Anchored Opportunity Queueing: A Low-Latency Scheduler for Fair Arbitration among Virtual Channels

Salil S. Kanhere[a] and Harish Sethu [a]

[a]Department of ECE, Drexel University
3141 Chestnut Street, Philadelphia, PA 19104, USA
E-mail: {salil, sethu}@ece.drexel.edu

Many wormhole interconnection networks for parallel systems, and more recently system area networks, implement virtual channels to provide a number of services including improved link utilization and lower latencies. The forwarding of flits from the virtual channels on to the physical channel is typically accomplished using Flit-Based Round-Robin (FBRR) scheduling. This paper presents a novel scheduling strategy, *Anchored Opportunity Queueing (AOQ)*, which preserves the throughput and fairness characteristics of FBRR while significantly reducing the average delay experienced by packets. The AOQ scheduler achieves lower average latencies by trying, as far as possible, to complete the transmission of a complete packet before beginning the transmission of flits from another packet. The AOQ scheduler achieves provable fairness in the number of opportunities it offers to each of the virtual channels for transmissions of flits over the physical channel. We prove this by showing that the relative fairness bound, a popular measure of fairness, is a small finite constant in the case of the AOQ scheduler. Finally, we present simulation results comparing the delay characteristics of AOQ with other schedulers for virtual channels. The AOQ scheduler is simple to implement in hardware, and also offers a practical solution in other contexts such as in scheduling ATM cells in Internet backbone switches.

1. Introduction

A fundamental requirement for designing scalable parallel computer systems is a high-bandwidth low-latency interconnection network. Almost all interconnection networks, both direct and indirect, are constructed out of switches connected together in a certain topology. The switching technique is one of the primary aspects that characterizes the architecture of a switch. Wormhole switching, frequently also referred to as wormhole routing [1], and its variations are widely used in a variety of parallel systems and more recently in system area networks [2–6]. In wormhole switching, a packet is usually divided into a number of *flits* (flow control digits) for transmission. In order to reduce the per-flit overhead, only the flit at the head of the packet, or the header flit contains information necessary to route the packet through the network. As the header flit advances along the specified route, the remaining flits follow it in the same path. Since only the header flit contains the routing information, the flits in one packet cannot be readily interleaved with flits from another packet over the same physical channel, thus allowing a slow moving packet to unnecessarily block access to an output link. The network throughout and latency, however, can be improved by organizing the buffers associated with each physical channel to implement multiple *virtual lanes*[1] [7], with each flit carrying a small field indicating the virtual channel to which it belongs. Flits from different packets, as long as they belong to different virtual lanes, may now be interleaved over a single physical channel. This allows a packet from one virtual lane to bypass a blocked packet of another virtual lane, and thus utilize the network bandwidth that would otherwise go unused. Systems with virtual lanes have now increasingly gained significant presence in the commercial marketplace [8–11].

Given flits from multiple virtual lanes awaiting transmission through an output link at a switch, the decision on which flit to transmit next involves the

[1]In addition to wormhole networks, this work is also applicable in the transmission of IP packets over ATM networks in Internet backbones. The concept of *virtual channels* in ATM networks is not the same as the concept of *virtual channels* introduced in [7]. Therefore, in order to avoid confusion over terminology, we use the term virtual *lanes* in our current context instead of virtual *channels*.

competing issues of achieving fairness among the different virtual lanes and achieving a low latency for the packets. Most wormhole switches today employ Filt-Based Round Robin (FBRR) scheduling, First-Come First-Served (FCFS) scheduling or Packet-Based Round Robin (PBRR) scheduling among the virtual lanes. However, none of these scheduling schemes can be fair and provide a good performance at the same time. FBRR exhibits sub-optimal delay characteristics, while PBRR has among the worst throughput characteristics of all scheduling strategies for wormhole networks [12]. FCFS achieves low average delays but is fundamentally unfair as given by the widely used measure of relative fairness first proposed in [13]. In this paper, we present a novel scheduling strategy, called Anchored Opportunity Queueing (AOQ), that achieves provable fairness as well as a very low latency.

1.1. Motivation

The motivation for achieving low latencies and high throughput is well-understood in the parallel processing research community. The secondary goal of this work, to achieve fairness in the allocation of the physical channel among the various virtual lanes, is not as well-understood; therefore, we will describe this component of our motivation in greater detail.

It should be noted that fairness among virtual lanes does not necessarily imply equal access to the physical channel since different virtual lanes may be at different levels of priority. Most switches used in interconnection networks of parallel systems today guarantee the elimination of only the worst kinds of unfairness, such as starvation scenarios wherein packets from a certain virtual lane may not receive service for an indefinite period of time. Strict fairness, however, is desirable in a variety of contexts in which virtual lanes are employed in switches:

- In systems that use virtual lanes for deadlock avoidance [7], it is important that data traffic flows from different virtual lanes are treated fairly by the network. Strict fairness, besides being intuitively appealing, can actually improve performance by eliminating some bottlenecks.

- In some parallel systems, virtual lanes are used to provide pre-emptive priority for control packets. Allowing a separate virtual lane for control packets enables the scheduler to give a higher priority to a control flit and pre-empt a data packet currently being transmitted. Some parallel systems, such as the RS/6000 SP systems, allow more than one level of priority in classifying user jobs. This can be implemented by aggregating each class of traffic into a separate virtual lane and scheduling fairly based on pre-assigned weights for each virtual lane.

- Virtual lanes may be used to implement virtual networks. For the purposes of software development, debugging, testing and reliability, it sometimes helps to have multiple functioning virtual networks on the same physical network so that regular jobs can be protected from the experimental software installed for testing on another virtual network. Implementing virtual networks through using virtual lanes is one of the effective methods that one can use to isolate some traffic from others, as has frequently been desired by customers of parallel systems [14]. Such fairness also protects flows using a particular virtual lane from misbehaving or heavy flows possibly generated by experimental code in other virtual lanes, offering a more predictable performance to users of parallel systems.

