An Analytical Model for a QoS Capable Cluster Interconnect*

Eun Jung Kim, Ki Hwan Yum, and Chita R. Das
Department of Computer Science and Engineering
The Pennsylvania State University, University Park, PA 16802 U.S.A.
{ejkim|yum|das}@cse.psu.edu

Abstract

The growing use of clusters in diverse applications, many of which have real-time constraints, requires Quality-of-Service (QoS) support from the underlying cluster interconnect. In this paper, we present an analytical model for a wormhole-switched router with QoS provisioning. In particular, the model captures message blocking due to wormhole switching and bandwidth sharing due to a rate-based scheduling mechanism. Average message latency for different traffic classes is computed using the model. We evaluate a 16-port router and hypercubes of different dimensions with a mixed workload of real-time and best-effort traffic. Comparison with the simulation results shows that the single router and the network models are quite accurate in providing the performance estimates.

1 Introduction

Quality-of-Service (QoS) provisioning in clusters is becoming a critical issue with the widespread use of clusters in diverse commercial applications. The traditional best-effort service model that has been used for scientific computing is not adequate to support many cluster applications with varying consumer expectations. As an example, many web servers and database servers make efficient use of clustering technology from cost, scalability, and availability standpoints. However, the tremendous surge in dynamic web contents, multimedia objects, e-commerce, and other web-enabled applications requires QoS guarantees in different connotations. This in turn mandates that the cluster system, and hence the cluster interconnect, should be able to handle user specified service demands instead of adopting the *same-service-to-all* model. Hence, design and analysis of QoS capable routers[1] and cluster networks has become a current research focus.

High performance cluster networks, also known as system area networks (SANs), usually use switch-based architectures. Most commercial routers (switches) such as Cray T3D/E,

*This research is supported in part by NSF grants MIPS-9634197, CCR-9900701, and equipment grants from NSF and IBM.

[1]QoS capable routers for Internet have been designed with various flavors. ATM networks can provide QoS guarantee, but incur high latency. We are specially interested in low-latency cluster networks based on cut-through switching.

Tandem Servernet-II, Intel Cavallino, IBM SP2, and Myricom Myrinet [17, 8, 3, 20, 2] use wormhole switching to provide high performance. In wormhole switching, a message is broken into flits (a few bytes each) for transmission and flow control. The header flit(containing router information) establishes the route and the remaining data and tail flits follow in a pipelined fashion. The tail flit restores the resources. If the header is blocked, the remaining flits are blocked in their respective positions. These routers have not been designed for QoS assurance except for the Servernet-II [8], which provides a link arbitration policy (called ALU-biasing) for implementing limited bandwidth and delay control.

Since wormhole switching has been adopted in most commercial routers, it would really be advantageous if we could make them QoS capable with minimal design changes. Some recent modifications to wormhole routers have been considered for handling traffic priority [14, 18, 22, 21]. The most logical solution is to assign separate virtual channels to different traffic classes and use a rate-based scheduling mechanism such as Fair Queueing [5] or VirtualClock [23] to share the link bandwidth proportionately [14, 22]. Techniques such as preemption of lower priority traffic in favor of higher priority traffic have also been proposed [18]. Recently, we have proposed a QoS-aware pipelined router that supports features such as rate-based scheduling, preemption, and flit accelation mechanism [21].

A limitation of all prior studies on routers/networks that support integrated traffic is that they use simulation to evaluate the performance of various design trade-offs. In addition, the evaluations are confined to a single router in many cases. For example, in [7, 22, 21] where the router designs are evaluated with multimedia video streams, the study is limited to a single (4-port/8-port) router or a small network. Detailed flit-level simulation is quite expensive and prohibits full-blown analyses of various design trade-offs. On the other hand, an accurate analytical model can provide quick performance estimates and will be a valuable design tool.

In this paper we present a mathematical model for analyzing QoS capable cluster networks. We use a bottom-up approach first by developing the model for a single router and then extending it to a network. Here we use a hypercube-style cluster network primarily to keep the analysis tractable due to the symmetric nature of the network. Such a topology has been used in the SGI Origin architecture [13]. However, our QoS-aware router model can be extended to any regular topology such as k-ary n-cubes and meshes as long as the topology and routing algorithm can be captured mathematically. In fact, it should be possible to integrate our router model with the prior network models [1, 12, 4, 6, 9] to analyze different QoS-aware cluster networks.

Like many commercial designs, we use a pipelined wormhole router architecture. The model considers an integrated workload consisting of C different classes of traffic. $(C-1)$ classes represent real-time applications[2] with distinct service requirements. The last class is used for best-effort traffic applications. As proposed in our MediaWorm design [22], each class is statically assigned at least one virtual channel, and the virtual channels are sched-

[2]Here real-time application refers to any time-constrained application.

uled with a rate-based scheduling algorithm, VirtualClock [23], to regulate the bandwidth requirements. Average message latency for different traffic classes can be computed using this model.

The main contribution of this analytic model is that it captures the chained blocking possible in pipelined wormhole-switched networks, and the bandwidth sharing mechanism of the VirtualClock algorithm in finding the average latency. Unlike the prior lumped delay models [1, 12, 4], here we analyze contention at different stages of the pipelined router. Moreover, the average behavior analysis of the VirtualClock algorithm is applicable to other work conserving techniques such as Fair Queueing and Weighted Round Robin. While prior analyses of these scheduling algorithms provide only performance bounds [15, 19, 16], we demonstrate in this paper that the average behavior of these schemes can be captured analytically.

We validate the single router model (16-port) and the cluster network model (up to 7-cubes) through extensive simulation. We use a mixed workload of three traffic classes ($C = 3$, two real-time and one best-effort) in this study. It is shown that the models are quite accurate in predicting the average message latency. Using the model, it is not only possible to predict the per class delay behavior, but also the impact of application mix. In addition, the model can quantify different components of message latency (queueing time, transfer time, blocking time), effect of buffer length, and other design trade-offs in an effective manner. Thus, it can be used as an efficient design tool to analyze network and application centric performance parameters.

