). $\square$

We now proceed to prove the bound on the fairness measure of the ERR scheduling discipline. Note that the fairness measure, FM, 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 fairness measure, stated in Theorem 3, considering only the time intervals (t_1, t_2), where $t_1, t_2 \in \mathbf{T}_s$.

Lemma 2. $\text{FM} = \max_{t_1, t_2 \in T_s} \text{FM}(t_1, t_2)$.

Proof: This lemma is proved if for any $t_1, t_2 \in \mathbf{T}$, we can find $t'_1, t'_2 \in \mathbf{T_s}$, such that $FM(t'_1, t'_2) \geq FM(t_1, t_2)$.

Consider any two active flows i and j during the interval between t_1 and t_2, where $t_1, t_2 \in \mathbf{T}$. Without loss of generality, assume that during this 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 $FM(t'_1, t_2) \geq FM(t_1, t_2)$. 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 t_2, can lead to $FM(t'_1, t'_2) \geq FM(t_1, t_2)$. $\square$

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

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 Figure 1, it follows that after flow i receives service, it is added to the

Table 1. Fairness Measure and Work Complexity of Fair Queuing Algorithms

Scheduling Discipline	Fairness	Complexity
Packet-Based Round Robin	∞	O(1)
First-Come-First-Served	∞	O(1)
Fair Queuing [6]	m	O(log n)
Deficit Round Robin [17]	$Max + 2m$	O(1)
Elastic Round Robin	$3m$	O(1)

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,

$$\begin{aligned} & r_j \leq r_i + 1, \\ \text{and} \quad & n_i \leq n_j + 1 \end{aligned} \tag{5}$$

From Theorem 2, for flow i

$$Sent_i(t_1, t_2) \leq n_i + \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) + (m-1) \tag{6}$$

For flow j,

$$n_j + \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k) - (m-1) \leq Sent_j(t_1, t_2) \tag{7}$$

Combining Equations (6) and (7), and using (5), we get,

$$\begin{aligned} Sent_i(t_1, t_2) - Sent_j(t_1, t_2) \leq 1 + 2(m-1) + \\ \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) - \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k) \end{aligned} \tag{8}$$

Let us now consider the quantity D given by,

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

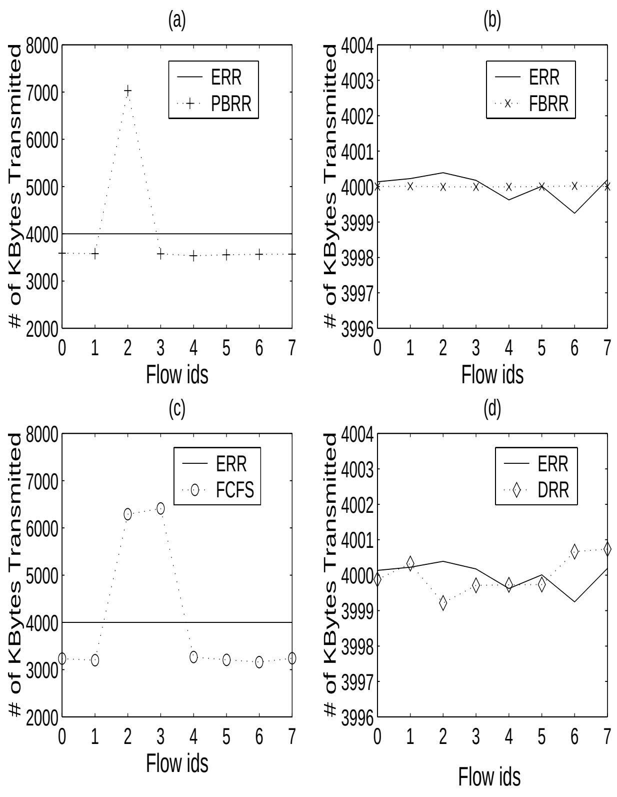

Figure 4. Fairness comparisons between ERR and other scheduling disciplines

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 (8), the statement of the theorem is proved. □

Table 1 shows the relative fairness measure of ERR and its work complexity in comparison to other scheduling disciplines.