It should be noted that achieving fairness among different virtual lanes for access to a physical channel is not the same as achieving fairness among various flows of traffic. Many individual flows of traffic may be aggregated into a few virtual lanes, and therefore, achieving fairness amongst virtual lanes does not necessarily ensure fairness amongst individual flows of traffic. This paper proposes a solution to the problem of ensuring fairness amongst virtual lanes; the problem of ensuring fairness amongst individual flows of traffic is an independent problem, solutions to which can co-exist with the solution proposed here. Our AOQ scheduler may be combined with schedulers that achieve fairness among individual flows, such as Elastic Round Robin (ERR) [15], to achieve overall fairness among the flows as well as among the virtual lanes. For example, ERR may be used to determine the order in which packets enter the output buffers corresponding to the different virtual lanes, while our

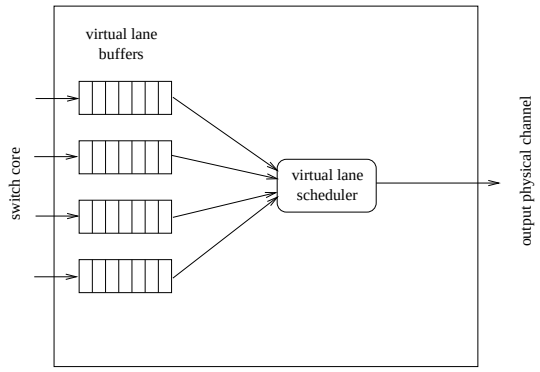

Figure 1. A generic block diagram of an output port in a wormhole switch

AOQ scheduler may be used to schedule flits from these buffers on to the physical channel.

1.2. Goals and Contributions

Fig. 1 illustrates the architecture of a generic output port of a wormhole switch with virtual lanes. The focus of this paper is to design an appropriate scheduler for forwarding flits from the various virtual lane buffers on to the output link.

In this paper, our goal is to develop a practical scheduling strategy that achieves:

- Low average delays through attempting, as far as possible, the complete transmission of a packet before beginning the transmission of another packet.
- High throughput through work-conserving operation, i.e., the physical channel should never be idle if there is a flit in a virtual lane buffer and is eligible for transmission.
- Fair allocation of the physical channel among the various virtual lanes.
- Ease of implementation as a hardware module in a switch chip.

We propose the Anchored Opportunity Queueing (AOQ), a novel scheduling discipline that achieves all of the above four goals. It employs the idea of anchoring used in ARR, and selects flits to transmit based on a virtual number associated with each virtual lane. When virtual lanes are all set to be of equal priority, the AOQ scheduler gives each virtual lane, within provably tight bounds, an equal number of opportunities to transmit a flit. This does not necessarily imply that the number of flits actually transmitted will be the same from each virtual lane since the amount of traffic on each lane may be very different. The AOQ scheduler achieves fairness in the opportunity to transmit a flit in spite of the fact that the scheduler prioritizes the transmissions of the virtual lane that is currently in the middle of transmitting a packet.

1.3. Organization

The rest of this paper is organized as follows. Section 2 presents a brief overview of fair scheduling algorithms. Section 3 describes the AOQ scheduler along with the rationale behind it. Section 3 also presents a pseudo-code description of the AOQ scheduler. Section 4 presents an analysis of the fairness achieved by the AOQ scheduler, and proves the upper bound on the relative fairness bound, a popular measure of fairness. Section 5 presents simulation results that show the low delay characteristics of the AOQ scheduler. Section 6 concludes the paper with some remarks on future work and on the implications of this work in contexts beyond wormhole networks.

2. Background

Consider the problem of allocating a shared resource among multiple requesting entities with equal rights to the resource but unequal demands. A classical method that defines an ideally fair allocation is the *max-min* strategy, stated as follows [16],

- No requesting entity obtains a share of the resource larger than its demand.
- All requesting entities whose demands are not satisfied get an equal share of the resource.

Here we assumed that all the entities had equal rights to the resource, but in reality there may be a weight associated with each entity that reflects its relative resource share. In this case the max-min fair share policy is modified such that the shared resource allocated to an entity is normalized by its corresponding weight.

The Generalized Processor Sharing (GPS) algorithm [17, 18] satisfies the above classical notion of fairness. Among all requesting entities with equal rights to the resource, GPS serves an infinitesimal amount of the shared resource to each of the requesting entities in a round robin fashion. The GPS scheduler, though ideally fair, is unimplementable because it assumes that all traffic in the network is infinitely divisible.

In recent years, a variety of algorithms that try to emulate the ideal GPS scheme have been proposed and implemented in packet switched networks such as the Internet. For example, the Weighted Fair Queueing (WFQ) scheduler [17] serves packets in the order in which they would finish transmission under the ideal GPS scheduler. Each packet is stamped with the finish time under GPS, and packets are then served in order of these timestamps. Many variations of WFQ have been proposed in the literature, and several of these have been adopted for use in Internet routers to ensure fairness in the management of best-effort traffic. These include Deficit Round Robin (DRR) [19], Self-Clocked Fair Queueing (SCFQ) [13], Worst-case Fair Weighted Fair Queueing (WF^2Q) [20] and Smoothed Round Robin (SRR) [21]. These scheduling disciplines differ in the level of fairness guarantees they provide and the ease of implementation.

The above fair scheduling strategies designed for use in Internet routers do not serve us well in our goal of trying to achieve fairness among the virtual lanes in a wormhole network. In Internet routers, each packet begins transmission only after it can be guaranteed that all of the packet will be available for completing the transmission without interruption. The problem reduces to determining the order in which to schedule packets for transmission. In wormhole networks with virtual lanes, the granularity of packet forwarding is a flit, a sub-division of a packet. A few flits of a packet may be transmitted while the remaining flits may not be available for transmission. During this time, the scheduler may choose to interleave the flits of another packet from a different virtual lane. In Internet routers, each service opportunity given to a flow leads to a transmission; in wormhole networks, each service opportunity given to a virtual lane may or may not lead to the transmission of a flit from the virtual lane. This is the unique challenge of trying to adapt the principles of fair scheduling developed for Internet routers into the context of virtual lanes in wormhole networks.

Fairness amongst virtual lanes in a wormhole network may be achieved by using Flit-Based Round-Robin (FBRR), where the scheduler visits each virtual lane in round robin order and tries to schedule a flit, if available, for transmission. However, this introduces an increased average delay for all the packets. For example, consider four packets of 10 flits each awaiting transmission in four different virtual lanes. Under FBRR, assuming one flit transmission per cycle, the transmissions of the four packets will complete at 37, 38, 39 and 40 cycles, assuming one cycle per flit transmission. However, if we transmitted all the flits of an entire packet in consecutive cycles, the average latency drops to 25 cycles since the packets complete transmissions at 10, 20, 30 and 40 cycles.