The rest of the paper is organized as follows. In Section 2, the router architecture and the VirtualClock algorithm are discussed. In Section 3, we present the analytic models. The performance results are analyzed in Section 4, followed by the concluding remarks in Section 5.

2 A QoS-aware Router Architecture

Most routers now use a pipelined design to minimize the network cycle time. Accordingly, we use a pipelined, wormhole-switched router in this paper. Fig. 1 shows the pipelined router consisting of five stages. Stage 1 of the pipeline represents the functional units, which synchronize the incoming flits, demultiplex a flit so that it can go to the appropriate input virtual channel buffer to be subsequently decoded. If the flit is a header flit, routing decision and arbitration for the correct crossbar output is performed in the next two stages (stage 2 and stage 3). On the other hand, middle flits and the tail flit of a message directly move to stage 4. Flits get routed to the correct crossbar output port in stage 4. Finally, the last stage of the router performs buffering for flits flowing out of the crossbar, multiplexes the physical channel bandwidth amongst multiple virtual chnnels, and transmits one flit at a time to the neighboring router or to the network interface of the node attached to this router.

In this n-port router architecture, we provide one virtual channel for each of the C traffic classes (thus C input and output virtual channels). More virtual channels per class

should improve the performance. Note that the crossbar used in our router is called a *full crossbar* since it has $n \times C$ inputs and $n \times C$ outputs. The model can be modified for a multiplexed crossbar, where the virtual channel multiplexing will be done before the crossbar stage.

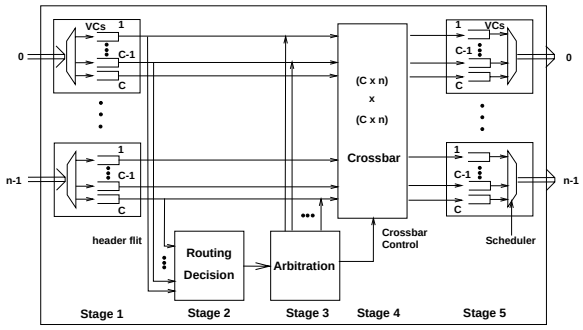


Figure 1: The pipelined router architecture with a full crossbar.

Unlike the lumped router models analyzed before [1, 12, 4, 9], a message entering the above pipelined router can experience delay at stages 1, 3 and 5 of the router. If the corresponding input buffer is full in stage 1, the message must wait outside the router until adequate space is available. In stage 3, the message again may be delayed because its destination crossbar output port could be busy. Crossbar output port arbitration is performed at a message level granularity. So the message has to wait until the output port is released by the message currently using it. Finally in stage 5, multiple virtual channels compete for the physical channel bandwidth. Traditionally, a Round Robin or FIFO scheduler is used to schedule the output channel in a time-division manner.

The above router design is modified for QoS provisioning by simply incorporating a rate-based scheduling algorithm to share the physical channel bandwidth. We use the VirtualClock algorithm [23] in this paper although all the three rate-based scheduling algorithms (VirtualClock, Fair Queueing and Weighted Round Robin) are shown to provide similar performance.

In the VirtualClock algorithm, there are two variables, called $auxVC$ and $Vtick$ for each connection. The values of these two variables are determined when a connection is set up. The $auxVC$ indicates the virtual clock value of the connection, while the $Vtick$ is the amount of time that should be incremented whenever a flit arrives at that connection. The $Vtick$ value specifies the interarrival time of flits from the connection. Therefore, a smaller $Vtick$ value implies higher bandwidth. Once these two values are set, the VirtualClock algorithm works as follows. For each connection i, when a flit arrives at the scheduler, the following computation is done.

$$auxVC_i \leftarrow \max(\text{real time}, auxVC_i),$$
$$auxVC_i \leftarrow auxVC_i + Vtick_i,$$
$$\text{timestamp the flits with the } auxVC_i.$$

The flits are queued and serviced in increasing timestamp order. For the best-effort traffic,

the timestamp is set as ∞. So the best-effort flits are processed only if there are no other flits with lower timestamp values.

3 An Analytical Model

In this section, we develop a mathematical model for a single router and then extend it to a hypercube network. The motivation for developing a single router model is two fold. First, the model can be used for evaluating small, single node clusters. Second, it can be extended to any other topology by capturing the impact of the network and routing algorithm. The hypercube topology is used as an example in this paper to demonstrate the applicability of the model.

As described in the previous section, the router model assumes a pipelined architecture with $P = 5$ stages. The model is derived for C classes of traffic consisting of $(C-1)$ classes real-time traffic and one class of best-effort traffic. Each class is assigned a dedicated virtual channel. (This assumption can be relaxed to assign multiple virtual channels to a class.) In addition, the model is based on the following assumptions typically used in analytical models: (i) The arrival pattern of each class c follows the Poisson processes with an average arrival rate of λ_c^g. (ii) Message length is M flits long. (iii) Message destination is uniformly distributed. (iv) The $Vtick_c$ value for real-time traffic belonging to class c is given by $1/(\lambda_c^g M)$, and the $Vtick$ value for best-effort traffic is set to ∞. (v) The input and output buffers(virtual channels) in stages 1 and 5 can hold b_s flits. Each class is assigned a dedicated injection/ejection queue outside the router, and these queues have infinite capacity.

The average message latency of class c $(1 \leq c \leq C)$ is composed of the average network latency, $\overline{L_c}$, which is the time to traverse the router (network), and the average waiting time, $\overline{W_c}$, at the injection channel. Thus,

$$\overline{Latency_c} = \overline{L_c} + \overline{W_c}. \tag{1}$$

3.1 Single Router Model

The average network latency($\overline{L_c}$) of a message of class c 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 of Fig. 1. The actual transmission time with P pipeline stages in a single router is $(P-1+M)$ cycles for an M-flit message.

