Performance Analysis of Mesh Interconnection Networks with Deterministic Routing

Vikram Adve
Mary K. Vernon

Technical Report #1001b

July 1993

Performance Analysis of Mesh Interconnection Networks with Deterministic Routing

Vikram S. Adve
Mary K. Vernon

Computer Sciences Technical Report #1001b †
July 1993

† Revised version of Computer Sciences Technical Reports #1001 (February 1991) and #1001a (June 1992). This version is to appear in *IEEE Transactions on Parallel and Distributed Systems.*

Performance Analysis of Mesh Interconnection Networks with Deterministic Routing

Vikram S. Adve
Mary K. Vernon

University of Wisconsin-Madison
Computer Sciences Department
1210 West Dayton Street
Madison WI 53706.

{adve,vernon}@cs.wisc.edu

ABSTRACT

This paper develops detailed analytical performance models for k-ary n-cube networks with single-flit or infinite buffers, wormhole routing, and the non-adaptive deadlock-free routing scheme proposed by Dally and Seitz. In contrast to previous performance studies of such networks, the system is modeled as a *closed* queueing network that (1) includes the effects of blocking and pipelining of messages in the network, (2) allows for arbitrary source-destination probability distributions, and (3) explicitly models the virtual channels used in the deadlock-free routing algorithm.

The models are used to examine several performance issues for 2-dimensional networks with shared-memory traffic. Some results obtained are: (1) when processors are allowed to have multiple outstanding requests, system performance is bandwidth-limited and hence network performance does not *scale* well with increasing system size, (2) communication locality improves system efficiency, but a very high level of locality is needed for system performance to scale well, (3) in contrast to previous hot-spot studies for indirect networks that assume non-blocking processors, this study finds that significant tree-saturation *does not occur* even in the presence of severe hot-spots in systems with up to four outstanding requests per processor, and (4) at some plausible system operating points there is a perceptible difference in the efficiencies of processors at different locations in the mesh due to asymmetric loads on the virtual channels by the deadlock avoidance algorithm. These results should prove useful for engineering high-performance systems based on low-dimensional k-ary n-cube networks.

Index terms – Approximate Mean Value Analysis, closed queueing networks, finite buffers, hot-spots, multiprocessor interconnection networks, k-ary n-cube networks, mesh networks, near-neighbor communication, performance analysis, wormhole routing.

This research was supported by the National Science Foundation under grant number DCR-8451405, and by an IBM Graduate Fellowship.

1. Introduction

Multiprocessor mesh interconnection networks are 2-dimensional networks, with the processors arranged at the nodes of a grid, and point-to-point links connecting each node to its neighbors. Mesh interconnection networks are a special case of k-ary n-cube networks in which the number of dimensions, n, is two. Recent studies of k-ary n-cubes with *wormhole routing* (a low-latency pipelined routing scheme [9]) have shown that under reasonable assumptions, the optimal value for n is two or three [2, 8, 10]. Many existing and emerging multiprocessor systems use such low-dimensional direct networks to interconnect the processors, including the Intel Paragon, Cray T3D, Stanford Dash [14], M.I.T. Alewife [1], M.I.T. J-Machine [16] and CMU-Intel iWarp [5].

In this paper, we develop performance models to study k-ary n-cube networks with wormhole routing, with either single-flit or infinite network buffers. Our model for the single-flit buffer case includes the deadlock free routing algorithm of Dally and Seitz [9]. In contrast to previous analyses of these networks [2, 10, 11], the models we derive are *closed* queueing network models. Also in contrast to previous work, (i) we include the effects of blocking and pipelining of messages in the network, (ii) we allow for arbitrary source-destination probability distributions, and (iii) we explicitly model the virtual channels used in the deadlock avoidance algorithm. In the single-flit buffer model, the representation of message pipelining and blocking and the asymmetric virtual channel loadings of the deadlock avoidance algorithm require an approximate Mean Value Analysis solution that is rather complex. These features, however, have a significant impact on system performance and are thus important to model. The model provides a further example that approximate Mean Value Analysis can be used for accurate performance prediction of highly complex systems with non-product-form queueing behavior.

We use the models to examine several performance issues for two-dimensional networks. We study network performance and scalability with processors that must block after each request, as well as with processors that can make multiple requests before blocking for responses. We compare the performance of three mesh network topologies: the unidirectional and bidirectional tori (meshes with end-around links connecting corresponding nodes on opposite edges) and the bidirectional mesh without end-around links. We first study the above issues under a uniform traffic pattern. We then examine the impact of communication locality on network performance and scalability, and discuss how the other conclusions obtained under uniform communication change in the presence of varying degrees of locality. We also study network performance when a communication hot-spot occurs, including the effect of a hot-spot on other traffic in the network. Finally, we analyze and explain a potentially important performance implication of the deadlock avoidance algorithm. Specifically, this algorithm produces asymmetric loads on the virtual channels sharing each physical network link. The performance analysis shows that this asymmetry can lead to a perceptible difference between the efficiencies of processors at different locations in the mesh.

The remainder of this paper is organized as follows. Section 2 describes the mesh network and key performance issues in more detail, and states the assumptions about the system workload. Section 3 presents an overview of the models, and gives the details for the new techniques developed. The models also use several previously developed Mean Value Analysis approximations; the complete set of equations is given in Appendix B. Section 4 first presents the results of the model validations we performed using simulation, and then presents the performance analysis using the analytical models. Section 5 contains the conclusions of our study.