In general, it is desirable that we try to complete the transmission of a packet, as long as flits of the packet are available, before beginning the transmission of another packet. Transmitting multiple flits from a packet also increases the throughput by eliminating the per-flit acknowledgments traveling on the reverse path [22, 23]. The use of packet-based round-robin (PBRR), however, wastes bandwidth since the physical channel may be idle when all the flits of a packets that have begun transmission are not available in the virtual lane buffer. PBRR, therefore, has among the worst throughput characteristics of all scheduling strategies for wormhole networks [12]. Low latencies and in addition, throughput characteristics similar to FBRR, may be obtained using Anchored Round-Robin (ARR) [24], proposed for use in wormhole networks with virtual lanes. ARR attempts to achieve the low latency properties of a packet-by-packet transmission strategy by attempting to transmit all the flits of one packet before moving on to another packet. If only the first few flits of a packet are available, the ARR scheduler transmits these flits and begins serving flits from another virtual lane. Meanwhile, the ARR scheduler remembers the previous virtual lane as the anchor lane and returns to the anchor lane as flits become available for transmission. The ARR scheduler, while achieving good throughput and low latency, is not fair in terms of giving all virtual lanes an equal opportunity to transmit flits over the physical channel. In addition, extreme unfairness may result when packets are of variable size.

3. Anchored Opportunity Queueing

We now introduce a few preliminary definitions that facilitate the discussion and analysis of the AOQ scheduler.

3.1. Preliminary definitions

Definition 1 *Let t_1 be the time instant at which the scheduler begins serving the header flit of a packet from the a virtual lane buffer. Let t_2 be the instant at which the scheduler completes serving the last flit of the packet. A virtual lane is said to be* busy *during the time interval (t_1, t_2).*

A virtual lane is busy only when one of its packets is in the middle of completing its transmission. Presence of flits in the virtual lane buffer awaiting transmission does not necessarily imply that the virtual lane is busy. For example, when the header flit of a packet is awaiting transmission at the head of the queue corresponding to the virtual lane, some packets from this virtual lane have completed transmission while a new packet's transmission has not yet begun.

Definition 2 *A virtual lane is said to be* active *over a time interval (t_1, t_2) if, and only if, at each time instant during the interval at least one of the following holds true:*

1. *The virtual lane is busy.*
2. *There are flits awaiting transmission in the virtual lane buffer.*

Definition 3 *The virtual lane scheduler is said to be busy over the time interval (t_1, t_2) if at each time instant during the interval, there is at least one active virtual lane.*

Only active virtual lanes are considered to be in competition for the opportunity to forward a flit on to the physical channel. Our goal is the design of a scheduler such that the number of opportunities to transmit a flit offered to each of the active virtual lanes is max-min fair.

3.2. The Principles

Anchored Opportunity Queueing (AOQ) is based on two important principles: *anchoring* and *opportunity scheduling*.

Anchoring. Anchoring is the principle of trying to ensure that an unfinished packet is given the first available opportunity to forward a flit before flits from other packets are forwarded. This reduces the average end-to-end delay in comparison to a scheduling strategy such as FBRR. We call this the anchoring principle since the virtual lane with the packet in the middle of its transmission serves as an anchor to the scheduler. When this anchor virtual lane does not have all the flits in its buffer available to complete transmission of the packet, the scheduler strays and transmits flits from other virtual lanes but returns to serve the anchor virtual lane as soon as one or more flits become available for transmission from this lane.

Opportunity Scheduling. An opportunity scheduler guarantees fairness in the opportunities it offers to competing entities in the access to the shared resource. Note that this is not the same as allocating equal amounts of the resources to each competing entity. For example, when a virtual lane has very little to transmit while another virtual lane has a lot to transmit, fairness does not entail that we force both lanes to transmit equal amounts of traffic. Fairness implies that we offer each virtual lane the same number of opportunities to transmit, while the lane with less traffic will use fewer of the opportunities offered to it.

In wormhole networks with virtual lanes, each service opportunity that is offered to an entity may not necessarily result in the entity obtaining a share of the resource. In other words, it is possible that an opportunity offered to a virtual lane does not result in a flit from that virtual lane being forwarded on to the physical channel. The following are two instances when a service opportunity offered to an active virtual lane is not used by the virtual lane.

- *Downstream congestion.* If the flits of a packet cannot make further progress due to downstream congestion the packet is blocked in place and typically occupies lane buffers in several routers.
- *Pipeline bubbles.* Ideally, in virtual lane flow control, all the flits of a packet should arrive at a buffer right behind each other in an uninterrupted pipeline fashion. However, due to uneven utilization of the physical channel, some flits of a packet may be forwarded much faster than others, creating what are known as

pipeline bubbles. This phenomenon in wormhole networks with virtual lanes, also described in [25], leads to a situation where service opportunities offered to a virtual lane will not be used since the lane buffer is empty.

In both the above cases, a work-conserving scheduler will visit other active virtual lanes and try to schedule a flit from one of them. The order in which the active lanes are considered for service is determined by the scheduling discipline in use.

Note that the total service opportunities required to transmit a packet need not be related to the length of the packet. In fact, the exact number of service opportunities needed to schedule a packet will only be known when the last flit of that packet is served. For instance, it may be possible that during a certain time interval, none of the service opportunities offered to a certain virtual lane i, are utilized. If the virtual lanes are considered to be contending for the link bandwidth, the resource allocated to virtual lane i is equal to zero during the interval under consideration. However, the scheduler has not really been unfair in the resource allocation since it has offered a fair number of service opportunities to virtual lane i. An appropriate measure upon which one should base the fairness criteria, therefore, is the total number of service opportunities received by an entity to access the resource rather than the actual share of the resource used by the entity. Similar principles have also been used in the management of computing resources on large multiuser UNIX systems such as those described in [26].

It may be noted that the principle of fairness used in fair scheduling algorithms in the Internet is a subset of this principle, applied in a context where each service opportunity offered to an active flow is always fully exploited by the flow. In the absence of hop-by-hop credit-based flow control, output buffers in Internet routers do not block packets due to downstream congestion. Also, with the granularity of forwarding being a packet, there are no pipeline bubbles within a packet. Thus, the amount of service utilized by an active flow corresponds exactly to the amount of service opportunities made available to the flow.