In order to compute the second part of the network latency, let us define $\overline{B_c}$ as the average blocking length (in number of flits) seen by the header flit at the input, output, and arbitration stage in the router. $\overline{B_c}$ captures the message blocking in a pipelined wormhole router. Then the effective length of the message becomes $(M + \overline{B_c})$ flits. Let $\overline{S_c}$ be the average number of cycles required to transfer one flit of a class c message. $\overline{S_c}$ represents the effect of bandwidth sharing mechanism of the VirtualClock algorithm. Thus, the average

network latency($\overline{L_c}$) for $1 \leq c \leq C$ is

$$\overline{L_c} = P - 1 + (M + \overline{B_c})\overline{S_c}. \tag{2}$$

Since blocking can occur in stages 1, 3, and 5 of the router as discussed in Section 2, the average blocking length ($\overline{B_c}$) can be separated into three parts as

$$\begin{aligned} I &= P[\text{input buffer is not empty}] \cdot \{\max(b_s, M)/2\} \\ O &= P[\text{output buffer is not empty}] \cdot \{\max(b_s, M)/2\} \\ A &= P[\text{arbiter is busy}] \cdot M/2 \end{aligned}$$

where I, O, and A represent the corresponding blocking lengths. In each of the above expressions, 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 $\max(b_s, M)/2$ flits. In the steady state, since the input and output rates of the router are the same, it is intuitively clear that the above three probabilities are equal, and are denoted as $P_{b,c}$.

Since the input/output buffer sizes are b_s, the blocking probability, $P_{b,c}$ for class c $(1 \leq c \leq C)$ can be expressed as

$$P_{b,c} = (\overline{L_c}\lambda_c')^{1+2\frac{\max(b_s,M)}{M}} \tag{3}$$

where λ_c' is the steady state message arrival rate of class c traffic. $(\overline{L_c}\lambda_c')$ is the router utilization (ρ_c) for class c. Since λ_c' and $\overline{L_c}$ are considered at the message-level granularity, the total buffer size($2b_s$ flits) of input and output queues becomes $2b_s/M$ when converted to message length. Including the currently serviced flit(message), the total number of messages becomes $2b_s/M + 1$. Hence, the channel utilization (or blocking probability of class c) is given by Eq. 3. The $\max(b_s, M)$ term is used to capture the buffer length $b_s < M$ since a new message must wait until the service for the previous message is completed. The steady state arrival rate λ_c' in Eq. 3 is given by

$$\lambda_c' = (1 - P_{b,c})\lambda_c^g. \tag{4}$$

Combining the three blocking lengths (I, O and A), $\overline{B_c}$ becomes

$$\overline{B_c} = P_{b,c}(\max(b_s, M) + M/2). \tag{5}$$

The next unknown term in Eq. 2 is $\overline{S_c}$, for $1 \leq c \leq C$. Since the scheduler treats a real-time message and a best-effort message differently, we compute $\overline{S_c}$ separately for the two broad classes of traffic. First we compute $\overline{S_c}$, $1 \leq c \leq C - 1$, for all real-time traffic based on a Markov model, and then use a different technique to compute $\overline{S_C}$ for best-effort traffic.

Average number of cycles to transfer a flit of Real-time traffic ($\overline{S_c}$): When a real-time message of class c arrives at the output buffer, if other buffers are empty, it will take only one cycle to transfer a flit from that message. Otherwise, the output

channel bandwidth is shared among the virtual channels according to the corresponding *Vtick* values. For example, when two output buffers are occupied by class i and j messages whose *Vtick* values are $Vtick_i$ and $Vtick_j$ respectively, the number of cycles to transfer a flit of class i (S_i) is $(\frac{1}{Vtick_i} + \frac{1}{Vtick_j})/(\frac{1}{Vtick_i})$, and the number of cycles required for a flit of class j (S_j) is $(\frac{1}{Vtick_i} + \frac{1}{Vtick_j})/(\frac{1}{Vtick_j})$.

With $(C-1)$ classes of real-time traffic, for any tagged class i, there are 2^{C-2} combinations of other real-time traffic that denote whether they occupy the corresponding virtual channels or not. All these combinations will affect the output channel bandwidth sharing. To model this effect, we number the combinations serially so that for each combination k we can compute the effective cycle time. We can express the average number of cycles per flit for real-time traffic c as

$$\overline{S_c} = \sum_{k=0}^{2^{(C-2)}-1} S_c(k) P_c(k), \tag{6}$$

where $S_c(k)$ is the number of cycles required for class c in the kth combination, and $P_c(k)$ is the probability of kth combination for traffic c.

Let Z be the buffer state of the $(C-1)$ output virtual channels assigned to real-time traffic. Then it can be expressed with a bit string $Z = (d_1, d_2, \ldots, d_{C-1})$, where $d_i = 1$ if the virtual channel i is occupied, and $d_i = 0$ otherwise. Let us define $|Z|_j = d_j$, for $1 \leq j \leq (C-1)$, where the state $Z = (d_1, d_2, \ldots, d_{C-1})$. Then $|Z|_j$ indicates the state of the jth output virtual channel.