2. System Description

We describe the system and workload assumptions made in this study in Sections 2.1 through 2.5. In Section 2.6, we discuss several performance issues related to mesh networks that will be studied using the model.

2.1. Mesh Network Topologies

The basic topology of multiprocessor mesh interconnection networks is illustrated in Figure 2.1. There are a number of variations on this basic topology. The connection between each pair of adjacent nodes may be *unidirectional* or *bidirectional*, with the latter usually being implemented as two unidirectional links. With the unidirectional topology, *end-around* connections that connect a node at one edge to the corresponding node at the opposite edge (as shown in Figure 2.1) are necessary. A mesh with end-around connections is often called a *torus*. End-around connections may also be included in the bidirectional case to reduce the average number of hops a message must travel in the network. A torus can be organized so that all links are of equal length, with each link being about twice as long as in the case without end-around connections [8].

2.2. Organization of a Node

A node in the system typically consists of one or more processors, some associated local memory, and a hardware switch that controls the routing of messages through the node (Figure 2.2). When the node needs to send a message to another node, it queues the message in a local buffer (not shown in the figure). The message waits until the node-to-switch link (connecting the processor and memory to the local switch) becomes free, and then until space becomes available in the outgoing virtual channel buffer. (The message must compete for the channel buffer with messages from neighboring nodes that request the same buffer; we assume that the switch chooses among competing requests in first-come-first-serve order.) Thereafter, the message is forwarded down the link into the channel buffer in the pipelined manner of wormhole routing, which we describe next. We assume that the processor is not involved in transferring the message from the local buffer into the outgoing channel in the local switch.

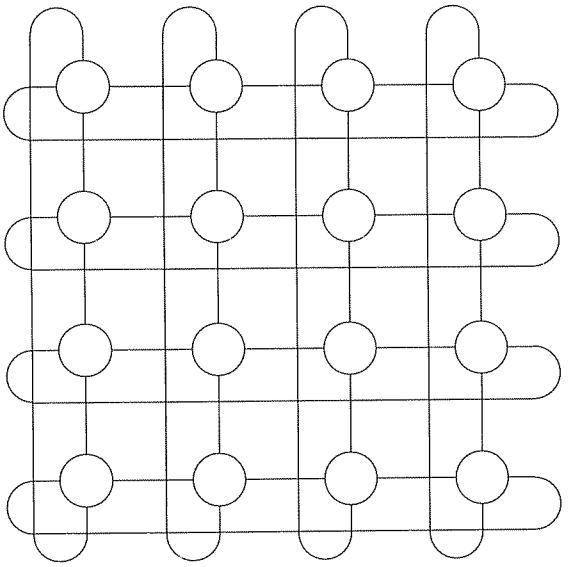


Figure 2.1. Basic mesh topology

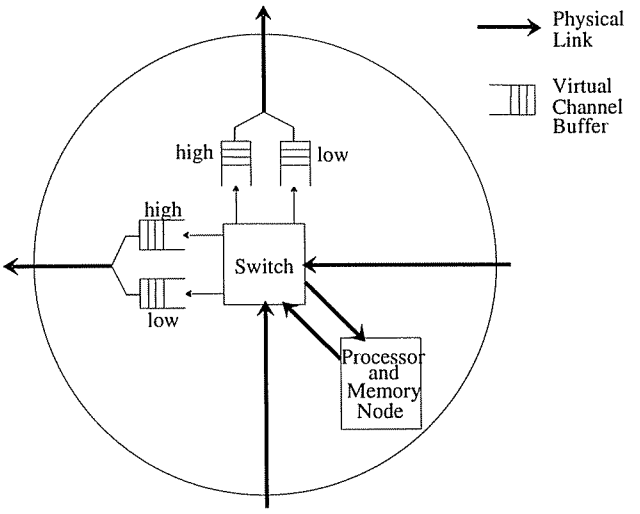


Figure 2.2. A node in a unidirectional network

2.3. Wormhole Routing

In *wormhole routing* a switch begins forwarding a message as soon as the header is received and the required channel buffer in the next switch can accept one or more flits of this message. Thus, the flits of a message are transmitted from one switch to the next in a pipelined fashion and may occupy several channels along the path from source to destination. Only the header flit of a message contains routing information. If the header flit of a message is blocked because the required buffer in the next switch along its path is full all the flits in the message are blocked and, therefore, so are the channels they occupy. If more than one flit can be buffered at a node, flits behind the header can "catch up" until the available buffer space is filled. At this point they block and can continue only after the header is unblocked. We assume this method of routing throughout the paper.

2.4. Deadlock Avoidance for Finite Buffers

In the ideal case, when buffer capacity is unlimited, deadlock cannot occur in the network and the wormhole routing scheme is equivalent to an optimized form[1] of the *virtual cut-through* routing algorithm defined for data communication networks [12]. In practice, buffer capacity in a node is limited and deadlock can occur in the networks with end-around connections because all the buffers in a cycle could be filled, with no message able

1. The virtual cut-through algorithm specifies that, if the header flit is blocked at a switch, the entire message has to be received before the message is forwarded. Instead, wormhole routing allows a partially received message to be forwarded as soon as its outgoing channel becomes available.