3.3. The AOQ scheduler

The AOQ scheduler employs the above two principles to achieve fairness as well as low average delays. In achieving fairness, we use the sorted packet selection scheme employed in sorted-priority schedulers which are widely used in the Internet [27]. The scheduler maintains a current state in the form of a virtual lane identifier called the *Anchor Virtual Lane* or *AnchorVL*. During each cycle the scheduler first visits the *AnchorVL* and tries to schedule a flit from its buffer. In addition, the scheduler also maintains a flag, *AnchorVLValid*, which indicates the validity of the *AnchorVL*. This flag is set over the entire time interval during which the *AnchorVL* is busy. Once the scheduler finishes serving the tail flit from the *AnchorVL*, the *AnchorVLValid* flag is reset, implying that the *AnchorVL* needs to be updated. In other words, the *AnchorVL* has significance only if the *AnchorVLValid* flag is set. The scheduler maintains a state for each virtual lane, called the *OpportunityCount* which keeps a count of the total service opportunities provided to the corresponding virtual lane. Whenever the scheduler visits an active virtual lane, its associated *OpportunityCount* is incremented by one irrespective of whether a flit was served or not. The scheduler also maintains a variable known as the *VirtualCount* which is similar to the virtual time function maintained in the sorted-priority schedulers [27]. The role of the *VirtualCount* is to reset the value of a non-active virtual lane's *OpportunityCount* when it becomes active. Let OC_i represent the *OpportunityCount* of virtual lane i. The *OpportunityCount* of a newly active virtual lane is calculated as follows,

$$OC_i = max(OC_i, VirtualCount) \quad (1)$$

This arrangement does not allow the accumulation of service credits by a non-active virtual lane which becomes active after a long time.

In order to avoid the complexity of emulating GPS as is done in some fair scheduling algorithms proposed for the Internet [28], we do not define the *VirtualCount* function with respect to the GPS system. The *VirtualCount* function is defined to be equal to the lowest value of the *OpportunityCount* among all the active virtual lanes. Note that, these rules for assigning the *OpportunityCount* and the *VirtualCount* are similar to those used in the SFQ and SCFQ scheduling disciplines [13, 29].

The AOQ scheduler also maintains a linked list, called the *ActiveList*, of all the active virtual lanes excluding the *AnchorVL*. This linked list is sorted in the increasing order of the *OpportunityCount* of the

```
Initialize: (Invoked when the scheduler is initialized)
    AnchorVLValid = FALSE;
    AnchorVL = 0;     /* This can be set to any value */
    VirtualCount = 0;
    ContinueService = TRUE;
    Counter = 1;
    for (i = 0; i < n; i = i + 1)
        OC_i = 0;

Enqueue: (Invoked when a header flit arrives)
    i = QueueInWhichHeaderFlitArrives;
    if (ArrivingFlitIsAtHeadOfQueue(i) == TRUE ) then
        OC_i = Max(VirtualCount, OC_i);
        AddEntryToActiveList(i, OC_i);
    end if;

Dequeue:
    while (TRUE) do
        ContinueService = TRUE;
        VirtualCount = Min(OC_AnchorVL, ReadCountFromActiveList(Head));

        SelectAnchorVL();
        ServeAnchorVL();
        ServeFromActiveList();
        CheckEndOfBusyPeriod();
    end while;
```

Figure 2. Pseudo-Code for Anchored Opportunity Queueing

constituent virtual lanes. This allows the virtual lane which has received the least service opportunities to become the next *AnchorVL*. Also, whenever the *AnchorVL* is unable to transmit a flit, the scheduler traverses this list so that service opportunities are first offered to those virtual lanes which have received the least share of the resource. Note that either the *AnchorVL* or the virtual lane at the head of the *ActiveList* will have the lowest value of the *OpportunityCount* among all the active virtual lanes. Hence,

$$VirtualCount = min(OC_{AnchorVL}, OC_{HeadVL}) \quad (2)$$

where *HeadVL* refers to the virtual lane at the head of the *ActiveList*. Note that it can be easily verified that the *VirtualCount* function is a monotonically non-decreasing function of time.

A pseudo-code implementation of the algorithm is shown in Fig. 2, consisting of the *Initialize*, *Enqueue* and *Dequeue* routines. The *Enqueue* routine is called when a header flit arrives into an empty virtual lane buffer, i.e, when an inactive virtual lane becomes active. The *OpportunityCount* for this virtual lane is calculated using Eq. (1). The virtual lane is then added to the *ActiveList* at the appropriate position determined by its *OpportunityCount*. The *Dequeue* routine is the heart of the algorithm which schedules packets from the virtual lane buffers. The following describes the four routines used in the execution of the *Dequeue* routine during each cycle. The pseudo-code for these routines is shown in Figs. 3–6. All of these routines can be implemented as simple hardware modules.

1. *SelectAnchorVL():* If the *AnchorVL* is invalid and the *ActiveList* is not empty then the virtual lane at the head of the *ActiveList* becomes the *AnchorVL*.

```
SelectAnchorVL()
    if ( (AnchorVLValid == FALSE) and (SizeOf(ActiveList) > 0) ) then
        AnchorVL = ReadHeadVLFromActiveList();
        RemoveHeadEntryFromActiveList();
        AnchorVLValid = TRUE;
    end if
```

Figure 3. *SelectAnchorVL()* Routine

```
ServeAnchorVL()
    if (AnchorVLValid == TRUE) then
        v = AnchorVL;
        if (CanTransmitFlit(v) == TRUE) then
            Flit = FlitAtHeadOfQueue(v);
            ContinueService = FALSE;
            if (IsLast(Flit) == TRUE) then
                AnchorVLValid = FALSE;
                if (QueueIsEmpty(v) == FALSE) then
                    AddEntryToActiveList(v, OC_v);
                end if;
            end if;
            Transmit Flit;
        end if;
        Increment OC_v;
    end if;
```

Figure 4. *ServeAnchorVL()* Routine

2. *ServeAnchorVL():* If the *AnchorVL* is valid, the scheduler visits the buffer for virtual lane *AnchorVL*. If the tail flit is served from this buffer, the *AnchorVLValid* is reset to indicate that the *AnchorVL* value is no longer valid and that a new anchor needs to be chosen. After transmitting the tail flit, if the lane buffer is occupied the virtual lane is added at the appropriate position to the sorted *ActiveList*. Also since a flit has been served in the current cycle the scheduler will skip the next step and jump to step 4.