Definition 1 *Let Z be a given state where $|Z|_c = 1$. $Z = (d_1, d_2, \ldots, d_{c-1}, 1, d_{c+1}, \ldots, d_{C-1})$. The numbering function ν, for a given Z and c, returns the serial number for each output buffer combination as*

$$\nu(Z, c) = \sum_{j=1}^{C-2} d'_j 2^{j-1} \quad \textit{where} \quad d'_j = \begin{cases} d_j & 1 \leq j < c \\ d_{j+1} & c \leq j \leq C-2. \end{cases}$$

For a given real-time of class c that occupies virtual channel c, we can find all combinations of the other $(C-2)$ virtual channels from the above expression. Given $(C-1)$ types of real-time traffic, and a state $Z = (d_1, d_2, \ldots, d_{C-1})$, where $|Z|_c = 1$ and $\nu(Z, c) = k$,

$$S_c(k) = (\sum_{\forall j, d_j = 1} \frac{1}{Vtick_j})/(\frac{1}{Vtick_c}). \tag{7}$$

This generalization is obtained from the two class example discussed earlier. Next, the probability of kth combination for class c, $P_c(k)$, can be determined using a Markov model. Let Z_1 be a state such that the cth output buffer is empty ($|Z_1|_c = 0$) and Z_2 be the state such that the cth output buffer is not empty($|Z_2|_c = 1$). The status of the rest $(C-2)$ buffers are all identical in the two states to make Z_1 and Z_2 adjacent. Using the definition of ν, we can find $\nu(Z_2, c) = k$. Now, the transition rate from state Z_1 to Z_2 is λ'_c, where λ'_c is the traffic rate of the cth virtual channel (Eq. 4), while the rate from Z_2 to Z_1 is $(1/L_c(k) - \lambda'_c)$, where $L_c(k) = P - 1 + (\overline{B}_c + M)S_c(k)$ from Eq. 2. The transition rate from

Z_2 is reduced by λ'_c to account for the arrival of a message while channel c is busy. From the Markov model, we get all the state probabilities, Π_{Z_i}, $0 \leq i \leq 2^{C-1} - 1$. Then

$$P_c(k) = \frac{\Pi_{Z_u}}{\sum_{\forall j, |Z_j|_c=1} \Pi_{Z_j}}, \quad \text{where} \quad \nu(Z_u, c) = k. \tag{8}$$

Average number of cycles to transfer a flit of Best-Effort traffic ($\overline{S_C}$): Note that since the *Vtick* value for the best-effort traffic is infinite, the best-effort message only uses the empty cycles when there is no real-time traffic. Moreover, transfer of best-effort flits can be interrupted if a real-time message arrives at the output buffer, and should be resumed after the real-time traffic transfer is complete. This can be modeled as a *preemptive resume priority queue* [10].

We model this phenomenon by computing the overall time to transfer a best-effort message of M flits long. This time, denoted as S_m, consists of three parts. The first part is the actual service time, which is M cycles. The second part is the average waiting time of the best-effort message or the residual service time of all real-time messages already in the output buffers. Note that the situation here is slightly different from the original *preemptive resume priority queue discipline* model in that, in our case we have already included the waiting time in the input and output buffers when we calculated the blocking length in Eq. 5. Hence, we only need the waiting time when the best-effort message is at the head of output buffer. This is written as $\frac{R_r}{1-(\rho_1+\rho_2 \ldots+\rho_{C-1})}$, where R_r is the residual time of all real-time messages in the output buffers, and is given by $\sum_{c=1}^{C-1} \frac{\lambda'_c M^2}{2}$ [10]. The last part of the delay is due to preemption of the best-effort traffic to yield to any of the $(C-1)$ classes of real-time traffic. This inflates the overall transfer time by the channel utilization of $(C-1)$ classes. S_m now becomes

$$S_m = M + \frac{R_r}{1 - (\rho_1 + \rho_2 + \cdots + \rho_{C-1})} + \sum_{i=1}^{C-1} \rho_i S_m.$$

The average number of cycles to transfer a best-effort flit after simplification becomes

$$\overline{S_C} = S_m/M = \frac{(2-\rho_r)}{2(1-\rho_r)^2}. \tag{9}$$

With Π_0 as the probability that there is no real-time traffic, we can find $\rho_r = 1 - \Pi_0$, where $\rho_r = \sum_{i=1}^{C-1} \rho_i$. We can get Π_0 from the previous Markov model, where state 0 is $(d_1, d_2, ..., d_{C-1}), \forall j, d_j = 0$. All the terms in Eq. 2 are now quantified to compute $\overline{L_c}$. Note that due to the inter-dependencies between $P_{b,c}$ and λ'_c, the solution becomes iterative.

3.2 Modeling of a Cluster Interconnect

The single router model can be extended to most of the regular networks as long as the topology and routing algorithm can be captured analytically. Here, we consider integrated traffic in the network, and use a hypercube topology to demonstrate this idea. We use the deadlock-free e-cube routing algorithm for message transfer.

The wormhole-routed hypercube (n-cube) model proposed in [12] is combined with our pipelined router model to compute the average message latency. The motivation for using the model proposed in [12] is that it is not only quite accurate over the entire workload, but also computes the message latency per link, which is required for accurate performance estimates per connection.

In an n-cube network, each node has n input and n output links in addition to an injection and an ejection channel for the local host. Messages generated by a node could travel h-hops, where $1 \leq h \leq n$. Thus, each physical channel is likely to experience a different load, and therefore, the single router model of the previous section should be modified to express traffic analysis for each link s, where $0 \leq s \leq n-1$.

3.2.1 Average Network Latency ($\overline{L_c}$)

The actual transmission time($\overline{T}$) with a P-stage router in an n-cube is $(P - 1 + P\overline{h} + M)$ cycles, where $\overline{h}$ is the average number of hops a message travels in the hypercube. The average number of hops is given by $\overline{h} = \sum_{k=1}^{n} kP_k$, where $P_k = {}_nC_k/(N-1)$, and ${}_nC_k = \begin{pmatrix} n \\ k \end{pmatrix}$. N $(= 2^n)$ is the number of nodes in an n-cube. The average network latency $\overline{L_c}$ of Eq. 2 needs to be modified to consider the latency for each class c when it starts with a specific physical channel. Let $L_{c,s}$ be the latency of a class c message when it uses a physical channel s as the first path to traverse towards its destination. Then $h_s = \sum_{k=0}^{n-s-1}(k+1) \cdot {}_{n-s-1}C_k/(2^{n-s-1})$ denotes the average number of hops a message travels starting with the physical channel s as the first path.