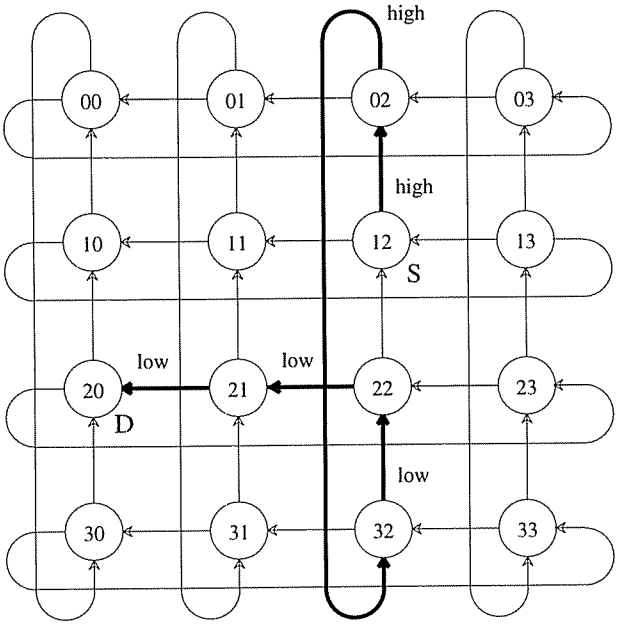


Figure 2.3. Example for deadlock-free routing algorithm
Unidirectional 4×4 torus
Channels on path from **S=12** to **D=20**

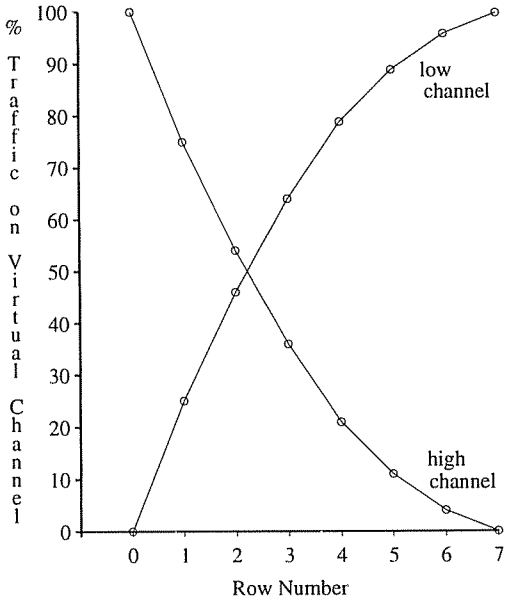


Figure 2.4. % Traffic on the high and low virtual channels along a column
Unidirectional 8×8 torus
Uniform traffic distribution.

to make progress along that cycle.

Dally and Seitz have proposed a deterministic routing scheme that uses the concept of *virtual channels* to break cycles and prevent deadlock in the networks with end-around connections [9]. In this scheme, each physical link is shared by two virtual channels that are fed by separate buffers. As long as both virtual channels have messages to send, they alternate their flits on the physical link. If one of the buffers is empty or blocked, the other channel can transmit continuously, using the entire link bandwidth. When a message is blocked, all the virtual channels occupied by the flits of that message are also blocked, and no other messages can use those channels.

The algorithm is illustrated in Figure 2.3, and operates as follows. Each node in the k-ary n-cube is assigned an n-digit, base-k number, which specifies the position of the node in the cube. Dimensions are numbered and messages are always routed in decreasing order of dimension. (For example, in Figure 2.3, d=1 for the columns, d=0 for the rows, and routing is column first.) In each dimension d, $d=n-1, \cdots, 0$, a message is routed in that dimension until it reaches a node whose dth digit agrees with the dth digit of the destination node. The message is routed along the "high" virtual channel if the dth digit of the destination address is greater than the dth digit of the present node's address. Otherwise, it is routed along the "low" virtual channel. For example, in a unidirectional mesh network (as in Figure 2.3), a message to a node with a higher row number is routed on the high virtual channel along the column until it crosses the link out of node on row 0 (shown at the edge of the network in the figure),

and thereafter uses the low virtual channel on the column until it reaches the destination row.

The algorithm imposes a total order on the virtual channels that are used in each direction along any dimension of the network. Furthermore, the requirement that messages are routed in decreasing order of dimension implies that no cycles exist across dimensions. The algorithm is thus deadlock-free because it imposes a partial order on all virtual channels in the network.

The above deadlock-free routing scheme generates asymmetric loads on the virtual channels in the network *even when all processors have a uniform message destination probability distribution* (i.e., even when the loads on the physical links are balanced). Figure 2.4 shows the fraction of total link traffic that uses each virtual channel for the links on a single column in a unidirectional 8×8 torus, assuming uniform message distribution. Note that all traffic on the link leaving the processor on row 0 uses the high virtual channel, and thus the buffer space of the low virtual channel is completely unused. In general, on a physical link near the "edge" row or column (after which traffic crosses over from the high to low channel or vice versa), the traffic tends to be concentrated on one of the two virtual channels. For links far away from the terminal row or column, the traffic is more evenly balanced on the two virtual channels. In parallel work, Bolding has recently observed the same phenomenon [4] and gives similar data as in Figure 2.4, showing the buffer utilizations on the high and low channels for bidirectional and unidirectional topologies.

2.5. Workload Assumptions

We assume that all the processors in the system execute subtasks of large MIMD parallel programs.[2] Most previous studies of mesh networks, the hypercube and other k-ary n-cubes, have assumed a message-passing workload [8, 10, 11], but a number of recent shared-memory systems have also been based on mesh networks (Alewife [1], Dash [14], Cray T3D). Our model of the network is applicable to both types of workloads.