3. *ServeFromActiveList():* If the *ActiveList* is occupied and the *AnchorVL* has not transmitted a flit, service opportunities will be offered to the virtual lanes in the *ActiveList* one after the other starting at the head of the list until either a flit is transmitted from one of the lanes or the scheduler reaches the tail of the list. If a tail flit is transmitted and the corresponding virtual lane buffer is empty the virtual lane is removed from the list since it is no longer active. Note that the *OpportunityCount* of all the virtual lanes visited by the scheduler is incremented by one. If the *OpportunityCount* of the virtual lane that is able to transmit a flit exceeds that of the succeeding virtual lane, then the *ActiveList* has to

```
ServeFromActiveList()
    while ( (ContinueService == TRUE) and (Counter <= SizeOf(ActiveList)) ) then
        v = ReadVLFromActiveList(Counter);
        Increment OC_v;
        if (CanTransmitFlit(v) == TRUE) then
            Flit = FlitAtHeadOfQueue(v);
            ContinueService = FALSE;
            if ( (IsLast(Flit) == TRUE) and (QueueIsEmpty(v) == TRUE) ) then
                RemoveEntryFromActiveList(Counter);
            else if ( (Counter < SizeOf(ActiveList)) then
                if (OC_v > ReadCountFromActiveList(Counter+1)) ) then
                    sortlist == TRUE;
                end if;
            end if;
            Transmit Flit;
        end if;
        Increment Counter;
    end while;

    /* Sort the Activelist if required */
    if (sortlist == TRUE) then
        Sort(ActiveList) ;
        sortlist = FALSE;
    end if;
```

Figure 5. *ServeFromActiveList()* Routine

```
CheckEndOfBusyPeriod()
    if ( ( SizeOf(ActiveList) == 0) and (AnchorVLValid == FALSE) ) then
        VirtualCount = 0;
        for (i = 0; i < n; i = i + 1)
            OC_i = 0;
    end if;
```

Figure 6. *CheckEndOfBusyPeriod()* Routine

be re-sorted. In our simulation-based observations, during the execution of the AOQ scheduler, the percentage of cycles in which this re-sorting is required is very low. Thus, if the number of virtual lanes is large (≥ 16) and the sorting operation cannot be accomplished in hardware in one cycle, this component may be designed in the hardware module outside the critical path with only a negligible effect on fairness.

4. *CheckEndOfBusyPeriod():* Irrespective of whether a flit is transmitted or not, at the end of the cycle, the scheduler checks if there is

at least one virtual lane which is active. If none of the virtual lanes is active, the server resets the *OpportunityCount* of each lane and also the *VirtualCount*. While our pseudo-code resets these counts, a scheduler may choose to preserve these counts without resetting them in order to compensate virtual lanes that were disadvantaged in a previous busy period. This is an implementation issue to be determined based on the time-span over which virtual lanes are expected to remain active, and whether or not, when a virtual lane becomes active again it can legitimately inherit the advantages or disadvantages gained while it was active the previous instance.

In the interests of clarity of presentation, in this section, we have discussed the AOQ scheduler assuming that all the virtual lanes have been equal weights, i.e., equal rights to the output link. One may, however want some virtual channels to receive a greater share of the opportunities to access the output link than some other lanes. Let the weight associated with channel i be w_i, indicating its relative share of service opportunities. The weighted version of the AOQ scheduler is exactly similar to the AOQ scheduler described above. The only difference is that the *OpportunityCount* of each virtual lane is now normalized with respect to its weight.

4. Fairness Analysis

We use a popular measure of fairness called the *Relative Fairness Bound (RFB)*, first used in [13] and later in numerous research articles on fair scheduling algorithms. In our evaluation of the AOQ scheduler, we define the RFB in terms of the number of opportunities offered to each virtual lane, since this is the quantity that the AOQ scheduler seeks to allocate fairly among the virtual lanes. The RFB is defined as the maximum difference in the service opportunities received by any two virtual lanes over all possible intervals of time during which both lanes are active. A more rigorous definition follows.

Definition 4 *Let $S_i(t_1, t_2)$ represent the total number of service opportunities received by virtual lane i during the time interval (t_1, t_2). We define the* Relative Fairness, *$RF(t_1, t_2)$ over a time interval (t_1, t_2) as,*

$$RF(t_1, t_2) = max\,|S_i(t_1, t_2) - S_j(t_1, t_2)| \quad \forall i, j \quad (3)$$

where virtual lanes i and j are active over the interval (t_1, t_2). The RFB is defined as the maximum of $RF(t_1, t_2)$ over all possible time intervals (t_1, t_2).

We now proceed to analytically derive the fairness properties of the AOQ scheduler.

Definition 5 *Define m as the size in flits of the largest packet that is actually served during the execution of a scheduling algorithm.*

Definition 6 *Define M as the maximum number of service opportunities that may be required to serve a packet during the execution of the scheduling algorithm. Note that $M \geq m$.*

The value of M depends on a variety of complex factors such as the size of pipeline bubbles, the topology and the level of congestion. Since the value of M is not available *a priori* to the scheduler, it is important that the operation of the virtual lane scheduler be independent of M. Note that the AOQ scheduler does not require the knowledge of the total service opportunities required to transmit a packet in order to make a scheduling decision and hence satisfies the above requirement. We introduce the quantity M above, only as a means to express the upper bound on the RFB as a function of M.

Definition 7 *Let τ_1 be the time instant when a certain virtual lane i is chosen to be the* AnchorVL *by the AOQ scheduler. Virtual lane i will continue to be the* AnchorVL *until the packet in service at this virtual lane has been completely served by the scheduler. Let τ_2 be the time instant when the tail flit from this packet is transmitted. Hence virtual lane i is the* AnchorVL *during the interval (τ_1, τ_2). This time interval (τ_1, τ_2) is said to be an* anchor interval *of the AOQ scheduler.*

A busy period of a scheduler can be partitioned into a sequence of contiguous anchor intervals. Note that these intervals are mutually exclusive since only one virtual lane can be the *AnchorVL* at any time instant.

