A Router Architecture for QoS Capable Clusters

by

Madhusudhanan Anantha Subramanian

A PROJECT

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed OCTOBER 23, 2003
Commencement June 2002

AN ABSTRACT OF THE PROJECT OF

Madhusudhanan Anantha Subramanian for the degree of Master of Science in Computer Science presented on OCTOBER 23, 2003.

Title: A Router Architecture for QoS Capable Clusters

Abstract approved: _____

Bella Bose

Interconnection Networks have been used as a high performance communication fabric in parallel processor architectures. Parallel processors built using off-the-shelf components, called *clusters*, are becoming increasingly attractive because of their high performance to cost ratio over parallel computers

Many web servers and database servers make efficient use of clustering from cost, scalability and availability standpoints. The Design of high performance cluster networks with QoS guarantees is becoming increasingly important to support a variety of multimedia applications, many of which have real time constraints. Most commercial routers, which are based on the wormhole switching paradigm, can deliver high performance, but lack QoS provisioning. A new router architecture with support for QoS provisioning was introduced in [1]. In this project we present a detailed analysis of the hardware complexity of the router in [1] and propose some architectural modifications to reduce the hardware complexity of the router. We have also developed a simulator to compare and analyze the performance characteristics of the proposed router architecture.

Master of Science project of Madhusudhanan Anantha Subramanian presented on OCTOBER 23, 2003

APPROVED:

Major Professor, representing Computer Science

Chair of the Department of Computer Science

Dean of the Graduate School

I understand that my project will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my project to any reader upon request.

Madhusudhanan Anantha Subramanian, Author

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my major professor, Dr. Bose, for his valuable ideas which put me in the right direction. He was also instrumental in showing me how analyze and work with a research problem.

I would like to express my sincere gratitude to Dr. Minoura, for his valualble help in solving some of the programming issues with the simulator and serving in my committee.

I would also like to express my sincere gratitude to Dr. Cull, for serving in my committee.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

A ROUTER ARCHITECTURE FOR QOS CAPABLE CLUSTERS

CHAPTER 1

INTRODUCTION

Parallel computers with multiple processors form the computational hardware which are able to deliver high performance (tera flops). In parallel computers, several processors cooperate to solve a large problem. Parallel computers with direct interconnection networks provide scalable processing power.

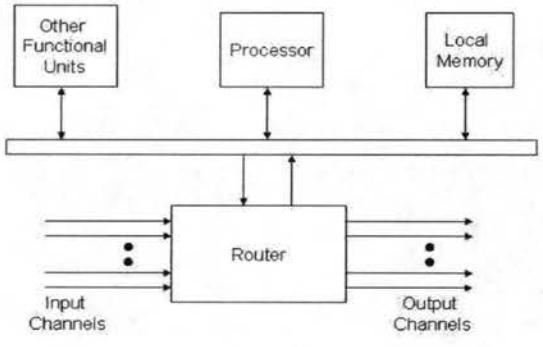

FIGURE 1.1: A generic node architecture

A direct network consists of a set of *nodes*, each one directly connected to a (usually small) subset of other nodes in the network. Each node is a pro-

grammable computer with its own processor, local memory and other supporting devices. These nodes may have different functional capabilities. Figure 1.1 shows the architecture of a generic node. A common component of these nodes is a *router*, which handles message communication between nodes. For this reason, direct networks are also referred to as *router-based networks*. Each router has direct connections to the routers of its neighbours. Usually two neighbouring nodes are connected by a pair of unidirectional channels in opposite directions. As the number of nodes in the system increases, the processing power, memory bandwidth, and total communication bandwidth also increase.

Clusters are becoming increasingly popular because of their high performance to cost ratio over parallel computers. Cluster systems are thus becoming more attractive for designing scalable servers with switched network architectures that offer much higher bandwidth than broadcast-based networks. Quality-of-Service (QoS) provisioning in such clusters is becoming a critical issue with the widespread use of these systems in diverse commercial applications. For example, web servers and database servers make efficient use of clustering technology from cost, scalability, and availability standpoints. However, there has been a tremendous surge in dynamic web content, multimedia objects and other web-enabled applications which require QoS guarantees in different connotations. These demands in turn are passed on to the building blocks of the interconnects, the switching fabrics or routers.

A router architecture with QoS capabilities is proposed in [1]. In this project we study the hardware complexity of the proposed router and propose some architectural modifications to reduce the hardware complexity without sacrificing performance. We use simulation to compare the performance characteristics of the new architecture with that of the existing architecture.

In the next chapter we give some background on network toplogies, basic switching mechanisms and routing fabrics. Then, in Chapter 3, we describe architecture of the QoS capable router presented in [1]. In Chapter 4 we analyze the hardware complexity of the router presented in Chapter 3. Chapter 5 describes the modifications to the router architecture. In chapter 6 we present an analytical model to estimate the performance of the proposed router. Then, in Chapter 7, we describe the simulation frame work and compare the performance characteristics of the modified router with the existing router. We summarize the conclusions and future work in the last chapter.

CHAPTER 2

BACKGROUND

2.1 Popular Network Topologies

Many network topologies have been proposed in terms of their graph theoretical properties. Most of the implemented networks have an orthogonal topology. A network topology is *orthogonal* if and only if nodes can be arranged in an orthogonal n-dimensional space, and every link can be arranged in such a way that it produces a displacement in a single dimension. In a *strictly orthogonal* topology, each node has at least one link crossing each dimension. In a *weakly orthogonal* toplogy , some nodes may not have a link in some dimensions.

The most interesting property of strictly orthogonal topologies is that routing is very simple. Effectively, in a strictly orthogonal topology, nodes can be numbered using their coordinates in the n-dimensional space. Since each link traverses a single dimension and every node has at least one link in each dimension, the distance between two nodes can be computed as the sum of dimension offsets.

The most popular networks are the *n-dimensional mesh*, the *k-ary n-cube* or *torus*, and the *hypercube*. Formally, an n-dimensional mesh has $k_{n-1} \times k_{n-2} \times \cdots \times k_1 \times k_0$ nodes with k_i nodes in each dimension i, where $k_i \geq 2$ and $0 \leq i \leq n-1$. Each node is identified by n coordinates, $(x_{n-1}, x_{n-2}, \cdots, x_1, x_0)$, where $0 \leq x_i \leq k_i - 1$ for $0 \leq i \leq n-1$. Two nodes X and Y are neighbours if and

only if $y_i = x_i$ for all $i, 0 \leq i \leq n-1$, except one, j, where $y_j = x_j \pm 1$. Thus the nodes have n to $2n$ neighbours, depending on their location in the mesh. A topology in which each node has the same number of neighbours is said to be *regular*. Therefore, this topology is not regular.

In a bidirectional k-ary n-cube, all nodes have the same number of neighbours. The definition of a k-ary n-cube differs from that of a n-dimensional mesh in that all of the k_i are equal to k and two nodes X and Y are neighbours if and only if $y_i = x_i$ for all $i, 0 \leq i \leq n-1$, except one, j, where $y_j = (x_j \pm 1) \mod k$. The change to modular arithmetic in the definition adds the wraparound channels to the k-ary n-cube, giving it regularity and symmetry. Every node has n neighbours if $k = 2$, and $2n$ neighbours if $k > 2$. When $n = 1$ the k-ary n-cube collapses to a bidirectional ring with k nodes.

Another topology with regularity and symmetry is the hypercube, which is a special case of both n-dimensional meshes and k-ary n-cubes. A hypercube is an n-dimensional mesh in which $k_i = 2$ for $0 \leq i \leq n-1$, or a 2-ary n-cube.

Figure 2.1(a) depicts a 3×3 mesh , Figure 2.1(b) depicts a 3×3 torus and Figure 2.1(c) depicts a 3 dimensional hypercube.