We allow a processor to have a maximum of N_{out} requests outstanding before it is required to block for a reply. (N_{out} is a parameter of the model.) For the model and results described in this paper, we assume that the rate of generation of requests is independent of how many requests are outstanding, until the maximum of N_{out} is reached. This assumption can accurately capture the behavior of systems in which each processor can switch between multiple contexts [1, 21], and should also be a reasonable model for many message-passing workloads. The assumption may be somewhat more approximate for processors that permit non-blocking memory operations (e.g, as with buffered writes, non-blocking caches or prefetching), where the intervals between successive requests may depend in complex ways on the number of outstanding requests. Finally, as explained in Section 3, a

2. We do not explicitly model synchronization events. Instead, we assume that these are reflected in the rate at which processors generate messages.

simple modification would allow the model to capture the behavior of hierarchical multiprocessors [14] containing multiple processors per node (by allowing the rate at which a node generates requests be proportional to the number of additional requests it can make before blocking).

Our model does not restrict the communication patterns in the system. Each processor sends a message to each other processor with a specified probability, and these arbitrary probability distributions are inputs to the model. This permits us to study the effect of non-uniform traffic patterns on system efficiency.

The workload and system parameters used in the study are defined in Section 3.

2.6. Performance Issues for Mesh Networks

In Section 4, we use the model to study the performance as well as the scalability of mesh networks in varying configurations (various system sizes, buffer sizes, network topologies) and under different workloads (varying communication rates, single/multiple outstanding requests per processor, uniform/non-uniform communication patterns). We begin by examing the performance and scalability of a *baseline* system: a bidirectional torus with a uniform communication workload and processors that must block after each request (N_{out}=1). We then study a number of variations on this system, to evaluate several issues that arise in the design of mesh networks. The issues we examine are as follows.

Channel Buffer Size: The buffer size per network link or channel is a design parameter that has significant cost and performance implications. In studying various network design issues, we compare network performance with single-flit buffers per virtual channel against the performance with infinite switch buffers. These are extreme cases that bound the performance of any particular finite buffer size, and show how much can be gained by increasing switch buffer sizes.

Multiple outstanding requests: Allowing a node to have more than one request outstanding has the potential to at least partially hide the latency of remote communication, but there is also potential for higher congestion in the network. We investigate how much improvement in absolute system efficiency is possible with multiple outstanding requests (due to overlapping communication with execution) and whether at some point network congestion cancels this gain. We also investigate how system scalability is affected by allowing multiple outstanding requests per processor.

Mesh Topology: There are performance and cost tradeoffs between the three network topologies mentioned in Section 1. In a k-ary n-cube network, the mean number of links that a message must traverse assuming all other nodes are equally likely to be the destination is approximately $\frac{1}{4}n.k$, $\frac{1}{3}n.k$ and $\frac{1}{2}n.k$ for the bidirectional torus, the bidirectional mesh without end-around connections, and the unidirectional torus respectively.

The extra links in the bidirectional networks imply a higher cost, however, and this must be accounted for to allow a fair comparison between the various topologies. When comparing these networks, we assume that the number of input and output *wires per switch* is fixed, which implies that the channels of the unidirectional network can be twice as wide as those in either of the bidirectional networks, offsetting the larger mean number of hops required. Furthermore, the bidirectional mesh without end-around connections, unlike the torus networks, does not require the deadlock-prevention algorithm of Dally and Seitz since the fixed-dimension-first routing is sufficient to prevent cyclical dependencies between links in the network. To allow a fair comparison, we compare the torus networks with single-flit buffers per virtual channel to non-torus mesh networks with two-flit buffers per physical link. This ensures that the buffer capacity per switch is equal for all three topologies.

Locality of Communication: A large class of scientific algorithms, called *continuum models* [19], involve a grid structure where a particular variable depends only on its nearest neighbors. Such problems can be mapped to the mesh network so that any processor requires mostly values calculated by its four neighboring nodes (or some set of nodes situated within at most a few hops). This would reduce network latency and contention compared with uniform communication. We investigate how near-neighbor communication locality affects system performance and scalability, and re-evaluate the design issues discussed above under workloads exhibiting varying degrees of near-neighbor locality.

Communication hot-spots: It has been shown that communication hot-spots can seriously degrade the overall performance of indirect (eg., multistage) interconnection networks with non-blocking processors. Furthermore, in such systems, hot-spots can cause buffers to fill up in large portions of the network, severely increasing the latency of unrelated (non-hot) network traffic as well, a phenomenon called *tree-saturation* [17]. We use our model to study the effect of communication hot-spots in mesh networks, with processors that block after a limited number of outstanding requests. We study the degradation in overall system performance due to a hot-spot, as well as the effect of a hot-spot on the latency of other traffic in the network.

Performance imbalance caused by the deadlock-avoidance algorithm: In Section 2.4 we pointed out that the deadlock-free routing algorithm of Dally and Seitz generates asymmetric loads on the virtual channels in the network. The asymmetry does not necessarily imply that the processors near the edge are more adversely affected than the processors near the center of the mesh. The actual effect is complicated and requires careful reasoning about the pipelining effects of wormhole routing. A more detailed explanation of the asymmetry, and a quantitative analysis of its potential impact on performance, are given in Section 4.7.

Table 3.1. Model input parameters.