In the following, we will derive the RFB of the AOQ scheduler with help from a couple of lemmas that we prove below. Note that the *OpportunityCount* of each active virtual lane is a function of time, and

therefore, let $OC_i(t)$ represent the *OpportunityCount* of virtual lane i at the time instant t.

Lemma 1 *If (t_1, t_2) represents an anchor interval during which virtual lane avl is the* AnchorVL *and virtual lane i is in the* ActiveList *throughout this anchor interval,*

$$0 \leq S_{avl}(t_1, t_2) - S_i(t_1, t_2) \leq M, i \in ActiveList$$

Proof. In each cycle of the anchor interval (t_1, t_2), the virtual lane avl receives the first opportunity to transmit a flit. The AOQ scheduler visits the other virtual lanes in the sorted *ActiveList* only if the *AnchorVL* cannot utilize its service opportunity. Therefore, the anchor virtual lane will always receive either the same or more service opportunities than any other virtual lane during this anchor interval. This proves the lower bound on the difference between $S_{avl}(t_1, t_2)$ and $S_i(t_1, t_2)$.

During the time interval (t_1, t_2), the anchor virtual lane can transmit at most one packet and since the maximum number service opportunities that may be required to transmit a packet is M,

$$S_{avl}(t_1, t_2) \leq M$$

On the other hand it may be possible that during this time interval under consideration, flow i present in the *ActiveList* does not receive a single service opportunity. This proves the upper bound on the difference between $S_{avl}(t_1, t_2)$ and $S_i(t_1, t_2)$. ■

Let *MaxVL* and *MinVL* refer to the virtual lanes with the maximum and minimum values of the *OpportunityCount*, respectively, amongst all active virtual lanes.

Lemma 2 *At any time instant t, during an execution of the AOQ scheduler,*

$$OC_{MaxVL}(t) - OC_{MinVL}(t) \leq M$$

Proof. We will prove the lemma by induction on the anchor intervals. As a basis step, at the beginning of the first anchor interval during the execution of the scheduler, all the *OpportunityCount* values are set to zero. During this anchor interval, by Lemma 1, we know that Lemma 3 will hold true.

For the inductive step of our proof, assume that the Lemma holds at the beginning of anchor interval k. We have to now prove that it will continue to hold until the beginning of the next anchor interval $k+1$. Note that, at the start of any anchor interval, the AOQ scheduler chooses the *MinVL* as the *AnchorVL*. Since *MinVL* already has a lower *OpportunityCount* than any other virtual lane, at the end of the anchor interval, by Lemma 1, the *OpportunityCount* of *MinVL* will not exceed that of any other virtual lane by more than M. Meanwhile, since the AOQ scheduler gives opportunities to all lanes other than the anchor lane in a round robin fashion, the differences between the *OpportunityCount* values of the other lanes will remain the same. Thus, the *OpportunityCount* of no virtual lane, at the end of anchor interval k and the beginning of anchor interval $k+1$, will exceed that of another lane by more than M. ■

Theorem 1 *For any execution of the AOQ scheduling discipline, $RFB \leq 2M$.*

Proof. From Lemma 2 above, for any two virtual lanes i and j which are active at time instant t during a busy period of the AOQ scheduler,

$$|OC_i(t) - OC_j(t)| \leq M, \forall i \in ActiveList \quad (4)$$

Consider a time interval (t_1, t_2) during which virtual lanes i and j are continuously active. The relative fairness during this time interval is given by,

$$\begin{aligned} RF(t_1, t_2) &= |S_i(t_1, t_2) - S_j(t_1, t_2)| \\ &= |OC_i(t_2) - OC_i(t_1) - \\ &\quad (OC_j(t_2) - OC_j(t_1))| \\ &= |OC_i(t_2) - OC_j(t_2) + \\ &\quad OC_j(t_1) - OC_i(t_1)| \end{aligned}$$

Using Eq. (4), we have,

$$RF(t_1, t_2) \leq 2M \quad (5)$$

This proves that the relative fairness bound has an upper bound of $2M$. ■

Note that the upper bound in Eq. (5) is independent of the time interval under consideration. Theorem 1 proves the fairness of the AOQ scheduler through providing a small finite bound on the difference between the number of service opportunities received by any two virtual lanes, independent of the length of the time interval under consideration.

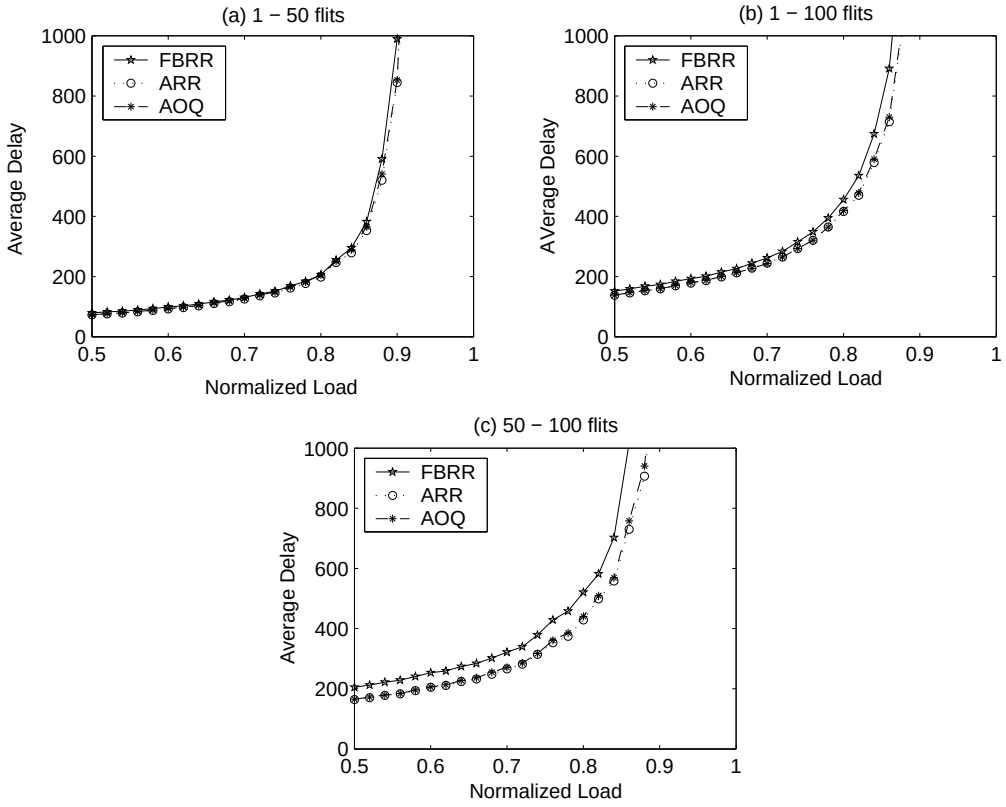

Figure 7. Average delay characteristics with 4 virtual lanes with number of flits per packet ranging from (a) 1–50 flits, (b) 1–100 flits and (c) 50–100 flits