The network contention in the hypercube network is divided into three separate parts — blocking at the input stage of the first router, blocking at the ejection channel of the last router, and blocking in the middle routers. Let $I_{c,s}$ be the blocking length of a class c real-time message at stage 1 of the first router that uses channel s as the first route, and let $O_{c,n}$ be the blocking length at stages 3 and 5 in the ejection channel of the last router. Also, let $B_{\text{middle}}(c,s)$ be the blocking length between the source and the destination (*i. e.* middle nodes) excluding the blocking length at stage 1 of the source and the blocking length at stages 3 and 5 of the destination.

To compute the average number of cycles required to transfer a flit due to sharing of real-time messages at the output virtual channels, we need to consider two separate cases again — sharing at the ejection channel and in the rest of the channels. Consequently, let $\overline{S_{c,s}}$ be the average number of cycles required to transfer a flit of class c message that uses channel s for its first path, and $\overline{S_{c,n}}$ be the average number of cycles per flit in the ejection channel. With these definitions, $L_{c,s}$ can be expressed as

$$L_{c,s} = \{P - 1 + Ph_s\} + \{(O_{c,n} + M) \cdot \overline{S_{c,n}}\} + \{(I_{c,s} + B_{\text{middle}}(c,s)) \cdot \overline{S_{c,s}}\}. \qquad (10)$$

The first term in Eq. 10 indicates the number of cycles the header will take without contention. The second term represents the transfer time at the ejection channel(n). The total message length that includes the blocking length ($O_{c,n}$) at the ejection channel and the message length (M) is multiplied by the average number of cycles required to transfer

a flit ($\overline{S_{c,n}}$) at the ejection channel. Similarly, the total message length at the input buffer ($I_{c,s}$) and in the middle nodes is multiplied by the inflated cycle time ($\overline{S_{c,s}}$) to find the last term of Eq. 10. The average network latency ($\overline{L_c}$) becomes

$$\overline{L_c} = \sum_{s=0}^{n-1} L_{c,s} \cdot \frac{\lambda'_{c,s}}{\lambda'_c},$$

where λ'_c is the steady state message generation rate of class c while $\lambda'_{c,s}$ is the steady state message generation rate of class c in channel s. We need three types of traffic rates to complete the delay analysis. The first one is $\lambda^g_{c,s}$ which represents the message generation rate of class c for channel s, and the second is $\lambda'_{c,s}$, and the third is $\lambda_{c,s}$, the total steady state rate (including transit message). We have used the traffic analysis equations given in [12]. We defer the traffic rate equations to [11] due to space limitation.

Now from Eq. 3, the probability of blocking for class c traffic in channel s can be written as

$$P^s_{b,c} = (L_{c,s}\lambda_{c,s})^{1+2\frac{\max(b_s,M)}{M}}. \tag{11}$$

Note that $\lambda_{c,s}$ is the total rate here. Similarly Eq. 4 is modified as

$$\lambda'_{c,s} = (1 - P^s_{b,c})\lambda^g_{c,s}. \tag{12}$$

The $I_{c,s}$ and $O_{c,n}$ terms in Eq. 10 are similar to the I term and $\overline{B_c}$ in the single router and are given by $\overline{I_{c,s}} = (P^s_{b,c} \cdot \max(b_s, M)/2)$, and $\overline{O_{c,n}} = (P^n_{b,c} \cdot (\max(b_s, M)/2 + M/2))$. Finally the average number of cycles for transferring a flit at the ejection channel ($\overline{S_{c,n}}$) and at the other output buffers($\overline{S_{c,s}}$) can be obtained from equations 6, 7, and 8 after modifying the terms for representing the starting channel s for class c traffic. The only unknown term in Eq. 10 is $B_{\text{middle}}(c, s)$, and its detailed derivation can be found in [11].

Average number of cycles for transferring a flit of Best Effort traffic($\overline{S_{C,s}}$): Similar to the single router model, now we need to compute the inflated number of cycles to transfer best-effort traffic. However, we cannot use the *preemptive resume priority queue model* alone here due to the fact that the idle period between best-effort flits becomes completely random while traversing through the network. Therefore, we use the *Busy and Idle period* concepts from the M/G/1 queue to find $\overline{S_{C,s}}$ and $\overline{S_{C,n}}$.

Let *Busy* be the average length of busy period for all real-time traffic and *Idle* be the average length of idle period for real-time traffic. Given the total real-time traffic rate $\lambda^g_r = \sum_{i=1}^{C-1} \lambda^g_i$, the *Idle* and *Busy* periods become $Idle = 1/\lambda^g_r$ and $Busy = Idle(1-\Pi_0)/\Pi_0$, where Π_0 is the probability that the server is idle ($Idle/(Busy + Idle) = \Pi_0$).

A best-effort message could arrive during the idle period or busy period of real-time traffic [3]. If the message arrives during the idle period, then the message completion time is M cycles. If it arrives during the busy period, the completion time is $(M + Busy)$. In addition, a real-time traffic could arrive during the transmission of a best-effort traffic. This increases the message completion time by $(M \cdot \rho_r)$ or $(M + Busy) \cdot \rho_r$ corresponding

[3]Actually there are three types of arrival epochs. Due to space limitation and simplicity, we handle only two cases here.

to the above two cases, where ρ_r is the channel utilization due to all real-time traffic as explained in Eq. 9. Hence, the total transfer time is given by

$$S_m(s) = (M + M \cdot \rho_r)\Pi_0 + (M + Busy + (M + Busy) \cdot \rho_r)(1 - \Pi_0). \quad (13)$$

Then $\overline{S_{C,s}}$ can be obtained by dividing $S_m(s)$ by the message length M.