Parameter	Description
N	Number of processors in the system
N_{out}	Maximum number of requests a processor can have outstanding before it must block for a reply
τ	Mean time between messages when less than N_{out} requests are outstanding
F_{ij}	Fraction of messages by processor i that are directed to processor j, $\sum_{j \neq i} F_{ij} = 1$, $i = 1,...,N$
P_1, P_2	Probability that a message is type 1 ($msg\ 1$) or type 2 ($msg\ 2$) respectively
L_i	Length of message of type $i \in \{msg\ 1, msg\ 2, resp\ 1, resp\ 2\}$
$D_{mem,r}, D_{mem,w}$	Time to respectively read and write one word from a memory module

3. The Model

In order to study the design trade-offs outlined at the end of the previous section, we have created closed queueing network models of the k-ary n-cube network for each of two buffer sizes: finite buffers of size one flit, and infinite buffers. The parameters of the models are defined in Table 3.1. N_{out} denotes the number of outstanding requests that each processor can have before it blocks. For the model and analyses in this paper, we assume that when a processor has less than N_{out} requests outstanding, it generates request messages with a mean interval of τ cycles between requests. A request message generated by processor i is directed to processor j, $j \neq i$ with probability F_{ij}. We allow two sizes of messages to be generated: $L_{msg\,1}$ and $L_{msg\,2}$, with respective probabilities P_1 and P_2 (these probabilities are the same for all processors). The sizes of the respective responses are $L_{resp\,1}$ and $L_{resp\,2}$. For a shared-memory workload the two request message types could represent memory *read* and *write* requests, and the reply types could represent *data* and *acknowledgement* responses respectively. The network is assumed to operate synchronously. The values of τ, $D_{mem,r}$, and $D_{mem,w}$ are assumed to be in units of switch cycles.

In our models, each processor forms a class of customers with its own destination probability distribution and with population equal to N_{out}. In other words, each possible message a processor can have outstanding is modeled as a separate customer in the system. When there are $n < N_{out}$ requests outstanding, the remaining $N_{out}-n$ customers are served in FCFS order at the processor,[3] as in [22]. Thus, each customer in the system repeatedly performs the following actions:

– *execute* for an amount of time measured in switch cycles that is geometrically distributed with mean τ,

3. FCFS service at the processor is appropriate for systems such as those that maintain multiple contexts at each processor since only one context executes at any time. An *infinite server* would be more accurate for hierarchical systems. The equations for queueing at the processor can easily be modified for this case.

- *visit a remote node and return to the processor* (representing a remote memory access, or sending a message and receiving an acknowledgement),
- *queue at the processor* to resume execution.

We develop the equations assuming that a remote processor is not interrupted when it receives an incoming message, which would be true for a shared memory system. In this case, an incoming message only requires a memory access at the remote node. (The equations can easily be modified to reflect message processing by the node processor or message-handling co-processors.)

We choose to develop approximate Mean Value Analysis models because of the previous success of this technique for analyzing other interconnection networks with features that violate separable model assumptions [20, 22]. Approximate Mean Value Analysis is based on estimating the mean round-trip time, or cycle time, for each class of customers in the queueing network, relative to some reference point. The processor serves as the reference point for the residence time equations in our model. The mean round-trip time for a customer of class i is the sum of its mean residence times (queueing and service) in the local processor, in the network, and at the remote node:

$$R[i] = r_{proc}[i] + r_{network}[i] + r_{remote}[i], \quad i=1 \cdots N. \tag{1}$$

Each processor in the system has a distinct mean round-trip time because of non-uniform virtual channel loads as well as possibly non-uniform communication patterns.

The mean round-trip time in the network is the weighted sum of the mean times for the message and the response, for each type of request:

$$r_{network}[s] = \sum_{d \neq s} F_{sd}(P_1(r_{msg\,1,sd}+r_{resp\,1,ds}) + P_2(r_{msg\,2,sd}+r_{resp\,2,ds})), \quad s=1, \ldots, N, \tag{2}$$

where $r_{j,sd}$ is the mean time for a message of type j from node s to node d.

To calculate $r_{j,sd}$, we need to model the routing, pipelining, and blocking of messages in the network. These features require an approximate model solution. Our model for systems with infinite channel buffers is similar to the model developed for Banyan networks in [22]. Their equations assume that processor cycles are required to transfer a message into the network; we do not make this assumption. The only other difference is that we use a somewhat more accurate technique to estimate residence times at the processor for $N_{out}>1$. This technique is also employed in our finite buffer model, and is discussed in Section 3.3 below. Otherwise, we do not give the model equations for the infinite buffer case.

For our model of wormhole routing with single-flit channel buffers, we have developed new approximations to estimate (i) the channel waiting and blocking times, (ii) the customer queueing time at the processor, and (iii) the mean queue length seen at the first outgoing link when multiple channels connect the processor to its switch. Below we present an overview of the model for networks with single-flit buffers, and then describe each of the

three new approximations. Our notation is summarized in Appendix A and the full set of model equations for the model with single-flit channel buffers is given in Appendix B. In Section 4 we discuss the results of validating the model and the results of analyses using the model. The validation studies show that the model is accurate over a wide range of input parameter values.

3.1. Overview of the Model with Single-Flit Buffers

Since messages in the mesh network can occupy several channels simultaneously, the mean message residence time, $r_{j,sd}$, is the sum of the following three terms:

(i) the mean waiting time for the link from the node to its switch, ($w_{node,sd\,|\,j}$),

(ii) the mean residence time for the *header* flit on each virtual channel c between s and d ($r_{j,c,sd}[1]$), and

(iii) the mean delay until the remaining flits of the message reach d ($T_{catchup}$):

$$r_{j,sd} = w_{node,sd\,|\,j} + \sum_{c} r_{j,c,sd}[1] + T_{catchup}, \quad j \in \{\text{msg1, msg2, resp1, resp2}\}, \tag{3}$$

where the summation is over virtual channels, c, on the path from s to d, including the channels out of the processor at s and into the processor at d.[4] Note that the above equation is similar in form to the equation used in [2] and [22]. One difference between our equation and the corresponding one in [2] is that we include the waiting time for the link that connects the processor to the switch, not just for the first switch buffer. A difference between our model and both [2, 22] is that $T_{catchup}$ is not deterministic, since at each link the flits may or may not have to alternate with flits on the link's other virtual channel. To compute $T_{catchup}$, we assume that the probability a flit must share a link is approximately equal to the utilization of the link by messages on the other virtual channel mapped to the link. Appendix B contains the details.