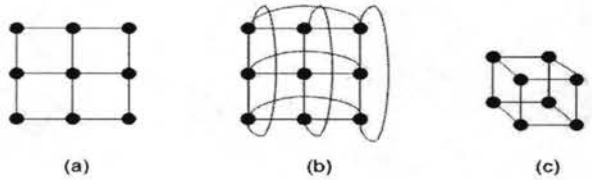

FIGURE 2.1: Strictly Orthogonal Network Topologies: (a) 2-D 3×3 mesh, (b) 2-D 3×3 torus and (c) 3-D hypercube

2.2 Basic Switching Techniques

Interprocessor communication can be viewed as a hierarchy of services starting from the physical layer. We find it useful to distinguish between the three layers in the operation of an interconnection network: the *routing layer*, the *switching layer* and the *physical layer*. The physical layer refers to link-level protocols for transferring messages. The switching layer utilizes the physical layer protocols for implementing mechanisms for forwarding messages through the network. Finally, the routing layer makes routing decisions to determine candidate output channels at intermediate router nodes.

In order to compare different switching techniques we will consider the latency of an L-bit message in the absence of any traffic. The phit size and flit size are equivalent and are equal to the physical channel bandwidth of W bits. The routing header is 1 flit; thus the message size is $L + W$ bits. A router can make a routing decision in t_r seconds. The physical channel operates at B Hz; that is, the physical channel bandwidth is BW bits per second. We assume that the channel wires are short enough to complete transmission in 1 clock cycle. Therefore, the propagation delay is denoted by $t_w = \frac{1}{B}$. Once a path has been set up the intra-router delay is t_s. The source and destination processors are assumed to be D links apart.

2.2.1 Circuit Switching

In *circuit switching*, a physical path from the source to the destination is reserved prior to the transmission of data. A routing header flit called a *routing probe*, which contains the destination address and some control information, is sent. The probe sets up the path at the intermediate routers. When it reaches the

destination the complete path is set up and an acknowledgement is sent back to the source. The message contents can now be transmitted through the router at full bandwidth of the hardware path. The circuit may be released by the destination or the last few bits of the message.

The latency of transmission of a message in a network which uses circuit switching can be calculated as follows:

$$t_{circuit} = t_{setup} + t_{data} \tag{2.1}$$

here,

$$t_{setup} = D[t_r + 2(t_s + t_w)] \tag{2.2}$$

$$t_{data} = \frac{1}{B}\left\lceil \frac{L}{W} \right\rceil \tag{2.3}$$

where t_r = time to take the routing decision, t_s = propagation delay through the router and t_w = propagation delay of an inter-router link.

2.2.2 Packet Switching

In circuit switching, the complete message is transmitted after the circuit has been set up. Alternatively, the message can be partitioned and transmitted as fixed-length packets. The first few bytes of a packet contain routing and control information and are referred to as a *packet header.* Each packet is individually routed from source to destination. This technique is referred to as *packet switching.* A packet is buffered completely at each intermediate node before it is forwarded to the next node. This is the reason why this switching technique is referred to as *store and forward* (SAF) switching. The header information is extracted by the intermediate router and used to determine the output link over which the packet is to be forwarded.

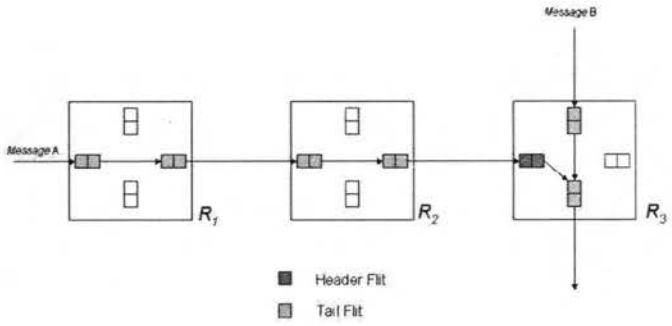

FIGURE 2.2: An example of blocked wormhole-switched message.

The latency experienced by the packet is proportional to the distance between the source and destination nodes. Thus the base latency of a packet can be computed as follows:

$$t_{packet} = D\left\{t_r + (t_s + t_w)\left\lceil \frac{L+W}{W} \right\rceil\right\} \tag{2.4}$$

2.2.3 Wormhole Switching

The need to buffer complete packets within a router can make it difficult to construct small, compact and fast routers. In *wormhole switching* , message packets are pipelined through the network. A message is broken up into flits. The *flit* is the unit of message flow control, and input and output buffers at routers are typically large enough to store a few flits. The message is pipelined through the network at flit level and is typically too large to be buffered within a router. Thus, at any instant a blocked message occupies buffers in several routers.

The base latency of a wormhole switched message can be computed as follows:

$$t_{wormhole} = D(t_r + t_s + t_w) + max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \tag{2.5}$$

2.2.4 Comparison of Switching Techniques

The evolution of switching techniques was influenced by the need for better performance. Wormhole switching used techniques like reduced buffering and fine-grained pipelining of message transmission to provide good performance over circuit switching and store-and-forward switching. In wormhole switching the message is buffered across routers, precluding access to network bandwidth to other messages. Thus, while the average message latency can be low, the network saturates at a fraction of the maximum available bandwidth, and the variance of message latency can be high. In packet switching, messages are completely buffered in the routers. As a result, the messages consume network bandwidth proportional to the network load.

2.3 Router Architectures

2.3.1 Generic Router Architecture

The architecture of a generic router is shown in Figure 2.3 and is comprised of the following major components.

- *Buffers.* These are the first-in first-out (FIFO) buffers for storing messages in transit. In the model shown in Figure 2.3, a buffer is associated with an input and output physical channel. The buffer size is an integral number of flow control units

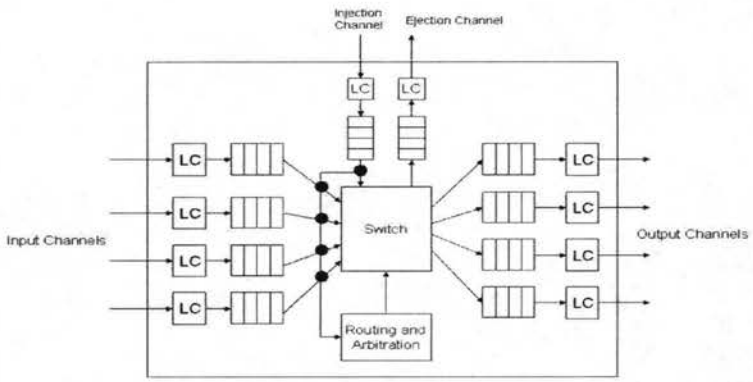

FIGURE 2.3: Generic router model.(LC = Link Controller)

- *Switch.* This component is responsible for connecting router input buffers to the output buffers. High speed routers will utilize crossbar networks with full connectivity, while lower speed implementations may utilize networks that do not provide full connectivity between the input and output buffers.

- *Routing and Arbitration unit.* This component implements the routing algorithms, selects the output link for an incoming message, and accordingly set the switch. It arbitrates multiple requests to the same output link. If the requested buffer is busy the message waits in the input buffer till it is free.

- *Link Controllers(LCs).* The flow of messages across the physical channel between adjacent routers is implemented by the link controller. The link controllers on either side co-ordinate to transfer units of flow control.

- *Processor Interface.* This component implements a physical channel interface to the processor rather than to the adjacent router. It consists of one or more injection channels from the processor and one or more ejection channels to the processor.

