

# A Round-Robin Scheduling Strategy for Reduced Delays in Wormhole Switches with Virtual Lanes \*

Harish Sethu, Hongyuan Shi, Salil S. Kanhere and Alpa B. Parekh  
Department of Electrical and Computer Engineering  
Drexel University  
3141 Chestnut Street  
Philadelphia, PA 19104, U.S.A.

**Abstract.** *The forwarding of flits from one switch to another in wormhole networks with virtual lanes is typically accomplished using Flit-by-Flit Round-Robin (FFRR) scheduling. This paper presents the Anchored Round-Robin (ARR) scheduling discipline, which preserves the throughput characteristics of FFRR while significantly reducing the average delay experienced by packets. The ARR scheduler achieves the lower average delay by attempting to transmit all flits of a packet before beginning the transmission of flits from another packet. The ARR scheduler is idle only when there are no flits to transmit, thus achieving the same throughput characteristics as FFRR. This paper further presents a simulation-based analysis of the ARR scheduler, and the dependence of the performance obtained on the offered load and the packet lengths.*

**Keywords:** Wormhole, Switch, Virtual Channels, Fair Scheduling, Round-Robin.

## 1 Introduction

Wormhole switching, frequently also referred to as wormhole routing, is a technique in which the unit of flow control is a *flit*, a pre-defined number of bits smaller than a packet [1]. Variations of the original wormhole switching technique are used today in the majority of parallel systems and system area networks. In wormhole switching, each flit of a packet can make forward progress to the next switch in its path, as long as there is buffer space available for at least one flit in that next switch. In order to ensure that slow-moving packets do not unnecessarily block access to an output link, the concept of virtual lanes [2], also known as virtual channels, is frequently used. Since each flit is marked by the virtual lane it belongs to, in scheduling flits for transmission over an output

link, it is not necessary to schedule all flits belonging to a packet before a flit from another virtual lane is scheduled. This paper is concerned with the scheduling of flits from among buffers for different virtual lanes in wormhole switches.

Over the last decade, a number of advances have been made in the architecture of switches for improved delay and throughput characteristics. Very few researchers, however, have focused on possible performance gains with scheduling strategies that may be used at the output link or for access to output queues from the input queues. These scheduling strategies, in addition, are critical to the fairness achieved in the switch, beyond just the elimination of starvation scenarios. Almost all wormhole switches today use the Flit-by-Flit Round-Robin (FFRR), the Packet-by-Packet Round-Robin (PPRR) or the First-Come-First-Served (FCFS) method of scheduling packets from output queues to the output link, and also from input queues to the output queues. FFRR exhibits sub-optimal delay characteristics, while PPRR has among the worst throughput characteristics of all scheduling strategies for wormhole networks [3]. The First-Come-First-Served scheduling strategy is fundamentally unfair as given by the widely used measure of relative fairness first proposed in [4]. Since our larger research goal is a strictly fair switch with high-performance, this study focuses on round-robin scheduling strategies that can be modified to achieve fairness as in [5, 6].

This paper presents the *Anchored Round-Robin (ARR)* scheduling strategy which preserves the throughput characteristics of FFRR while achieving significantly improved delay characteristics. This scheduling strategy maintains an *anchor* virtual lane, to which the scheduler returns during each operation to attempt to serve a flit, if possible. Only when no flits can be served from the anchor virtual lane, does the ARR scheduler attempt to serve a flit from one of the other virtual

\*This work was supported in part by U.S. Air Force Contract F30602-00-2-0501.

lanes. This serves to significantly improve the average delay experienced by packets using the ARR scheduler as opposed to the FFRR scheduler.

Section 2 describes the simulation environment used in this study. Section 3 describes the details of the ARR scheduling strategy. Section 4 presents the simulation results and an analysis based on these results. Section 5 concludes the paper.