5. Simulation Results

In this section we present simulation results on the delay characteristics of the AOQ scheduler. We use a time-driven simulator to model wormhole routing in a multi-stage interconnection network at the flit level. We use 2×2 switches with 4 or 8 virtual lanes implemented. Our goal is to study, in isolation, the effect of the scheduling strategies used at the output ports in the wormhole routers on the performance. For this reason, in spite of the improved performance obtained with dynamic sharing of input and output buffers between the various virtual lanes, our switch model uses dedicated buffers for each virtual lane at each input and output port. In our simulations, we use an 8×8 Banyan network topology. The banyan topology makes sure that there is exactly one path available from any input to any output in the network and hence each packet that enters a switch has a fixed destination output port. This allows us to isolate the impact of the scheduling strategy across the virtual lanes.

Figs. 7(a)–(c) illustrate plots of the average delay (measured in cycles) obtained with the FBRR, ARR and AOQ scheduling strategies with 4 virtual lanes per port. We use uniformly distributed packet sizes in the following three ranges: (a) 1–50 flits, (b) 1–100 flits (c) 50–100 flits.

We assume that the virtual lane scheduler dequeues one flit from one of the virtual lanes in one cycle. In order to eliminate the effect of the buffer size in relation to packet size, we use large input buffers of size

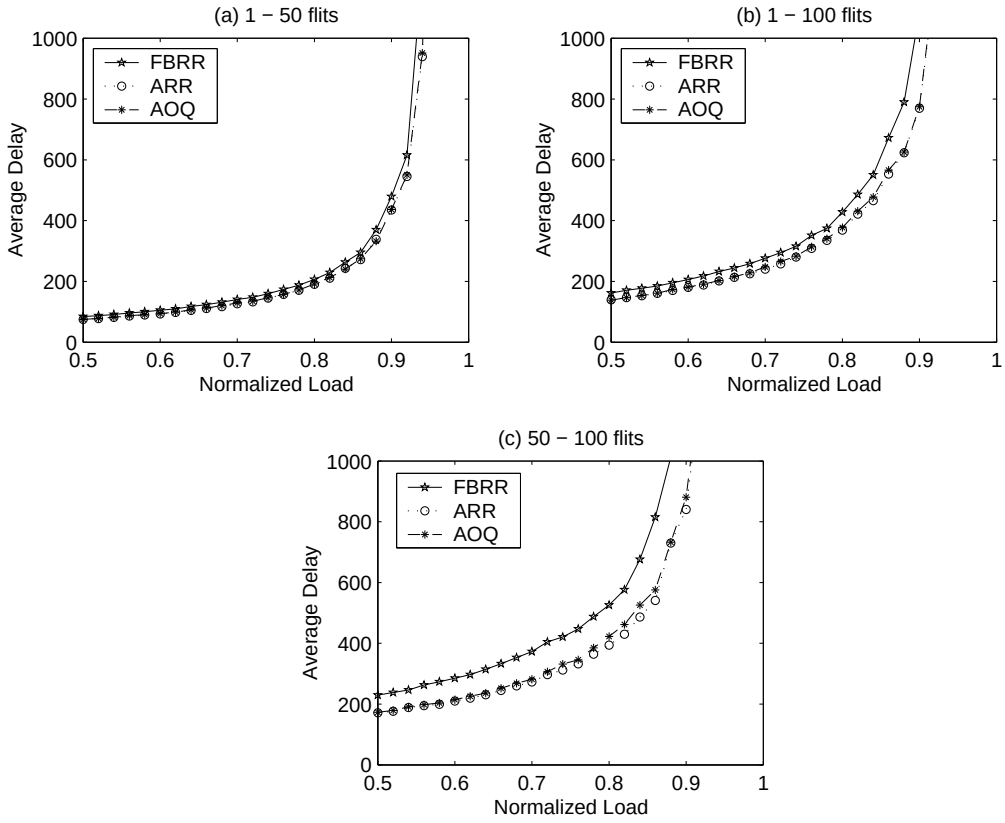

Figure 8. Average delay characteristics with 8 virtual lanes with number of flits per packet ranging from (a) 1–50 flits, (b) 1–100 flits and (c) 50–100 flits

512 flits for these plots. Note that we plot the delay only for high loads. This is because at low loads, during any given cycle, the scheduler is not likely to have any more than one virtual lane with flits ready to be scheduled. Hence the order in which flits are scheduled from the virtual lanes is the same for all scheduling disciplines. It is easily observed from the figures that both AOQ and ARR yield a lower average delay than FBRR. The improvement in delay for ARR and AOQ with longer packets is significantly more than with smaller packets. This is due to the fact that the FBRR scheduler uniformly increases the delay of all packets in proportion to the packet length.

For the results in Figs. 8(a)-(c), we repeat the above simulations with 8 virtual lanes per port. As the number of virtual lanes is increased, the delay advantage of ARR and AOQ over FBRR increases. This is because as the number of virtual lanes increases, so do the stops made by the FBRR scheduler before the same virtual lane is served again.

6. Conclusion

In this paper, we have presented a novel practical scheduling discipline called Anchored Opportunity Queueing (AOQ) for scheduling flits among the virtual lanes multiplexed on the same output link. AOQ reduces the average delay experienced by the packets in a wormhole network as compared to the traditional Flit-Based Round Robin (FBRR) scheduler. In addi-

tion, AOQ preserves the fairness and throughput characteristics of FBRR. We have analytically proved the fairness properties of AOQ, and shown that its relative fairness bound is a small finite quantity equal to $2M$, where M is the maximum number of service opportunities that may be required for a virtual channel to completely transmit one packet. Therefore, by this measure, our scheduler is at least as fair as most schedulers proposed in the literature on scheduling algorithms, while also achieving a low average delay.