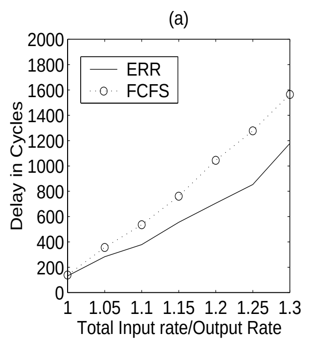

Figure 5. Average packet delays in ERR and other scheduling disciplines

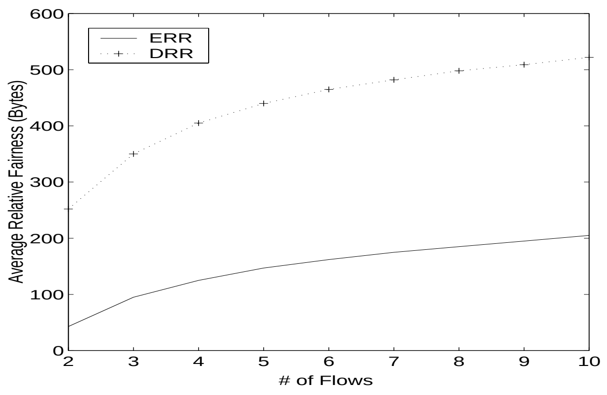

Figure 6. Average relative fairness of ERR and DRR

5. Simulation Results

In this section, we present simulation results on the performance and the fairness properties of the ERR scheduling discipline. The ERR algorithm is compared with other scheduling algorithms of equivalent work complexity like DRR, FBRR, FCFS, and PBRR. We compare the fairness of these disciplines by plotting, for a given interval during which all the flows are active, the number of bytes sent by each of the different flows. Performance comparisons are made based on the average delays experienced by the packets in all of the flows.

Figure 4 shows the results of our first set of experiments. In these experiments, we simulate 8 flows with flow ids from 0 to 7. We collect results for a period of 4 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 1 and 64 flits for all the flows except flow 2. Packets arriving in queue 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.

Figure 4(a) demonstrates that ERR is fair in terms of throughput achieved by the different flows, while PBRR is not. With PBRR, the flow sending larger packets (flow 2 in the figure) gains an unfair fraction of the bandwidth. Figure 4(b) 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. One may note from this figure, that as stated by Theorem 3, the maximum difference between the number of bytes served to different flows is less than $3 \times 128 \times 8$ bytes or 3 KBytes. Figure 4(c) 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 twice the bandwidth. ERR, on the other hand, maintains fairness among the flows independent of packet lengths or injection rates. Finally, Figure 4(d) compares ERR with DRR, and shows that the two scheduling disciplines, for uniformly distributed packet lengths, are comparable in fairness.

Figure 5 shows the simulation results on the average delay of packets with ERR, PBRR and FCFS. We measure the delay of a packet as the number of cycles between the instant it is placed in the queue for scheduling, to the instant its last flit is dequeued. In comparing scheduling disciplines it only makes sense to compare delays experienced by packets in flows that are active during the interval. If the sum of the rates at which the packets are arriving in the flows is greater than the possible output rate, the delays will eventually reach infinity. For a more realistic comparison, therefore, we create a transient congestion period lasting 10,000 cycles, during which the sum of the input rates are higher than the output bandwidth. After these 10,000 cycles, 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 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 4 flows in the simulations in Figure 5, 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 1 to 64, except for flow 2, in which the packet lengths are uniformly distributed between 1 and 128 flits.

Figure 5(a) shows that ERR has a better average latency than FCFS when packet sizes and input rates of the flows can be different. A well-known result in queuing theory states that if a scheduling discipline achieves better average delay than FCFS, it comes at the expense of increasing the delays of some flows [13]. The better average delay of ERR is achieved through the increased delay experienced by flows sending at twice the rate, or flows sending larger packets. ERR, similarly, has a much better average latency than PBRR, as shown in Figure 5(b).

During congestion, since DRR, ERR and FBRR all have bounded relative fairness measures, the packets actually scheduled by these disciplines do not differ much in terms of when they are scheduled. Thus, as far as average delay during transient congestions is concerned, these scheduling disciplines are nearly equal.