Further development of the model equations requires new techniques for estimating $r_{j,c,sd}[1]$, r_{proc}, and the waiting time for the first network virtual channel when there are multiple channels from processor to switch. These approximations are motivated and outlined in Sections 3.2 - 3.4 respectively. Section 3.5 concludes with a discussion of the model complexity.

3.2. Mean Channel Residence Time ($r_{j,c,sd}[k]$)

Let $r_{j,c,sd}[k]$ denote the average residence time of the k^{th} flit of a type j message from s to d on channel c. The mean residence time for a header flit (k=1) on channel c is itself the sum of three terms:

(i) the average waiting time for the next channel on the path from s to d (we denote this channel by $(c+1)_{sd}$),

4. Henceforth, we use variables to denote the number of a virtual channel, and ensure that the appropriate number or set of numbers in a summation is clear from the context.

(ii) the average waiting time for a flit on the virtual channel that is multiplexed onto the same physical link as c; we denote this channel by $\bar{c}$, and approximate this term by $u_{link,\bar{c}}$, the mean utilization of the link by messages on $\bar{c}$ (i.e., the fraction of time the link is actually transmitting flits from $\bar{c}$), and

(iii) the one cycle for transferring the flit to the next queue:

$$r_{j,c,sd}[1] = w_{(c+1)_{sd}|I} + u_{link,\bar{c}} + 1, \tag{4}$$

where a message from s to d enters $(c+1)_{sd}$ via input port I. The possible input ports to a virtual channel are the virtual channels from neighboring switches or the channel from the processor at the current node. The waiting time for $(c+1)_{sd}$ is a function of the input port because the traffic on $(c+1)_{sd}$ coming from the various input ports is asymmetric, in general.

For flits numbered k, $k>1$, if d is k or more steps away from c, the mean residence time on c is estimated by the mean residence time of the header flit on channel $(c+k-1)_{sd}$ (the channel k-1 steps ahead on the path to d), plus waiting for a flit that might be on $\bar{c}$. Otherwise the header flit has already reached the destination and the residence time of the k^{th} flit is one plus the mean waiting time for a flit on $\bar{c}$:

$$r_{j,c,sd}[k] = \begin{cases} r_{(c+k-1)_{sd}}[1] + u_{link,\bar{c}} & d \text{ is } k \text{ or more steps away from channel } c \\ 1+u_{link,\bar{c}} & \text{otherwise} \end{cases} \qquad (k > 1). \tag{5}$$

The key question for the model is how to calculate $w_{c|I}$, the waiting time for virtual channel c experienced by a header flit of a message that enters c via input port I. This waiting time is the sum of three terms:[5]

(i) the mean residual residence time of a message in service at c that arrived via some other input port $i \neq I$, if any,

(ii) the mean residual residence time for the last flit of a message in service at c from port I, if any, and

(iii) the mean time to serve messages waiting to use c from other input ports (at most one per port):

$$\begin{aligned} w_{c|I} = & \sum_{i \neq I} \sum_{j} \sum_{k=1}^{L_j} u_{j,c,i}[k] \left[\frac{1}{2} r_{j,c,i}[k] + \sum_{l=k+1}^{L_j} r_{j,c,i}[l] \right] \\ & + \sum_{j} \left[\frac{u_{j,c,I}[L_j]}{1 - \sum_{j'} \sum_{l=1}^{L_{j'}-1} u_{j',c,I}[l]} \times \frac{r_{j,c,I}[L_j]}{2} \right] \\ & + \sum_{i \neq I} \sum_{j} \left[u_{j,(c-1)_i}[1] \sum_{l=1}^{L_j} r_{j,c,i}[l] \right] \end{aligned} \tag{6}$$

The calculation of each of these terms is explained below.

5. To denote a summation over all four types j we write $\sum_{j}$ instead of $\sum_{j \in \{msg1,msg2,resp1,resp2\}}$

(i) The total residence time of a message at c is random with an unknown distribution. Rather than assume knowledge of this distribution to calculate the mean residual life of a message in service, we assume that the residence time of *each flit* of a message is deterministic, i.e. its mean residual life is $r_{j,c,i}[k] / 2$. We expect that this assumption will be good for low to moderate network traffic, and will introduce only small error at higher loads since a flit residence time is small compared with total message residence time. Thus, the mean residual residence time of an entire message (seen by an arrival on input port I) can be calculated by conditioning on the event that the arrival finds the k^{th} flit of a type j message that arrived from input port i in service at channel c. The probability of this event is approximated by the average utilization of c by such a flit: $u_{j,c,i}[k]$. The mean residual life of the message in this case is $(\frac{1}{2} r_{j,c,i}[k] + \sum_{l=k+1}^{L_j} r_{j,c,i}[l])$. Summing over all flits $1 \le k \le L_j$ for all message types j, and all possible input ports $i \neq I$, gives the first term in the above equation.