2.3.2 Baseline Router Model

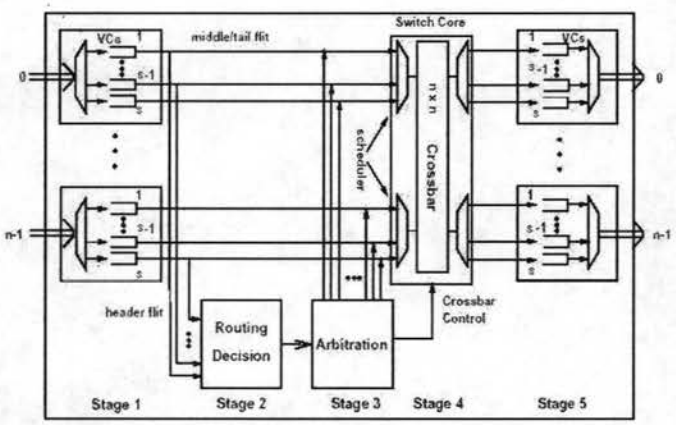

FIGURE 2.4: A five-stage pipelined router model

Figure 2.4 shows the baseline router model used for the proposed modifications. The router uses a pipelined design to minimize network cycle time. The router has n(n is the number of dimensions in the network) input ports and n output ports.

- Stage 1 of the pipeline represents the functional units that synchronize an incoming flit, de-multiplex the flit so that it can go to its corresponding

Virtual Channel (VC).

- Stage 2 represents the *routing unit* which makes decisions on forwarding the flit to a specific output port. This stage is bypassed by data and tail flits.

- Stage 3 represents the *arbitration unit* that reserves the Crossbar Ports and the Output Virtual Channel Buffers. This stage is also bypassed by data and tail flits.

- The *crossbar* in Stage 4 performs the flit forwarding to the corresponding output port. Stage 4 also contains a rate-based scheduler which selects packets from flows for transmission each cycle.

- Stage 5 contains the output virtual channel buffer from which the *output link controller* picks flits for transmission to the subsequent router

CHAPTER 3

PREVIOUS WORK

In this chapter we will discuss the modifications to the pipelined router model presented in the last chapter to enable QoS provisioning.

3.1 Rate-Based Scheduling for QoS Support

The most intuitive way to support QoS provisioning, in wormhole switched networks, is to add a rate based scheduling scheme and to attach a notion of priority to flows on the basis of which the router selects flits for transmission in a given clock cycle. In this case a Virtual Clock algorithm [5] based scheduler is used, which picks flits from flows on the basis of flow priority and maximum allowed bandwidth.

3.2 Router Design Alternatives

Traditional routers are called *non-preemptive* routers because they statically divide the *virtual channels* (VC) among the traffic classes. This restricts the flexibilty of the router to handle traffic changes. The proposed solution to this problem is the development of a *preemptive* router, where several classes of traffic with different priorities share the same VC, with the provision that a higher priority message can *preempt* a lower priority message. The preemptive

model can dynamically allocate a given VC to any traffic class. Hence it is more suitable for handling dynamic workloads.

3.2.1 Preemption in the Input Buffer

The additional hardware required for preemption at any input buffer (VC) include an extra buffer (Figure 3.1) of size (s - 1) where s is the total number of priority levels, and a history stack of the same size. The extra input buffer is used for diverting higher priority messages when the regular VC is occupied by a lower priority message. If the input buffer is occupied by a higher priority message, a lower priority message is not allowed to use the extra buffer, and it is blocked behind the higher priority message. On the other hand, if the input buffer is used by a lower priority message, a higher priority message is sent to the extra buffer so that it can subsequently preempt the lower priority message in stage 1 of the router.

The flit-preemption process is outlined as follows:

- In stage 1, when the extra buffer has a header flit from a higher priority message, the input buffer preemption begins

- The router checks to see if the tail flit of the lower priority message has passed through stage 1. If not, a *dummy tail flit* is created for the preempted message. A dummy tail flit does not carry any payload, but behaves as a normal tail flit releasing all resources held by the low priority message.

- The routing information of the message is stored in the history stack to be used later for making a dummy header flit to resume transmission of

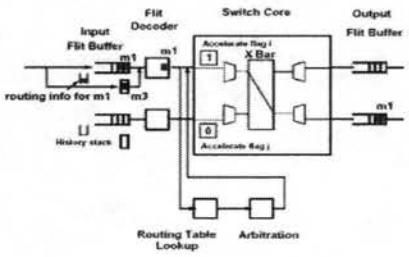

FIGURE 3.1: A higher priority message (m3) is blocked and needs to preempt a lower priority message (m1) in the input buffer using the extra buffer

the preempted message.

- During preemption, the remaining flits of the preempted message remain in the input buffer.

3.2.2 A Flit Acceleration Mechanism

When the input buffer preemption starts, there could be remaining flits of m1 between the flit decoder and the input port of the crossbar as shown in Figure 3.1. In addition, when the header flit of m3 tries to reserve the output VC, it could be already occupied by another low priority message like m2 in Figure 3.2. In both cases the lower priority flits will slow down the progress of m3, until these flits are pushed out of the output buffer.

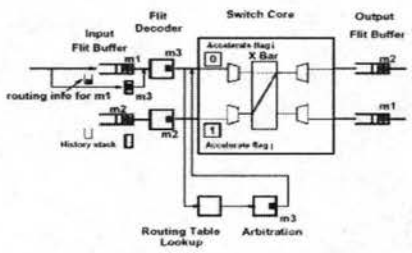

FIGURE 3.2: When the header flit of m3 tries to reserve the Output VC, it is already occupied by another lower priority message m2. The accelerate flag of the Input VC of m2 is set to expedite the flow of the remaining flits.

Therefore, we use a flit acceleration mechanism that helps expedite the delivery of flits of such lower priority messages (like m1 and m2) by assigning a low virtual clock value to them. This guarantees that these messages will be selected in the next cycle unless there are other preempted messages at other VCs. For this purpose a flag called *accelerate* is associated with each input VC. This flag is set until the tail flit of the preempted message (m1) or expedited message (m2) passes the crossbar.

CHAPTER 4

HARDWARE COST ANALYSIS

The architectural modifications proposed in Chapter 3 incur a cost in implementation. In this section we will present the harware complexity analysis of the *flit preemption logic* and the *flit acceleration logic*. We will use this analysis to investigate the possibility of modifications to the architecture to reduce the hardware cost without sacrificing performance. Throughout the analysis n is the number of dimensions in the interconnect and s is the number of prioritized flow classes. The comlexity is expressed in terms of number of gates used for a typical implementation.

4.1 Cost Analysis of the Flit Preemption Unit

Figure 4.1 shows a schematic of the flit preemption unit. The flit preemption unit has to do the following tasks in a clock cycle.

- check the extra buffer to see if there is a header flit.
- if there is a header flit, check whether this flit can preempt a flow in the virtual channel buffer.
- if the tail flit of the preempted flow has already been received in which case a dummy tail flit is not created.

- it is clear that at any given clock cyle there is only one buffer with a header flit in the extra buffer that needs to be processed.

The hardware complexity of the *flit preemption logic* scales in the order of the number of prioritized flow classes. We will have to calculate the hardware complexity of the *flit preemption logic*, the *history stack* and the *extra buffer* to compute the effective growth in hardware complexity of the flit preemption unit.

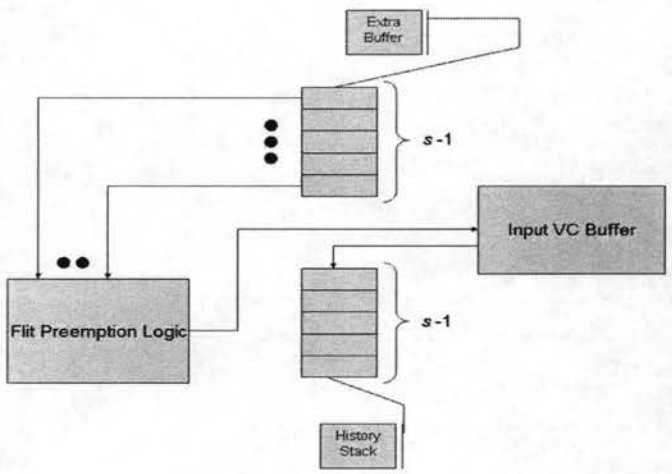

FIGURE 4.1: Flit Preemption Unit. This block is repeated for each input dimension in the Interconnect

We can compute the hardware complexity of each of these components as follows.