## 2 Simulation Environment

This study uses a model of a multistage interconnection network using output-queued switches with input buffers operating in a credit-based flow control system. In the simulation results presented in this paper, we use  $2 \times 2$  switches with either 2 or 4 virtual lanes implemented. It is well-known that dynamic sharing of input and output buffers between various virtual lanes yields a better performance. However, as mentioned in Section 1, our ultimate goal is a strictly fair switch and in this respect, our research currently seeks to investigate the limits of performance with round-robin scheduling strategies and with completely partitioned buffering. Our switch model, therefore, uses dedicated buffers for each of the virtual lanes at each input and output port.

The forwarding control logic in our switch for all output and input ports is centrally controlled instead of distributed at the input and output ports. In order to reduce the complexity of the crossbar implementation with multiple input and output ports and large numbers of virtual lanes with dedicated buffers, we assume that the input and output buffers are not completely connected. In fact, during any given switch cycle, a maximum of one input queue and a maximum of one output queue may be served. This aspect of our switch is similar to the IBM SP switch discussed in [7], where only one input (output) port can write (read) the shared buffer during any given switch cycle. In order that the switch forwarding logic has the same bandwidth as the input or output bandwidth of its links, we assume that the flit size is defined in such a way that the length of time it takes to receive or transmit a flit at a link is equal to the number of input or output ports on the switch multiplied by the switch cycle time. In our model, the minimum delay through the switch is 3 cycles, and the propagation delay on all the cables is equal to exactly 1 cycle.

When a new packet appears at the head of an input buffer, and it has to choose a virtual lane, we use a greedy algorithm in a round-robin manner. For example, all virtual lanes are examined in a

round-robin fashion, and the first virtual lane that has room for at least one flit is used. The virtual lane of a packet may change during each switch hop as the packet progresses toward a destination. This strategy of virtual lane assignment is different from that in [2] where a virtual lane is randomly chosen among the available virtual lanes. Achieving acceptable fairness bounds provides our motivation to avoid random assignment of virtual lanes.

The focus of this paper is on scheduling strategies among virtual lanes for entry into and exit out of dedicated output buffers for virtual lanes. In order to isolate the impact of the scheduling strategy across virtual lanes, we have chosen to completely eliminate any choice in the output ports for packets entering a switch. The Banyan network topology allows us to do precisely that by making sure that there is exactly one path available from any input to any output of the network. The simulation results presented in this paper use an  $8 \times 8$  Banyan network topology.

## 3 Anchored Round-Robin

As in a Packet-by-Packet Round-Robin (PPRR) strategy, the ARR algorithm attempts to first transmit all flits of the same packet before beginning the service for the next packet. During some cycles, however, there is either no token or no flit available to continue the transmission of the same packet. The ARR algorithm presented here moves on to transmit a flit from another virtual lane when such a condition occurs. This strategy offers a hybrid approach that combines the advantages of both FFRR and PPRR.

In each switch, a version of the ARR algorithm is executed at each of the output links for scheduling packets from the output queues corresponding to different virtual lanes to the output link. In addition, a version of the algorithm is used by the schedulers which control the entry into the output queues from the input queues. In this paper, for the sake of clarity and brevity in presenting the central concept of ARR, we describe only the version that is implemented at the output link.

The ARR scheduler maintains a current state in the form of a virtual lane identifier called the *Anchor Virtual Lane* or *AnchorVL* updated during each switch cycle. The *AnchorVL*, during each cycle, receives the first opportunity to send a flit given that it has a flit and the token necessary for transmission. If *AnchorVL* is not able to avail of this service opportunity, other virtual lanes, in round robin fashion, receive the opportunity to send a flit to the output link. Once a flit is transmitted, or if no flit is

transmitted, the execution of the scheduler during that cycle is complete. During each new execution of the ARR scheduler, the first service opportunity is always given to *AnchorVL*, independent of which virtual lane was served during the last cycle. *AnchorVL* is updated to the next virtual lane in the round-robin order whenever the last flit of a packet has been transmitted from the *AnchorVL*. The quantity *AnchorVL* is initialized to some valid value at the time the switch is initialized for operation. The following step-by-step description of the execution of the ARR scheduler during each cycle clarifies further details.