To compute $\overline{S_{C,n}}$ for best-effort traffic in Eq. 10, we need to capture the delay between successive flits of a best-effort message. Unlike most real-time transmissions, after the header flit of a best-effort message arrives at the destination, the remaining flits will arrive in random intervals due to delay at different hops of the network. Let the effective length of a best-effort message (M_s flits) be the difference between the arrival time of the header flit and that of the tail flit when a message uses channel s as the first path. Each one of the $(M-1)$ flits will need $\overline{S_{C,s}}$ cycles to transfer. However, the overall time can be reduced due to the blocking of the header flit in the middle routers, which is given by $B_{\text{middle}}(c, s)$. Thus, the effective length of a best-effort message becomes $M_s = \max(M, (M-1)\overline{S_{C,s}} + 1 - B_{\text{middle}}(C, s))$. The extra one cycle in this expression represents the header flit transfer time at the ejection channel.

Now using this effective message length, we compute the transfer time at the ejection channel by considering the two cases used in deriving Eq. 13. While the best-effort message yields to real-time traffic during a busy period, flits of the best-effort message gets accumulated in the buffer. So the effective length of best-effort message decreases. To quantify this phenomenon, the $(M + Busy)$ term in Eq. 13 is replaced by $(M_s/2 + Busy + M/2)$. Thus, the ejection channel transfer time is

$$S_m(n) = (M_s + M_s \cdot \rho_r)\Pi_0 + (M_s/2 + Busy + M/2 + (M_s/2 + Busy + M/2) \cdot \rho_r)(1 - \Pi_0).$$

Then, $\overline{S_{C,n}}$ can be obtained by dividing $S_m(n)$ by the message length M.

3.3 Average Waiting time at the Source Node

Finally, we need the average waiting time at the source node ($\overline{W_c}$) to find the average message latency in Eq. 1. Since the average waiting time accounts for the time consumed outside the router, it can be seen as the delay in the Network Interface (NI). Typically the delay is due to two data transfers involved in the NI. One is from the host to the NI and the other is from the NI to the network. In this model we only consider the transfer time from the NI to the network. To facilitate real-time scheduling in the NI, we assume that the NI buffer is divided into C classes and the VirtualClock algorithm is implemented among the C buffers [21].

The average waiting time consists of two parts. The first part is the time spent in the injection channel before arriving at the head of the respective queue. Since each class has a dedicated injection buffer, messages are transfered in FIFO manner within each class, Thus, the waiting time can be obtained by the queueing time of an $M/G/1$ queue as $\frac{\lambda_c^g \overline{L_c}^2 (1+\delta^2/\overline{L_c}^2)}{2(1-\lambda_c^g \overline{L_c})}$ with an arrival rate λ_c^g, mean service time $\overline{L_c}$, and variance $\delta^2 \approx (\overline{L_c} - \overline{T})^2$. The second part is the delay due to the VirtualClock algorithm, and is given by the average

number of cycles required to transfer the header flit of a message to the network. This is simply $\overline{S_c}$ in Eq.6 for the single router model. For modeling of a cluster interconnect, we can get $\overline{S_c}$ by

$$\overline{S_c} = \sum_{s=0}^{n-1} S_{c,s} \cdot \frac{\lambda'_{c,s}}{\lambda'_c} \tag{14}$$

where $S_{c,s}$ is the average number of cycles required to transfer a flit of class c message which uses physical channel s for its first path, $\lambda'_{c,s}$ and λ'_c have been defined before.

Summation of two parts yields the average waiting time as

$$\overline{W_c} = \frac{\lambda_c^g \overline{L_c}^2 (1 + \delta^2 / \overline{L_c}^2)}{2(1 - \lambda_c^g \overline{L_c})} + \overline{S_c}. \tag{15}$$

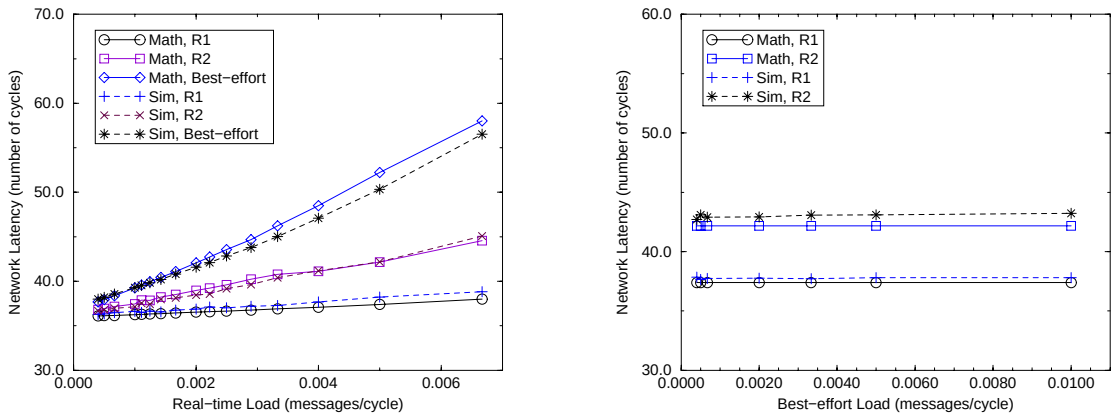
4 Performance Results

In this section, we analyze the performance results for a 16-port router and n-cubes of various sizes. The performance parameter is average message/network latency in cycles. (Although we have extended the model to compute deadline missing probability for real-time traffic, we don't include it due to space limitations.) To validate the analytical models, we have developed a flit-level simulator using CSIM. The switch size is 16×16 for a singel router and $(n+1) \times (n+1)$ for an n-cube. The Message size is 32 flits and so also is the Input/Output buffer size.