- *flit preemption logic* $\Rightarrow n \times s+c$
- *extra buffer complexity* $\Rightarrow n \times s+c$

- *history stack complexity* $\Rightarrow n \times s{+}c$

- *flit preemption unit complexity* $\Rightarrow 3(n \times s{+}c) \Rightarrow O(s)$ (because $s \geq n$)

where c is a constant.

4.2 Cost Analysis of the Flit Acceleration Unit

Figure 4.2 shows a schematic of the flit acceleration unit. The flit accelerarion unit has to do the following.

- Check to see if any lower priority flow is occupying an output buffer. There are $(n \times s) - 1$ possible channels that have to be checked for a given channel.

- if there exists such a buffer then the accelerate flag is set in the buffer.

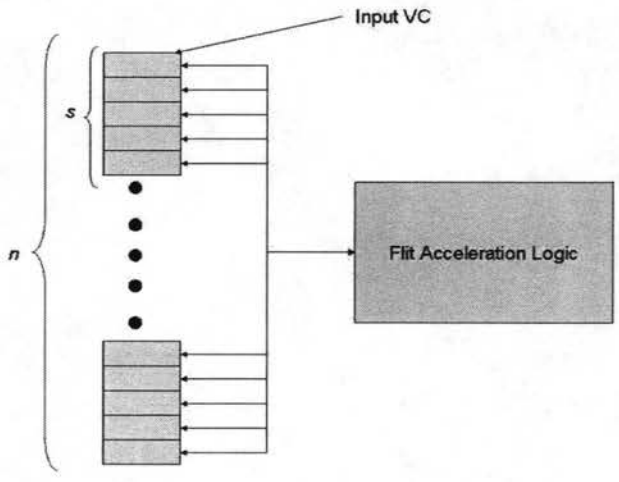

FIGURE 4.2: flit acceleration unit

The effective complexity in the Flit acceleration unit is in the *flit acceleration logic*. Therefore it will suffice to compute the hardware complexity of the flit acceleration logic.

flit unit complexity $= (n \times s)(n \times s) \Rightarrow \mathrm{O}(s^2)$(because $s \geq n$)

Thus we see that the effective complexity of the flit acceleration unit is higher than that of the flit preemption unit. With this result we will propose some modifications to the router to reduce the hardware complexity.

CHAPTER 5

PROPOSED MODIFICATIONS

From the previous chapter it can be seen that hardware complexity of the flit preemption unit is *linear* in the number of prioritized flow classes but the hardware complexity of the flit acceleration unit is *quadratic* in the number of prioritized flow classes. In this section we will propose modifications to the router architecture to replace the flit acceleration logic with lesser complexity functional units which perform the same function. Figure 5.1 shows the location of placement of the proposed functional units in the router.

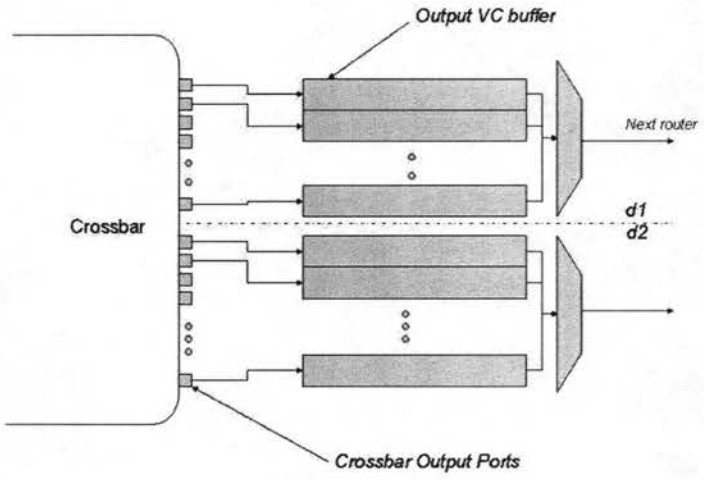

FIGURE 5.1: *Router Back End.*(the multiplexors represent the link schedulers)

5.1 Buffer Status Aware Link Scheduling

Figure 5.2 shows the operation of an asynchronous flow control protocol between two routers. The sequence of operations are as follows:

- Router 1 makes the request(RQ) line high to request permission to transmit a flit.
- Router 2 sets the acknowledge(ACK) line to represent buffer availability and hence permission to transmit.
- Router 1 begins transmission if the ACK line has been set to allow transmission of a flit. In which case the flit is transmitted as a series of *phits.*

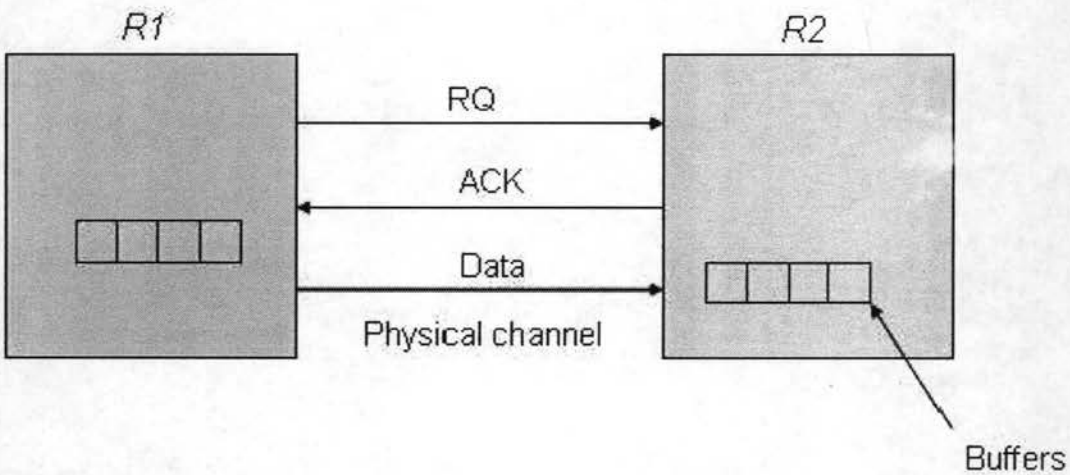

FIGURE 5.2: An example of an asynchronous physical channel flow control

A näive *priority based link scheduler* designed to transmit flits on the basis of priority can waste bandwidth and hence increase the blocking time at the

output VC buffers if the buffer at the next router is full and remains so because of the presence of higher priority flows in the neighbouring router. Better utilization and hence a higher probability of buffer availability can be achieved by sending flits from other flows whose buffers are *not* full. We will use this reasoning to propose improved link scheduling algorithms. We will also present the hardware complexity analysis for each algorithm. The main charateristic of these algorithms is that they *store the last received ACK status* of each flow in the output VC buffer.

5.1.1 Highest Non N-Ack Flow

The algorithm is as follows:

- Get the list of channels ready for transmission at a given clock cycle $\{S\}$ set of all channels which can be scheduled $\{R\}$ set of ready channels $\{S\} \leftarrow \{R\}$
- Pick the channel which has the
  1. *Highest* Flow class.
  2. Does not have an *N-Ack* (Negative Acknowledgement).
  3. Choose the *first such channel* in case of a tie.
- Read the flit from this channel and record the ACK status for the next iteration.

Discussion: Figure 5.3 shows a hardware implementation of the link scheduling algorithm.The algorithm uses a tree shaped circuit to pick the "winner" channel at a given clock when its starts flit transmission. At each clock cycle a flit from

the selected channel is transmitted and the ACK status of the transmission is stored for use in a subsequent transmission.