1. If *AnchorVL* has already been examined in this cycle, go to Step 8.
2. Examine the buffer for virtual lane *AnchorVL*. If there is no flit in this buffer *AND* if the scheduler is not in the middle of transmitting a packet from *AnchorVL*, increment *AnchorVL* to the next virtual lane in the round-robin order and go back to Step 1.
3. Set *vl* to *AnchorVL*.
4. If the buffer for virtual lane *AnchorVL* has a flit *AND* if there are sufficient credits available for transmitting this flit, then transmit the flit from virtual lane *AnchorVL* and go to step 7.
5. Increment *vl* to the next virtual lane in the round-robin order. If virtual lane *vl* has already been examined in this cycle, go to Step 8.
6. Examine the buffer for virtual lane *vl*. If this buffer has a flit *AND* if there are sufficient credits available for transmitting this flit, then transmit the flit from virtual lane *vl*. Otherwise, go back to Step 5.
7. If the flit transmitted was the last flit of a packet *AND* if *vl* equals *AnchorVL*, increment *AnchorVL* to the next virtual lane in the round-robin order.
8. Exit. This step completes the execution of the scheduler during this cycle.

## 4 Simulation Results and Analysis

Figure 1 shows a plot of the average delays (measured in cycles) obtained with the FFRR and the ARR scheduling strategies, in an  $8 \times 8$  Banyan network, using  $2 \times 2$  switching elements. This figure plots the delays for 2 and 4 virtual lanes per

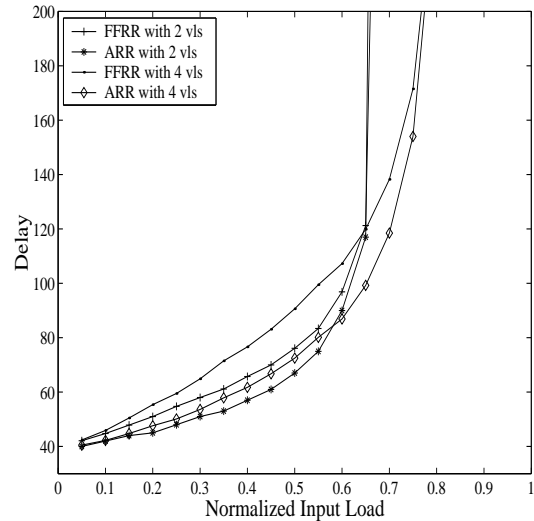


Figure 1: Average delays with FFRR and ARR

port, using a packet size of 32 flits. It is easily observed from this figure that ARR yields a lower average delay than FFRR. An additional observation that can be made is that, as the number of virtual lanes increases, the delay advantage of ARR over FFRR increases. This is consistent with intuition, since the increase in the number of virtual lanes increases the number of stops of the round-robin scheduler before the same virtual lane gets served again. This uniformly increases the delay of packets from all virtual lanes, and this increase is proportional to the number of virtual lanes. One may observe, however, that the saturation point does not change significantly with whether FFRR or ARR is used. This is again quite expected since the scheduling algorithm is only changing the order in which flits are served but not how many flits are served within a given period. It is for this reason that we do not plot throughput curves in this paper—ARR and FFRR yield almost identical throughput at all offered loads.