The results are reported for a mixed workload of two real-time (R1, R2) and one best-effort(BE) traffic types. For a given real-time load in messages/cycle, we generate two types of real-time traffic such that the intergeneration time of the second type is twice that of the first type. For example, if the load is 0.01, then the intergeneration time for the first real-time traffic (R1) is 100 cycles, and for the second real-time traffic (R2) is 200 cycles. If there are 3 types of real-time traffic, the intergeneration time of R2 is given by $1.5 \times$ (intergeneration time of R1) and that of R3 is given by $2 \times$(intergeneration time of R1). So in the following figures, real-time load implies the message generation rate of R1 only. The actual link load should include R1, R2, and BE. Note that these intergeneration times are used to represent the *Vtick* values ($Vtick_c = 1/(\lambda_c^g M)$. Best-effort traffic load is generated independent of the real-time traffic. After determining the intergeneration time for each class, messages are generated using exponential distribution. In the following subsections, we discuss only a selected set of results due to space limitation. The omitted results can be found in [11].

4.1 Single Router Results

We first validate the analytical model with simulation results for the 16-port router. Fig. 2 (a) shows the variation of average network latency($\overline{L_c}$) for the three classes of traffic. The analytical and simulation result differ by at most 5%. The graphs exhibit the QoS ability of the router in that both classes of real-time traffic incur smaller latency compare to the best-effort traffic, and also the R1 traffic with a smaller *Vtick* value has better performance



(a) (b)

Figure 2: Network latency comparison of analytical model and simulation model in a 16-port router with (a) varying real-time load and fixed best-effort load (0.01 msgs/cycle), and (b) varying best-effort load and fixed real-time load (R1:0.002 msgs/cycle).

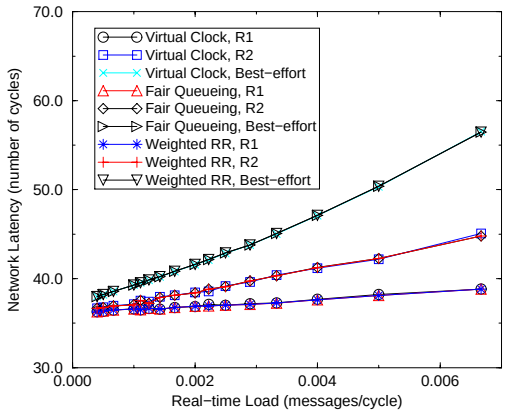


Figure 3: Comparison of VirtualClock, Fair Queueing, and Weighted Round Robin in the 16-port router with varying real-time load and fixed best-effort load (0.01 msgs/cycle).

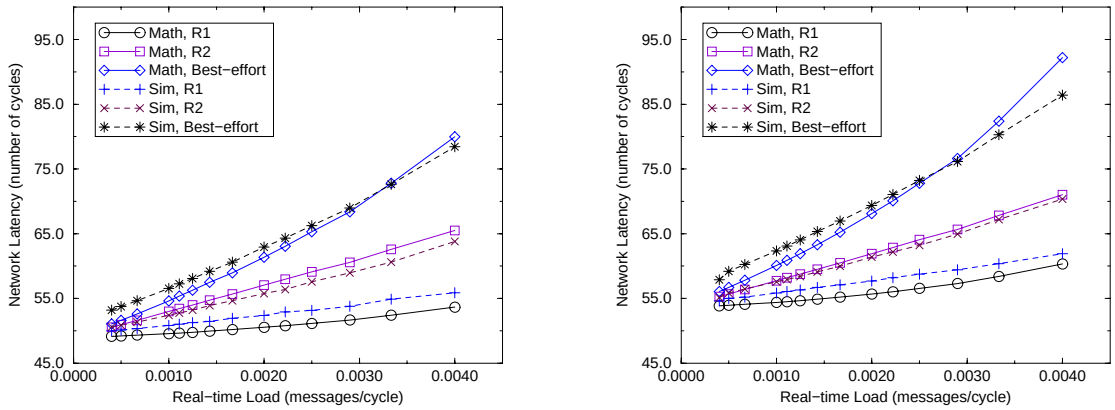
than the R2 class with a higher *Vtick* value. (Although not shown in the figure, it was observed that by replacing the VirtualClock algorithm with the Round Robin scheduling, the performance differences were lost. Rather R1 latency was higher than R2 latency since its input load was higher.)

Fig. 2 (b) also indicates the effect of VirtualClock scheduler. Since the best-effort messages are serviced only when there are no real-time messages, and a separate virtual channel is provided for each type of traffic, the best-effort load variation does not affect the real-time traffic latencies of classes R1 and R2.

Fig. 3 shows the simulation results for Fair Queueing, VirtualClock and Weighted Round Robin scheduling algorithms in order to reconfirm that these three algorithms have the same performance [16]. The average network latency curves for each traffic type (R1, R2, BE) match over the entire workload. We also get similar performance from our analyt-

ical model for the VirtualClock algorithm, as depicted in Fig. 2 (a). Thus, we can restrict our discussion to only the VirtualClock algorithm, although the results are applicable to the other two scheduling schemes. Moreover, unlike the bounding analysis reported in [15, 19, 16], here we can predict the average behavior of the work conserving scheduling mechanisms.

4.2 n-cube Results



(a) 5-cube (b) 7-cube

Figure 4: Network latency comparison of analytical and simulation models in (a) a 5-cube and (b) a 7-cube with varying real-time load and fixed best-effort load (0.002 msgs/cycle).