Note that only one bit is added to the lowest level processing units because this level "filters" all channels that are ready and the subsequent levels work by picking a flow to *highest priority* among the ready channels. Thus the only addition in hardware complexity is the *O(s)* bits.

The **disadvantage** of this algorithm is that a high priority flow class which received an N-ACK might not be picked for a long time if there are enough flits from other flows. In essence, a high priority flow might "starve". This disadvantage becomes crucial in heavily loaded networks.

A simple fix to this problem lies in the introduction of new variables.

1. *cycle_wait* which keeps track of no. of cycles after receiving a N-ACK and is updated each clock cycle.

2. *max_count*, maximum value after which *cycle_wait* update should stop.

3. *cycle_wait* is *initialized/reset* after receiving a N-ACK after *max_count* was reached and a re-transmission was tried.

5.1.2 Modified Highest Non N-Ack Flow

The modified Highest Non N-ACK sheduling algorithm is described as follows.

- Get the list of channels ready for transmission at a given clock cycle $\{S\}$ set of all channels which can be scheduled $\{R\}$ set of ready channels $\{S\} \leftarrow \{R\}$

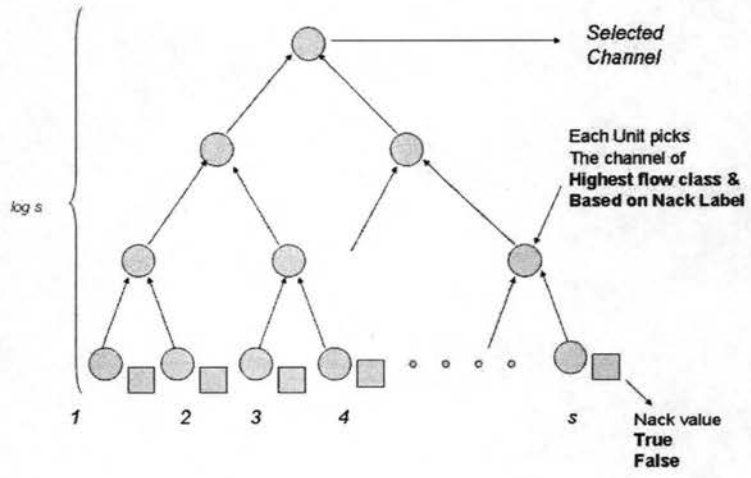

FIGURE 5.3: Hardware implementing Highest Non N-ACK Scheduling

- Pick the channel of the *Highest Flow class*, among Ready Channels, that does not have a N-ACK or whose counter has reached *max_count*.
- Choose the *first such channel* in case of a tie.
- Read the flit from this channel and record the ACK status for the next iteration.

Discussion: Figure 5.4 shows the hardware implementation of the modified version of the algorithm.At each clock cycle a flit from the selected channel is transmitted and the ACK status of the transmission is stored for use in a subsequent transmission.

Note that one counter and a register of size *log(s)* are added to the lowest level processing units because this level "filters" all channels that are ready and have the required counter constraints. The subsequent levels work by picking a flow of the *highest priority* among the channels selected at level 1. Thus the only

addition in hardware complexity is the *O(s.log(s))* bits. Therefore the hardware addition is small.

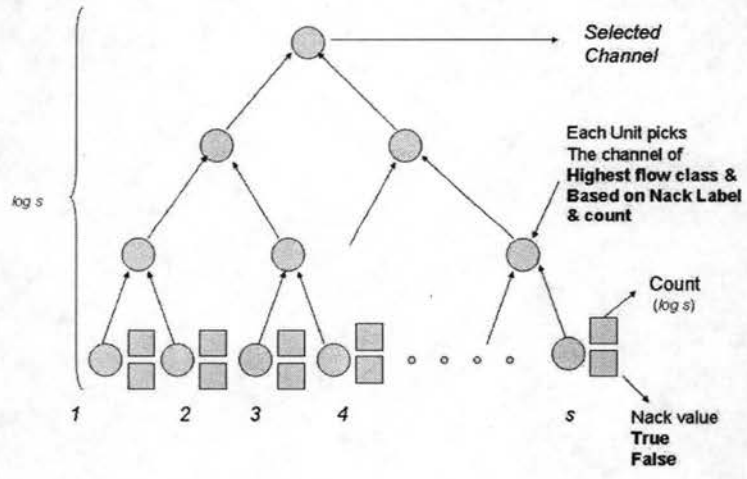

FIGURE 5.4: Hardware implementing the Modified Highest Non N-ACK Scheduling

5.2 Flexible Output Virtual Channel Allocation

In this section we discuss the next modification to the router. The scheduling algorithms discussed above assure a *free virtual channel* with a *high probability.* The main problem is that the algorithms *do not guarantee* whether a *specific virtual channel* is free. The main disadvantage of using the link scheduling algorithms in a configuration as shown in Figure 5.5 is that we might not be able to allocate virtual channels even though there are other free virtual channels in the VC buffer.

We therefore, propose a dynamic virtual channel allocation scheme which can flexibly allocate a free virtual channel from the VC buffer. Figure 10 shows a schematic of the proposed modification.

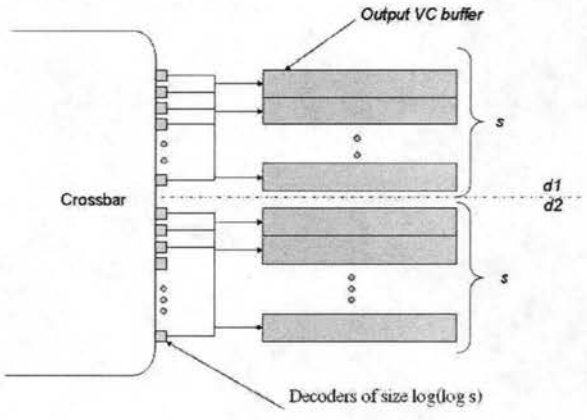

FIGURE 5.5: Flexible channel allocation at the Output VC Buffer

The hardware addition to the router is a *channel identifier* at each crossbar output port which contains the address of the channel allocated to this port. Each output port is allocated a specific set of channels it can choose from and the size of this identifier is $\log(\log s)$ and there are multiplexer circuits of complexity $\log s$ giving an overall gate complexity of $s.\log s$.

5.3 Analysis

Figure 5.6 shows the router architecture with both of the modifications in place. These changes are valid in the context of replacing the *flit acceleration unit*

because the main reason for introducing acceleration is to avoid the *blocking of high priority flows* due to other lower priority flows. Since the link scheduling algorithms work by maximizing the number of flits transmitted in a time frame and since the probability of servicing high priority flows is high, the probability of availability of a virtual channel is high. With a flexible channel allocation scheme to take advantage of the high availability of free channels the probability of blocking of a high priority flit is minimized. Therefore we can see that this combination of modifications replaces the flit acceleration unit functionally.

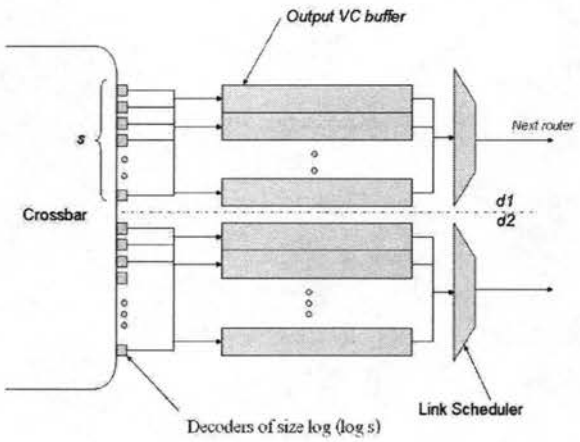

FIGURE 5.6: Modified Router Architecture

The effective hardware complexity of the modifications is the total of the hardware complexities of the proposed modifications.