A further interesting observation is that the benefits of ARR over FFRR are greatest at moderate input loads. At very low loads or at very high loads, FFRR and ARR yield average delays that are approximately equal. The phenomenon at low loads is easily understood—at low loads, the FFRR and the ARR algorithms tend to be equivalent in the order in which they schedule flits since, during any given cycle, the scheduler is not likely to have any more than one virtual lane with flits ready to be scheduled. At high loads, however, the phenomenon is more complex and is caused by the

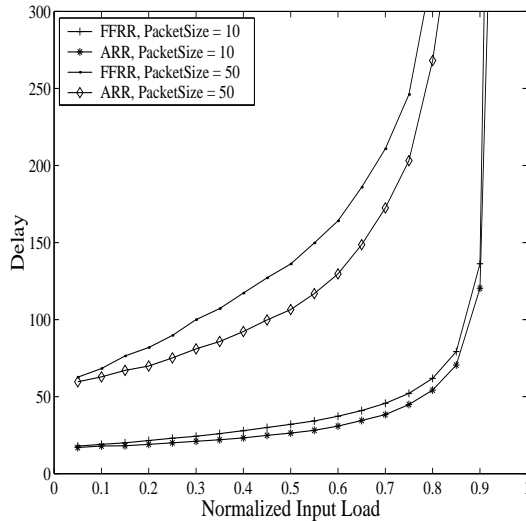


Figure 2: Avg. Delays with large & small packets

increased blocking experienced by packets. Consider the ARR scheduler that has just initiated the scheduling of a packet. In the absence of blocking, the ARR scheduler would transmit all flits of this packet before proceeding to transfer flits from another virtual lane. At high loads, however, in a credit-based switching network, the scheduler runs out of tokens more often than at low or moderate loads. This leads to blocked packets, causing the ARR scheduler to schedule flits from other virtual lanes. The ARR scheduler, thus, once again, begins to behave more like the FFRR scheduler since it is unable to schedule all flits of a packet in consecutive cycles.

Figure 2 compares the average delays obtained with packets of length 10 flits each against packets of length 50 flits each. In order to eliminate the effect of buffer size in relation to packet size at moderate loads, we use larger input buffers of size 512 flits for these plots. This figure is plotted using switch chips with 4 virtual lanes. Clearly, the advantage of ARR over FFRR with longer packets is significantly more than with smaller packets. This is quite simply due to the fact that FFRR uniformly increases the delays of all packets in proportion to the size of the packet.

## 5 Conclusion

In this paper, we have presented the Anchored Round-Robin (ARR) scheduler, which reduces the average delay experienced by packets in a wormhole network with virtual lanes as compared to

the traditional Flit-by-Flit Round-Robin (FFRR) scheduler. ARR is most effective at moderate loads in reducing the delays. At heavy loads, ARR approaches FFRR because increased blocking reduces the ARR algorithm to FFRR. This situation can be somewhat alleviated through synchronizing the packet forwarding of the schedulers on the same switch.

In this paper, the round-robin strategies are analyzed assuming that all packet sizes are of equal lengths. In order to ensure a fair allocation of bandwidth between different virtual lanes which may use different sized packets, a fair scheduler such as Deficit Round-Robin [5] or Elastic Round-Robin [6] would have to be used in conjunction with the ARR algorithm.

## References

- [1] W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, October 1986.
- [2] W. J. Dally, "Virtual Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 3, pp. 194–205, March 1992.
- [3] M. Pirvu, L. Bhuyan and N. Ni, "The Impact of Link Arbitration on Switch Performance" *Proceedings of the Fifth Symposium on High-Performance Computer Architecture*, January 1999.
- [4] S. J. Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," *Proceedings of IEEE INFOCOM'94*, Toronto, June 1994.
- [5] M. Shreedhar and George Varghese, "Efficient Fair Queuing Using Deficit Round-Robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996.
- [6] S. Kanhere, A. Parekh and H. Sethu, "Fair and Efficient Packet Scheduling in Wormhole Networks," to appear in *Proceedings of the International Parallel and Distributed Processing Symposium*, Cancun, May 2000.
- [7] C. B. Stunkel, *et al.*, "The SP2 High-Performance Switch," *IBM Systems Journal*, vol. 34, no. 2, pp. 185–204, February 1995.