Fig. 4 (a) and 4 (b) depict the average network latency($\overline{L_c}$) results form the analytical and simulation models in a 5-cube and a 7-cube, respectively. The figures reveal that the analytic results closely match with the simulation results. For the 7-cube, the error in the best-effort results is relatively large for higher workload. This is due to the fact that with the same total load, the best-effort traffic enters the saturation region faster. With more classes of real-time traffic, the best-traffic latency enters into saturation region even faster compare to the results of Fig. 4. The graphs indicate that the rate-based scheduler favors higher priority traffic and thus, lower priority traffic suffers. Using the model as a design tool, we examined the effect of input/output buffer size. (Results are omitted due to space limit.) The results concur with prior studies in that the network latency is marginally affected by the buffer size.

5 Concluding Remarks

Provisioning for QoS in cluster networks is becoming a pressing issue with the increasing use of clusters in many commercial applications that need more sophisticated service than the traditional best-effort service model. While a few design alternatives have been proposed to support QoS in clusters, to our knowledge, there is no efficient mathematical

technique to evaluate the design trade-offs. The simulation or limited implementation approach used in prior studies is expensive and inflexible in providing fast-hand estimates to the wealth of questions that arise in making QoS design decisions. This paper introduces an analytic approach for evaluating a QoS-aware wormhole router and a hypercube-style cluster network, designed using such routers. The model captures the pipelined design, and analyzes the blocking delay at different stages of the pipe. In addition, the effect of VirtualClock scheduling algorithm is reflected in the model. Comparison with the simulation results indicates that the router as well as the hypercube models are quite accurate in predicting average message latency.

The present model can be improved in a variety of ways. First, the exponential arrival distribution for real-time traffic may not be quite practical to apply to media streams. We need to develop the model with a CBR/VBR source to capture inputs like media streams. Second, QoS comes with different connotations, and extension of the model to predict other performance parameters such as bandwidth assurance and jitter should be useful. Third, the model can be extended to other topologies. Finally, co-evaluation of the cluster network with a detailed network interface model should answer many questions regarding the QoS ability of the entire communication system.

References

[1] V. S. Adve and M. K. Vernon. Performance Analysis of Mesh Interconnection Networks with Deterministic Routing. *IEEE Trans. on Parallel and Distributed Sys.*, 5(3):225–246, March 1994.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[3] J. Carbonaro and F. Verhoorn. Cavallino: The Teraflops Router and NIC. In *Proc. of Hot Interconnects*, pages 157–160, August 1996.

[4] W. J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Trans. on Comp.*, 39(6):775–785, June 1990.

[5] A. Demars and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proc. of the ACM SIGCOMM*, pages 1–12, 1989.

[6] J. T. Draper and J. Ghosh. A Comprehensive Analysis Model for Wormhole Routing in Multicomputer Systems. *Journal of Parallel and Distributed Computing*, 32:202–214, 1994.

[7] J. Duato, S. Yalamanchili, M. B. Caminero, D. Love, and F. J. Quiles. MMR: A High-Performance Multimedia Router-Architecture and Design-Tradeoffs. In *Proc. of Intl. Symp. on High-Perf. Comp. Arch.*, pages 300–309, January 1999.

[8] D. Garcia and W. Watson. Servernet II. In *Proc. of 1997 Parallel Computing, Routing, and Communication Workshop (PCRCW'97)*, June 1997.

[9] P. T. Gaughan and S. Yalamanchili. A Performance Model of Pipelined k-ary n-cubes. *IEEE Trans. on Comp.*, 44(8):1059–1063, Auguest 1995.

[10] N. K. Jaiswal. *Priority Queues.* Academic Press, 1968.

[11] E. J. Kim, K. H. Yum, and C. R. Das. Performance Analysis of a QoS Capable Cluster Interconnect. Technical Report CSE-01-001, Pennsylvania State Univ., University Park, PA, January 2001.

[12] J. Kim and C. R. Das. Hypercube Communication Delay with Wormhole Routing. *IEEE Trans. on Comp.*, 43(7):806–814, July 1994.

[13] J. Laudon and D. Lenoski. The SGI Origin 2000: A CC-NUMA Highly Scalable Server. In *Proc. of Intl. Symp. on Comp. Arch.*, pages 241–251, June 1997.

[14] J.-P. Li and M. Mutka. Priority Based Real-Time Communication for Large Scale Wormhole Networks. In *Proc. of Intl. Parallel Processing Symp.*, pages 433–438, May 1994.

[15] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Trans. on Networking*, 1(4):344–357, June 1993.

[16] N. Pekergin. Stochastic Bounds on Delays of Fair Queueing Algorithms. In *Proc. of INFOCOM*, pages 1212–1219, March 1999.

[17] S. L. Scott and G. M. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Proc. of Hot Interconnects*, pages 147–156, August 1996.

[18] H. Song, B. Kwon, and H. Yoon. Throttle and Preempt: A New Flow Control for Real-Time Communications in Wormhole Networks. In *Proc. of Intl. Conf. on Paralle Processing*, pages 198–202, August 1997.

[19] D. Stiliadis and A. Varma. Rate-Proportional Servers: A Design Methodology for Fair Queueing Algorithms. *IEEE/ACM Trans. on Networking*, 6(2):164–174, April 1998.

[20] C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker. The SP2 High-Performance Switch. *IBM Sys. Journal*, 34(2):185–204, 1995.

[21] K. H. Yum, E. J. Kim, and C. R. Das. QoS Provisioning in Clusters: An Investigation of Router and NIC Design. To be presented at the Intl. Symp. on Comp. Arch. (ISCA 2001), June 2001.

[22] K. H. Yum, A. S. Vaidya, C. R. Das, and A. Sivasubramaniam. Investigating QoS Support for Traffic Mixes with the MediaWorm Router. In *Proc. of Intl. Symp. on High-Perf. Comp. Arch.*, pages 97–106, January 2000.

[23] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet-Switched Networks. *ACM Trans. on Comp. Sys.*, 9(2):101–124, May 1991.