$Total\ Complexity = O(\text{s.log(s)}) + O(\text{s.log(s)}) \Longrightarrow O(\text{s.log(s)})$

This hardware complexity bound is therefore better than that of the flit acceleration unit ($O(\text{s}^2)$) we have replaced.

CHAPTER 6

AN ANALYTICAL MODEL

In this section we will develop a mathematical model for deadline missing probability in a single router and extend the model to calculate the deadline missing probability of a message which traverses r routers to its destination. As described in Chapter 2 , the router model assumes a pipelined architecture with 5 stages and augmented with the modifications proposed in chapter 5. The model is described for S classes consisting of $(S-1)$ classes of real time traffic and one class of best-effort traffic.

Note that a message entering the pipelined router (Figure 2.4) can experience delay at stages 1, 3 and 5. If the input VC buffer is full in stage 1, the message must wait outside the router until adequate space is available. In stage 3, the message again might be delayed because the destination crossbar ports are full. Crossbar output port arbitration is performed at a message level granularity. So the message has to wait until the required port is released by the message already holding it. Finally in stage 5, multiple virtual channels compete for the physical channel bandwidth. This is the delay experienced due to the link scheduler. The model is based on the following assumptions:

1. The arrival Pattern of each class s follows a Poisson process with an average arrival rate λ_s^g .

2. Message length is M flits long.

3. Message destination is uniformly distributed.

4. The input and output virtual channel buffers in stages 1 and 5 can hold b_s flits.

5. Each class is assigned separate injection/ejection queues outside the router, and these have infinite capacity.

The average message latency of a message of class s $(1 \leq s \leq S)$ is composed of the average network latency, $\overline{L_s}$, which is the time to traverse the router (network), and the average waiting time, $\overline{W_s}$, at the injection channel. Thus,

$$\overline{Latency_s} = \overline{L_s} + \overline{W_s} \tag{6.1}$$

In this project, we will only consider the network latency. The average network latency of a message of class s consists of two parts. The first part is the actual message transfer time, T. The second part is due to blocking caused by the wormhole switching scheme, and due to sharing of the physical channel bandwidth by multiple virtual channels at stage 5 (Figure 2.4). The actual transmission time of a M flit message in a router with P pipe stages is $M + (P - 1)$.

In order to calculate the second part of the network latency , let us define $\overline{B_s}$ as the average blocking length (in number of flits) seen by the header flit at the input, output and arbitration stages of the router. $\overline{B_s}$ captures the message blocking in a pipelined wormhole router. Then the effective length of the message becomes $(M + \overline{B_s})$ flits. Let $\overline{S_s}$ be the average number of cycles required to transfer one flit of a class s message. $\overline{S_s}$ represents the effect of bandwidth sharing mechanism of the *Virtual Clock* Algorithm. Thus, the average nework

latency ($\overline{L_s}$) for $1 \leq s \leq S$ is

$$\overline{L_s} = P - 1 + (M + \overline{B_s})\overline{S_s} \tag{6.2}$$

Since blocking occurs in stages 1,3 and 5 as discussed above, the average blocking length can be separated into three parts as

$$Input = P[\text{input buffer is occupied}].\{M/2\} \tag{6.3}$$

$$Arbiter = P[\text{Arbiter is busy}].\{M/2\} \tag{6.4}$$

$$LC = P[\text{Ouput Buffer is full}].\{M/2\} \tag{6.5}$$

where *Input*, *Arbiter* and LC represent the corresponding blocking lengths at stages 1,3 and 5. In expressions 6.3, 6.4 and 6.5, the first term represents the probability that the corresponding buffer is not empty, and the second term is the average message length that will be affected due to blocking. For example, if the input buffer is not empty, the header flit will face an average delay of $M/2$ flits.

In order to calculate the average blocking length we will have to calculate the probability that the input buffer is full, the probability that the output buffer is full, and the delay due to bandwidth sharing. We can calculate these terms as follows.

6.1 Average Blocking Length in Stage 1

The router uses a *preemptive model* of virtual channel allocation. The preemptive model can dynamically allocate any virtual channel to any traffic class. Assuming a buffer size of b_s, we have to consider the following situations when a header flit of flow s reaches a router.

1. the input virtual channel buffer *is not* full, then there is no delay at the input and *any* free channel can be assigned to this flow.

2. the input virtual channel buffer is full, but if there is *at least one flow* which has a lower priority than this flow, then this flow can interrupt the lower priority flow and no blocking occurs.

3. the input virtual channel buffer is full and there are no flows whose priority is lesser than this flit. In this case the this flit will have to wait for channel to be released and hence a delay is incurred.

we can clearly see that only in case 3 does the header incurs a blocking delay. Therefore the blocking delay is

$$delay1_s = P[\text{buffer is full with no flows priority} < s].\{M/2\} \quad (6.6)$$

$$P_s[\text{full}] = P[\text{each channel has a flow} \geq s] \quad (6.7)$$

$$= \prod_{i=0}^{S-1} \sum_{j=s}^{S-1} P[X_j] \quad (6.8)$$

where, $P_s[\text{full}]$ = P[buffer is full with no flows priority $< s$] and $P[X_j]$ = probability that a message is of flow class j.

6.2 Average Blocking Length in Stage 3

The header flit is blocked waiting at stage 3 if the arbiter has higher priority flows in the arbitration slots which reserve all crossbar ports that this flow might require or if all the crossbar ports are occupied. The blocking delay is :

$$delay2_s = P[\text{Arbiter Busy}].\{M/2\} \quad (6.9)$$

$$P[\text{arbiter busy}] = \prod_{i=0}^{k} \sum_{j=s}^{S-1} P[X_j] + \left(1 - \sum_{i=1}^{l} {}^{l}C_i.p^i.q^{l-i}\right) \quad (6.10)$$

where M is the message length in flits, k is the number of arbitration slots, l is the number of crossbar ports that could be assigned, p is the probability that a given output VC is full and $q = 1 - p$.

6.3 Average Blocking Length in Stage 5

Since we are using a flexible virtual channel allocation scheme a flit is blocked waiting at the last stage only if no virtual channels are empty in the output virtual channel buffer.

$$delay3_s = P[\text{Ouput Buffer full}].\{M/2\} \quad (6.11)$$

$$P[\text{no VCs}] = 1 - P[\text{at least 1 free channel}] \quad (6.12)$$

$$P[\text{at least 1 free channel}] = \sum_{i=1}^{S-1} {}^{(S-1)}C_i.p^i.q^{n-i} \quad (6.13)$$

$$P[\text{no VCs}] = 1 - \sum_{i=1}^{S-1} {}^{(S-1)}C_i.p^i.q^{n-i} \quad (6.14)$$

where M is the message length in flits and n is the number of virtual channels in the VC buffer.

6.4 Deadline Missing Probability

We can now use the average blocking lengths at stages 1, 3 and 5 of the router pipeline to calculate the probability of a packet of class s missing a deadline D_s. The average blocking length ($delay$) for a message in a router is

$$delay_s = delay1_s + delay2_s + delay3_s \quad (6.15)$$

Given a delay ($delay_s$) the actual time for transfer (β_s) of the message is

given by:

$$\beta_s = delay_s.S_s \tag{6.16}$$

where S_s is the Average No of Cycles to transmit a flit of class s.

Equation (6.15) gives the average delay for a message of class s through a *single* router. The probability that the message misses its deadline is given by:

$$P_{m,s}(D_s,\beta_s) = P\{\beta_s > D_s\} = 1 - P\{\beta_s \leq D_s\} = \sum_{i=0}^{i=B_l} P_s(i) \tag{6.17}$$

where $P_{m,s}$ is the probability of missing the deadline D_s for a class s message and β_s is the actual delay and $P\{\beta_s \leq D_s\}$ is the probability that a class s message traverses the router within the deadline D_s and B_l is the highest blocking length such that $\beta_s \leq D_s$.