(ii) Because of the pipelined routing scheme, if the tagged message arriving at c via input port I finds another message at c that also arrived via I, then it can only find the tail flit of that message occupying c, and *cannot find any other flit of the message*. Therefore, we approximate the second term by the ratio of time that channel c is occupied by a tail flit from I ($u_{j,c,I}[L_j]$) to the total time that channel c is *not occupied* by any other flits from I $(1 - \sum_{j'} \sum_{l=1}^{L_{j'}-1} u_{j',c,I}[l])$. The residual residence time in this case is just $r_{j,c,I}[L_j] / 2$. Summing over j gives the second term.

(iii) The third term is the average waiting time for messages that are waiting on input ports other than i when a message arrives at input port I (we assume that these will be transmitted by channel c before the arriving message). For each input port $i \neq I$, the probability that a type j message is waiting to use c is approximated by the utilization of channel $(c-1)_i$ by header flits of type j messages *that will next use* c : $u_{j,(c-1)_i}[1]$. Multiplying by the total residence time of such a message and summing over $i \neq I$ and all message types j gives the third term.

The remaining unknowns in the above equations ($u_{j,c,i}[k]$, $u_{link,\bar{c}}$) are calculated using previously developed MVA techniques [20, 22], as described in Appendix B.

3.3. Processor Residence Times ($r_{proc}[i]$)

The processor is modeled as an FCFS queueing center, where the service time is geometrically distributed with a mean value of τ cycles. In early model validation experiments, we found that the widely-used Schweitzer approximation for product-form networks [18] is not sufficiently accurate for the processor queues since the customer population for each processor, N_{out}, can be small. Furthermore, previously developed approximations such as Linearizer [6], which achieve greater accuracy by solving the equations at a few neighboring populations, introduce too much additional complexity into the model. Below we develop a new approximation for $r_{proc}[i]$ that is

empirically accurate and yet requires very little additional computation when N_{out} is not large. The key idea is that we solve for $r_{proc}[i]$ recursively (i.e., similar to exact Mean Value Analysis) without recursively solving for the residence times at other queues in the system for each customer population. Empirically, we found the new approximation to be considerably more accurate than the Schweitzer approximation. Furthermore, we note that the approximation is applicable to any multiclass queueing network where all the demand at some queue comes from a small fraction of the customers in the network. As far as we are aware, this approach has not been previously reported in the literature.

Consider some processor i. Define $r_{proc}(i,n)$ to be the steady-state average residence time at processor i if there are n customers in its class. Thus, $r_{proc}[i] = r_{proc}(i,N_{out})$. Similarly, let $q_{proc}(i,n)$ and $u_{proc}(i,n)$ denote the mean queue length and the mean processor utilization at processor i with n customers in its class. $r_{proc}(i,n)$ is the sum of the mean service time (τ), the mean waiting time for customers found waiting in the queue, and the mean residual service time (res_{proc}) for the customer in service, if any. We estimate the mean queue length and processor utilization *seen by an arriving customer* by $q_{proc}(i,n-1)$ and $u_{proc}(i,n-1)$ respectively (just as in exact MVA), producing a recursion over $n = 1, \cdots, N_{out}$. The key to the approximation is that $q_{proc}(i,n-1)$ and $u_{proc}(i,n-1)$ are calculated using the same values of $r_{network}[i]$ and $r_{remote}[i]$, for all $n = 1, \cdots, N_{out}$. (These values are available from the previous iteration in the numerical solution of the overall model.) Thus,

$$r_{proc}(i,n) = \tau + [q_{proc}(i,n-1) - u_{proc}(i,n-1)] \times \tau + u_{proc}(i,n-1) \times res_{proc}, \quad n > 1, \tag{14}$$

$$q_{proc}(i,n) = \frac{n \times r_{proc}(i,n)}{r_{proc}(i,n) + r_{network}[i] + r_{remote}[i]}, \quad n > 1, \tag{15}$$

$$u_{proc}(i,n) = \frac{n \times \tau}{r_{proc}(i,n) + r_{network}[i] + r_{remote}[i]}, \quad n \geq 1, \quad \text{and} \tag{16}$$

$$q_{proc}(i,1) = u_{proc}(i,1) = \frac{\tau}{\tau + r_{network}[i] + r_{remote}[i]}. \tag{17}$$

(The equations are numbered to correspond to Appendix B.) The mean residual service time of a customer found in service, res_{proc}, has to be calculated as seen by the tail flit, not the header flit, since the customer is queued up at the processor only when its tail flit arrives. Conditioned on finding a customer in service, the mean residual service time seen by the head is τ, by the memoryless property of the geometric distribution. A returning message is of length L_{resp1} with probability P_1 and L_{resp2} with probability P_2. In one cycle, the probability that the customer in service at the processor does not complete service is $\gamma = 1 - (1/\tau)$. Therefore, the average residual service time seen by the tail flit is approximated by:

$$res_{proc} = (\tau - 1) \times (P_1 \gamma^{L_{resp1}-1} + P_2 \gamma^{L_{resp2}-1}) \tag{18}$$

Equations 14-18 are solved for each processor i separately, in order to calculate all the $r_{proc}[i]$.