The AOQ scheduler may also be used in other contexts besides wormhole switches with virtual lanes. Multi-Protocol Label Switching (MPLS) is an important new technology that is finding rapid deployment in high-end routers in the Internet [30]. In MPLS, each flow is assigned a locally unique label at every intermediate router in its path. MPLS offers a convenient technology to implement IP over ATM networks, and has been embraced by the industry as a solution at the network access points. In carrying IP over ATM, IP packets are divided into fixed-size ATM cells, each of which carries the label information. Routing and forwarding within the ATM switching network is thus similar to that in wormhole networks where each ATM cell plays the role of a flit. Thus, IP over ATM is analogous to wormhole switching, as is also described in [31]. In scheduling ATM cells for transmission over an output link, the solution presented in this paper may be readily employed to achieve fairness among the various flows while also achieving a low average end-to-end delay.

REFERENCES

1. W. J. Dally, C. L. Seitz, The torus routing chip, Journal of Distributed Computing 1 (1986) 187–196.
2. C. B. S. et al., The SP2 high-performance switch, IBM Systems Journal 34 (2) (1995) 185–204.
3. J. Beecroft, M. Homewood, M. McLaren, Meiko CS-2 interconnect elan-elite design, Parallel Computing 20 (10-11) (1994) 1627–1638.
4. Intel Corporation, Paragon XP/S Product Overview (1991).
5. Cray Research, Inc., Cray T3D System Architecture (1993).
6. ANSI, Inc., High-Performance Parallel Interface—6400 Mb/s Physical Layer (HIPPI-6400-PH) (June 1999).
7. W. J. Dally, Virtual-channel flow control, IEEE Transactions on Parallel and Distributed Systems 3 (2) (1992) 194–204.
8. S. L. Scott, G. M. Thorson, The Cray T3E network: adaptive routing in a high performance 3d torus, in: Proceedings of Hot Interconnects IV, 1996.
9. J. Carbonaro, F. Verhoorn, Cavallino: The teraflops router and NIC, in: Proceedings of Hot Interconnects IV, 1996.
10. M. Galles, Scalable pipelined interconnect for distributed endpoint routing: The SPIDER chip, in: Proceedings of Hot Interconnects Symposium IV, 1996.
11. InfiniBand^{SM} Trade Association, InfiniBand^{TM} Architecture Specification: Volume 1, Release 1.0 (http://www.infinibandta.com).
12. M. Pirvu, L. Bhuyan, N. Ni, The impact of link arbitration on switch performance, in: Proceedings of the Fifth Symposium on High-Performance Computer Architecture, Orlando, FL, 1999, pp. 228–235.
13. S. J. Golestani, A self-clocked fair queueing scheme for broadband applications, in: Proceedings of IEEE INFOCOM, Toronto, Canada, 1994, pp. 636–646.
14. H. Sethu, C. B. Stunkel, R. F. Stucke, IBM RS/6000 SP large system interconnection network topologies, in: Proceedings of the International Conference on Parallel Processing, Minneapolis, MN, 1998, pp. 620–627.
15. S. S. Kanhere, H. Sethu, A. B. Parekh, Fair and efficient packet scheduling using elastic round robin, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 324–336.
16. S. Keshav, An Engineering Approach to Computer Networks, Addison-Wesley Publishing Company, Reading, MA, 1997.
17. A. Demers, S. Keshav, S. Shenker, Design and analysis of a fair queuing algorithm, in: Proceedings of ACM SIGCOMM, Austin, TX, 1989, pp. 1–12.
18. A. K. Parekh, R. G. Gallager, A generalized processor sharing approach to flow control—the single node case, in: Proceedings of IEEE INFOCOM, Florence, Italy, 1992, pp. 915–924.
19. M. Shreedhar, G. Varghese, Efficient fair queuing

using deficit round-robin, IEEE Transactions on Networking 4 (3) (1996) 375–385.
20. J. C. R. Bennet, H. Zhang, WF^2Q: Worst-case fair weighted fair queueing, in: Proceedings of IEEE INFOCOM, Vol. 1, San Francisco, CA, 1996, pp. 120–128.
21. G. Chuanxiong, SRR: An O(1) time complexity packet scheduler for flows in multi-service packet networks, in: Proceedings of ACM SIGCOMM, San Diego, CA, 2001.
22. N. Boden, D. Cohen, R. Felderman, A. Kulawick, C. Seitz, J. Seizovic, W. Su, Myrinet: A gigabit-per-second local area network, IEEE Micro 15 (1) (1995) 29–36.
23. J. Duato, S. Yalamanchili, L. Ni, Interconnection Networks, Morgan Kaufmann Publishers, San Francisco, CA, 2002.
24. H. Sethu, H. Shi, S. S. Kanhere, A. B. Parekh, A round-robin scheduling strategy for reduced delays in wormhole switches with virtual lanes, in: Proceedings of the International Conference on Communications in Computing, Las Vegas, NV, 2000, pp. 275–278.
25. A.-H. Smai, D. K. Panda, L.-E. Thorelli, Prioritized demand multiplexing (PDM): A low-latency virtual channel flow control framework for prioritized traffic, in: Proceedings of the International Conference on High Performance Computing, San Antonio, TX, 1997, pp. 449–454.
26. R. Klamann, Opportunity scheduling: An unfair CPU scheduler for UNICOS, in: Proceedings of the Cray User Group, San Jose, CA, 1996, pp. 164–170.
27. D. Stiliadis, Traffic Scheduling in Packet-Switched Networks: Analysis, Design and Implementation, University of California at Santa Cruz, 1996.
28. S. Keshav, On the efficient implementation of fair queueing, Journal of Internetworking Research and Experience 2 (1991) 3–26.
29. P. Goyal, H. M. Vin, H. Cheng, Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks, IEEE/ACM Transactions on Networking 5 (5) (1997) 690–704.
30. B. S. Davie, Y. Rekhter, MPLS: Technology and Applications, Morgan Kaufmann Publishers, San Francisco, CA, 2000.
31. M. G. H. Katevenis, I. Mavroidis, G. Sapountzis, E. Kalyvianaki, I. Mavroidis, G. Glykopoulos, Wormhole IP over (connectionless) ATM, IEEE/ACM Transactions on Networking 5 (9) (2001) 650–661.