If the message were to traverse r routers to its destination, the probability that it misses its deadline can be calculated as the sum of the probabilities of all combinations of delays at these routers such that the total delay is less than B_m, which the maximum total length such that the message reaches on or before its deadline.

$$P\{\beta_s > D_s\} = 1 - P\{\beta_s \leq D_s\} = \sum P_s(D_0).P_s(D_1)\cdots P_s(D_{r-1}) \tag{6.18}$$

where

$$\forall D_0, D_1, \cdots, D_{r-1} \in \mathbb{Z}^+ \tag{6.19}$$

$$D_0 + D_1 + \cdots + D_{r-1} \leq B_m \tag{6.20}$$

Equation (6.18) represents all possible combinations of delays at the routers . We need a solution to $P_s(B)$ to calculate the deadline missing probability.

Since it is tough to exactly calculate this parameter, we approximate this probability using operational behaviour of a router. Under the uniform distribution assumption this probability can be calculated as follows.

$$P_s(B) \approx \begin{cases} 1 - \sum_{i=1}^{B^u} P_{b,s}/B^u, & B = 0 \\ P_{b,s}/B^u, & 1 \leq B \leq B^u \\ 0, & \text{otherwise} \end{cases} \tag{6.21}$$

where, B^u represents the worst case blocking length at a router and $P_{b,s}$ is the blocking probability of a class s message. The probability of blocking of a class s message can be calculated as follows.

$$P_{b,s} = \prod_{i=0}^{S-1} \sum_{j=s}^{S-1} P[X_j] + (1 - \sum_{i=1}^{S-1} {}^{(S-1)}C_i.p^i.q^{n-i}) + \prod_{i=0}^{k} \sum_{j=s}^{S-1} P[X_j] + \left(1 - \sum_{i=1}^{l} {}^{l}C_i.p^i.q^{l-i}\right) \tag{6.22}$$

and worst case blocking length is given by

$$B^u = M + M + M \Rightarrow 3.M \tag{6.23}$$

Therefore we can rougly estimate the deadline missing probability of a class s message through a series of routers using Equations (6.18), (6.21), (6.22) and (6.23).

CHAPTER 7

SIMULATION PLATFORM

7.1 Interconnection Network Simulator

We have developed an interconnection network simulator in Java to compare the performance characteristics of the modified router with the QoS enabled architecture discussed in chapter 2 and a router with the modifications we have proposed. For our experiments we simulated a 8-port Router and a 2×2 Mesh Network with 8-port routers. We have used 16 VCs per physical channel. The flit size is 128-bits and all the messages are 36 flits long. Physical link bandwidth is 1.6 Gbps and the flit buffers are 36 flits deep. Figure 7.1 shows a schematic of the simulator environment.

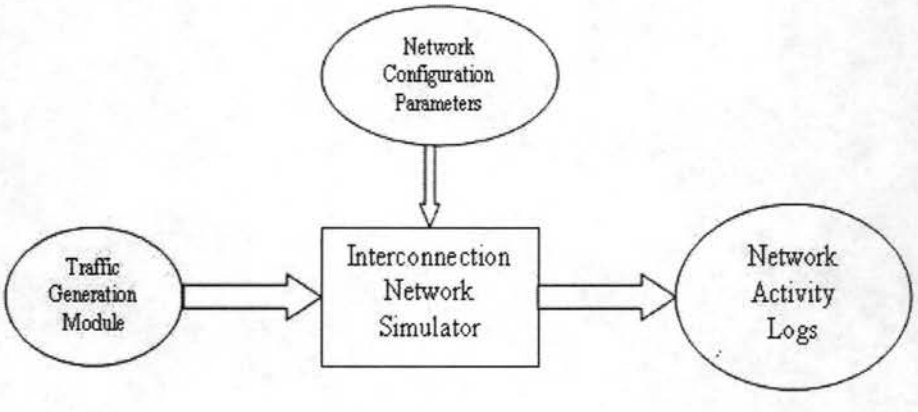

FIGURE 7.1: Interconnection Network Simulator

The main simulator components are as follows.

- *traffic generation module* generates traffic patterns with various packet mixes to test the performance characteristics of the router.
- *Network Configuration Parameters* specify the topology of the network, number of nodes and the type of connections (unidirectional or bi-directional).
- *Interconnection Network Simulator* uses the traffic used by the traffic generation module to simulate the network topology specified by the network configuration.
- *Network Activity logs* are generated, which measure the latency, deadline missing probability and average deadline missing time.

7.2 Workload

The workload includes messages from real time variable bit rate (VBR) traffic and best-effort traffic. The real time traffic streams are generated as synthetic MPEG streams at 30 frames/sec with different bandwidth requirements. Each stream generates a frame of data which is fragmented into flits. Best effort traffic is generated with a given injection rate λ and follows a poisson distribution. The message destination is assumed to be uniformly distributed.

An important parameter that is varied is the input load which is expressed as a fraction of the physical link bandwidth. For a specific input load, we vary the ratio of the two classes ($x : y$, where $x/(x + y)$ is the fraction of load for VBR traffic and $y/(x + y)$ is the fraction of load for best effort component) to generate mixed-mode traffic.

We vary the traffic ratio in 5 stages during the simulation to simulate dynamic workload. The important output parameters we measured were the *deadline missing probability* and *deadline missing time.* Deadline missing probability is the ratio between the number of frames which missed their deadline out of the total delivered frames. Deadline Missing time is the average time by which the packets miss their deadline. The deadline was set to 33.3 msec after receiving a frame from the flow. We have assumed a core clock frequency of 100 MHz and set the respective number of clock cycles for the deadline and the results are also expressed in term of clock cycles.

7.3 Performance Results

7.3.1 Comparison of the Two Router Models

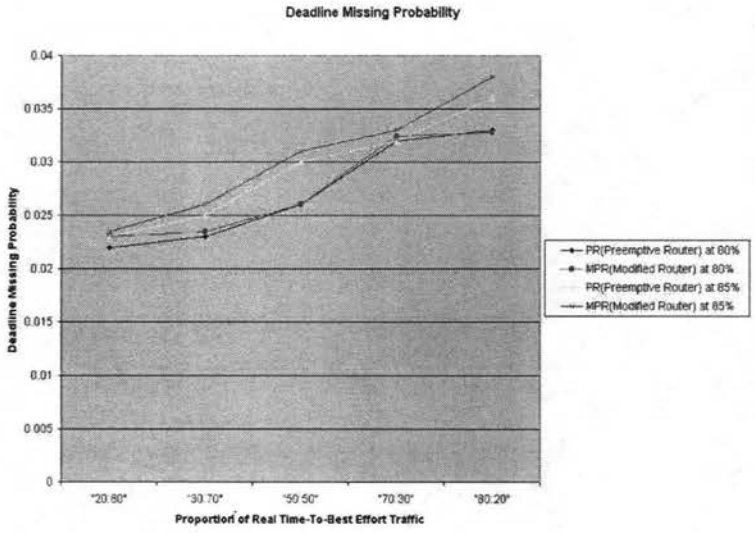

FIGURE 7.2: Deadline Missing Probability

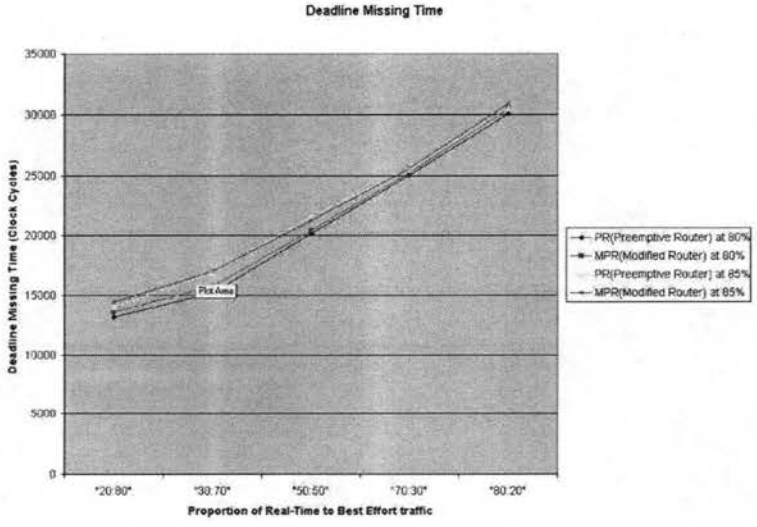

FIGURE 7.3: Deadline Missing Time

We used the simulation test bed to study the performance of the proposed router model with that of the existing router model. We ran the simulations at 80% and 85% network load. Figure 7.2 shows the deadline missing probability of the proposed router model to the existing preemptive router model and Figure 7.3 shows the deadline missing time between the two router models.