Recall that the relative fairness measure for DRR is $Max + 2m$ whereas that for ERR is $3m$. This difference in fairness between these two 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. Figure 6 shows the result of a simulation in which packet lengths in all the flows are exponentially distributed with $\lambda = 0.2$, in the range between 1 to 64. We compute average relative fairness achieved by the ERR and DRR scheduling disciplines, over 10,000 randomly chosen intervals during a period of 4 million cycles. Figure 6 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, ERR achieves better fairness than DRR.

6. Conclusion

In this paper, we have presented a novel scheduling discipline called *Elastic Round Robin* (ERR), which is fair, efficient and satisfies the unique requirements of scheduling in wormhole networks. 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, ERR has better fairness properties. The relative fairness measure of ERR has an upper bound of $3m$, where m is the size of the largest packet that actually arrives during the execution of ERR. Among equally efficient scheduling disciplines, DRR comes closest to ERR in fairness. However, DRR cannot be applied in wormhole networks, where the length of time a packet occupies a resource is not known before the scheduling of the packet. On the other hand, ERR can be adapted to wormhole networks easily, in addition to being perfectly applicable in other contexts such as datagram scheduling in the Internet.

References

[1] ANSI, Inc. *High-Performance Parallel Interface—6400 Mb/s Physical Layer (HIPPI-6400-PH)*, June 1999.

[2] J. Beecroft, M. Homewood, and M. McLaren. Meiko CS-2 interconnect elan-elite design. *Parallel Computing*, 20(10-11):1627–1638, November 1994.

[3] Cray Research, Inc. *Cray T3D System Architecture*, 1993.

[4] W. J. Dally. Virtual channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(3):194–205, March 1992.

[5] W. J. Dally and C. L. Seitz. The torus routing chip. *Journal of Distributed Computing*, 1(3):187–196, October 1986.

[6] A. Demers, S. Keshav, and S. Shenker. Design and analysis of a fair queuing algorithm. In *Proceedings of ACM SIGCOMM*, Austin, September 1989.

[7] J. Ding and L. N. Bhuyan. Evaluation of multi-queue buffered multistage interconnection networks under uniform and non-uniform traffic patterns. *International Journal of Systems Science*, 28(11), 1997.

[8] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[9] S. J. Golestani. A self-clocked fair queuing scheme for broadband applications. In *Proceedings of IEEE INFOCOM*, Toronto, June 1994.

[10] Intel Corporation. *Paragon XP/S Product Overview*, 1991.

[11] M. Katevenis, P. Vatsolaki, and A. Efthymiou. Pipelined memory shared buffer for VLSI switches. In *Proceedings of ACM SIGCOMM*, pages 39–48, August 1995.

[12] S. Keshav. On the efficient implementation of fair queuing. *Journal of Internetworking Research and Experience*, 2(3), September 1991.

[13] L. Kleinrock. *Queuing Systems, Volume 2: Computer Applications*. Wiley Interscience, 1975.

[14] V. P. Kumar, T. V. Lakshman, and D. Stiliadis. Beyond best effort: Router architectures for the differentiated services of tomorrow's internet. *IEEE Communications Magazine*, May 1998.

[15] M. Pirvu, L. Bhuyan, and N. Ni. The impact of link arbitration on switch performance. In *Proceedings of the Fifth Symposium on High-Performance Computer Architecture*, January 1999.

[16] H. Sethu, C. B. Stunkel, and R. F. Stucke. IBM RS/6000 SP large system interconnection network topologies. In *International Conference on Parallel Processing*, Aug 1998.

[17] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE Transactions on Networking*, 4(3), June 1996.

[18] C. B. Stunkel. The SP2 high-performance switch. *IBM Systems Journal*, 34(2):185–204, February 1995.

[19] Y. Tamir and G. L. Frazier. Dynamically allocated multi-queue buffers for VLSI communication switches. *IEEE Transactions on Computers*, 41(6), June 1992.

[20] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *Proceedings of ACM SIGCOMM*, Philadelphia, September 1990.