Figure 7.2 shows that at 80% load and at 85% network loading the deadline missing probabilities of the proposed router architecture and the existing router architecture are very close . It is also seen that the number of frames missing the deadline increases as the ratio of real time traffic increases. Figure 7.3 shows the deadline missing times of the two router models at 80% and 85% loads are very close. It is also seen that the average deadline missing time increases as the ratio of real time traffic increases.

7.3.2 A (2 × 2) Mesh Network Results

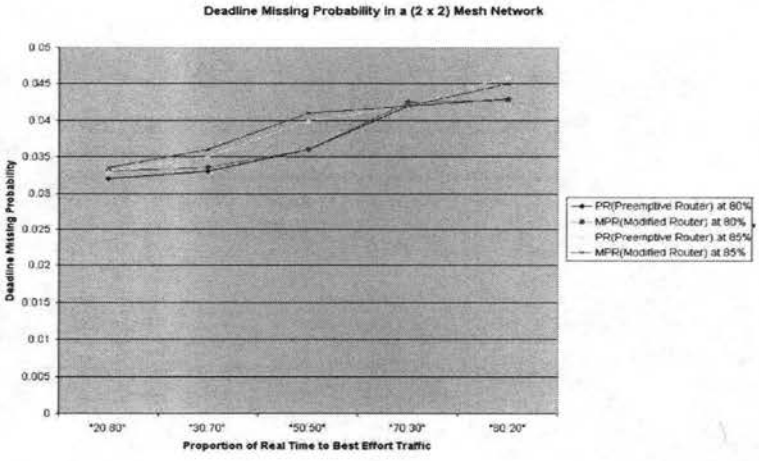

FIGURE 7.4: Deadline Missing Probability

Figure 7.4 shows that at 80% load and at 85% network loading the deadline missing probabilities of the proposed router architecture and the existing router architecture are very close. Like the single router results it is seen that the number of frames missing the deadline increases as the ratio of real time traffic increases.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This project addresses the issue of hardware complexity in QoS capable routers to enable faster switching. We studied the existing QoS capable router architecture to identify sources of hardware complexity and used techniques like Buffer Status Aware Link Scheduling and Dynamic Output VC allocation to reduce the hardware complexity without sacrificing performance. We used simulation to prove the performance of the proposed router architecture.We have also proposed an analytical model for analyzing QoS capable clusters.

We are now studying the causes for performance losses in the proposed router architecture to improve the model for higher performance. We are also studying the impact of adaptive routing algorithms on the QoS capabilities and hardware complexity of the routing fabric. We also propose to validate the analytical model using simulation.

BIBLIOGRAPHY

[1] Chita. R. Das, E. J. Kim, and K. H. Yum. *QoS provisioning in clusters: An investigation of Router and NIC design.* Proceedings of the 28th International Symposium on Computer Architecture, ISCA 01, Sweden, 2001

[2] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann Publishers, second edition, 2002.

[3] E. J. Kim, K. H. Yum, and C. R. Das, *An Analytical Model for a QoS Capable Cluster Interconnect* , in Proceedings of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB 2001), pp.9-24, September 2001, Germany

[4] E. J. Kim, K. H. Yum, and C. R. Das, *Calculation of Deadline Missing Probability in a QoS Capable Cluster Interconnect*,in Proceedings of IEEE International Symposium on Network Computing and Applications (NCA '01), pp.34-43, February 2002, Cambridge, MA.

[5] L.Zhang, *VirtualClock: A New Traffic Control Algorithm for Packet-Switched Networks.*, ACM Transactions on Computer Systems, 9(2):101-124, May 1991.

[6] W. J. Dally, *Performance Analysis of k-ary n-cube interconnection networks*, IEEE Transactions on Computers, vol. C-39, no. 6, pp. 775-785, June 1990.

[7] W. J. Dally, *Virtual-channel flow control*, IEEE Transactions on Parallel and distributed systems, vol. 3, no. 2, pp. 194-205, March 1992.

[8] W. J. Dally and C. L. Seitz, *Deadlock-free message routing in multiprocessor interconnection networks*, IEEE Transactions on Computers, vol. C-36, no. 5, pp. 547-553, May 1987.

[9] A. A. Chien, *A cost and speed model for k-ary n-cube wormhole rotuers*, Proceedings of Hot Interconnects93, August 1993.

[10] J. Duato, *A necesary and sufficient for deadlock-free adaptive routing in wormhole networks*, IEEE Transactions on Parallel and distributed systems, vol. 6, no. 10, pp. 1055-1067, October 1995.

Appendix

We developed a simulation testbed for analyzing the performance characteristics of the proposed router architecture because of the lack of simulators which simulate the functional aspects of the router accurately. In this chapter we will describe the design aspects of the simulator.

Simulator Design

The control module of the simulator is the *SimulatorCore* which controls the simulation and sets up interconnect and instantiates the *Interconnect* object. The simulator also keeps track of the number of clock cycles that have expired since the start of the simulation and all performance statistics. Figure 8.1 shows the high level organization of the simulator.

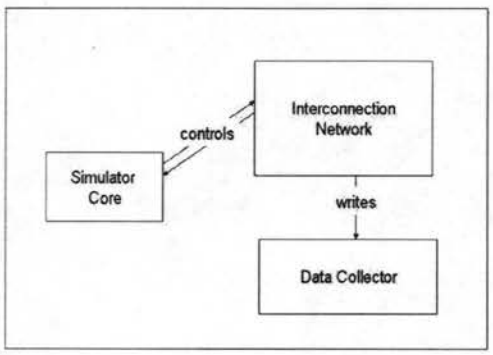

FIGURE 8.1: Organization of the simulator

The simulator is written in the JAVA programming language and is organized as a collection of interacting objects. All the classes derive from an abstract base class called *PipeStage*, which includes two methods :

- *compute* method, which represents the work done by a functional unit during a clock cycle.
- *clock* method, which represents the state updates at the pipe stage latches and buffers.

All functional units derive from this base class to specialize their respective functions. Figure 8.2 shows the class hierarchy of the simulator.

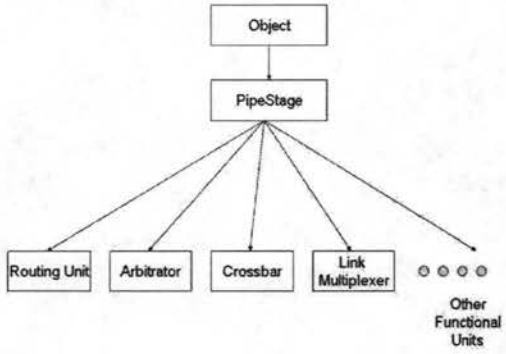

FIGURE 8.2: Class hierarchy

We have tested the simulator by comparing simulator output traces with some verified traces derived from specific input patterns.