3.4. Waiting Time for First Virtual Channel With Multiple Processor-to-Switch Channels

In our mesh network analysis, we found the physical link that connects each processor to its associated switch to be a bottleneck under high loads, in the network with single-flit channel buffers. Therefore, we investigated the use of multiple physical links connecting the processor to the switch, one for each outgoing virtual channel from the switch to neighboring nodes. (In practice, only two to three physical processor-to-switch links should provide about the same performance since almost all the messages out of a processor are concentrated on a few outgoing virtual channels.) With this organization, when a message from the processor finds its link to the switch busy and later reaches the head of the queue for this link (where it is has to wait for the outgoing virtual channel buffer), it cannot find a message from any other input port occupying the channel; it can find only the tail flit from the preceding message. Furthermore, such an arriving message is more likely to find waiting messages at other input ports (which were blocked by the preceding message). These observations lead to a somewhat different expression for the waiting time for the outgoing channel (i.e., the first virtual channel along the path of the message) in the case of multiple processor-to-switch links.

Consider a tagged customer of class q, and let c denote the first virtual channel along its path. Define $w'_{c|q}$ to be the average time this message has to wait before entering channel c. As in Equation (6), $w'_{c|q}$ is the sum of three terms:

- the mean waiting time for messages in service at c that arrived from input ports $i \neq PROC$, if the tagged message found the processor-to-switch link idle (*PROC* denotes the input port used by messages arriving to c from the processor),
- the mean waiting time for the tail flit of previous message from *PROC*, if the tagged message found the processor-to-switch link busy, and
- the mean waiting time for messages blocked at input ports $i \neq PROC$ that are waiting to use channel c:

$$w'_{c|q} = \sum_j \Bigg\{ \sum_{i \neq PROC} \sum_{k=1}^{L_j} b'_{j,c,i|q}[k] \left[\frac{r_{j,c,i}[k]}{2} + \sum_{l=k+1}^{L_j} r_{j,c,i}[l] \right] + \left[\sum_{k=1}^{L_j} b_{node,j,c|q}[k] \right] (r_{j,c|PROC}[L_j] - 1) + \sum_{i \neq PROC} \left[q'_{j,c,i|q} \sum_{k=1}^{L_j} r_{j,c,i}[k] \right] \Bigg\} \tag{24}$$

The form of the first term for $w'_{c|q}$ is the same as the first term in equation (6), with $u_{j,c,i}[k]$ replaced by $b'_{j,c,i|q}[k]$. Here, $b'_{j,c,i|q}[k]$ is the probability that a message of class q finds channel c busy serving the k^{th} flit of a type j message that arrived via input port $i \neq PROC$, $j \in \{msg\,1, msg\,2, resp\,1, resp\,2\}$. This is estimated as follows:

$$b'_{j,c,i\,|q}[k] = \frac{1 - Pr\{\text{processor-to-switch link is busy}\}}{\text{fraction of time } c \text{ not serving message coming from node}} \times u_{j,c,i}[k]$$

$$= \frac{1 - \sum_{j}\sum_{k=1}^{L_j} b_{node,j,c\,|q}[k]}{1 - \sum_{j}\sum_{k=1}^{L_j} u_{j,c,PROC}[k]} \times u_{j,c,i}[k], \tag{25}$$

where $b_{node,j,c\,|q}[k]$ denotes the probability that a class q message finds the processor-to-switch link corresponding to outgoing channel c busy serving the k^{th} flit of a type j message.

The second term in $w'_{c\,|q}$ is straightforward because the probability that the processor-to-switch link for channel c is found busy with a message of type j is merely $\sum_{k} b_{node,j,c\,|q}[k]$, and the tagged message must see all but one cycle of the residence time of the *last* flit of the message ahead of it.

For the third term, $q'_{j,c,i\,|q}$ is defined as the average number of messages of type j from $i \neq PROC$ found waiting to use c by the arriving class q message. The calculation of $q'_{j,c,i\,|q}$ requires another observation about the blocking phenomena in the mesh. A message following another message out of the node and into c is more likely than a random arrival at c to find a message on input port i already waiting for channel c. To account for this, we calculate the probability that a random message using c via $I=PROC$ blocks a type j message on incoming virtual channel i. (The latter message will then be waiting to use c when the following message from $I=PROC$ reaches the head of its queue for c.) Similarly, we consider messages from each $i' \neq PROC$ (and $i' \neq i$) blocking messages from i to c. Therefore, $q'_{j,c,i\,|q}$ is as follows:

$$q'_{j,c,i\,|q} = \left\{\sum_{j}\sum_{k=1}^{L_j} b_{node,j,c\,|q}[k]\right\} \times Pr\{\text{ message from } I=PROC \text{ blocks a type } j \text{ message from } i \text{ }\} +$$

$$\sum_{\substack{i' \neq PROC \\ i' \neq i}} \left\{\sum_{j}\sum_{k=1}^{L_j} b'_{j,c,i\,|q}[k]\right\} \times Pr\{\text{message from } i' \text{ blocks a type } j \text{ message from } i \text{ }\} \ . \tag{26}$$

We illustrate the calculation of one of these terms here. A key observation we make is that $Pr\{$a message from $I=PROC$ blocks a type j message from $i\}$ is proportional to the relative number of messages to c that arrive from i and I respectively (counting only type j messages from i). But this relative number is exactly the ratio of the visit ratio of type j messages from i to c to the visit ratio of all message types from $I=PROC$ to c. This ratio of visit ratios, multiplied by the probability that a random message from i is blocked by a message from $I=PROC$ then gives us the probability that a random message from I to c blocks a message on i. Thus, define $V_{j,c,I}$ to be the sum of the visit ratios of customers of all classes as type j messages to channel c via input port i. Then,

$$Pr\{\text{ message from } I=PROC \text{ blocks a type } j \text{ message from } i \text{ }\} = \frac{V_{j,c,i}}{\sum_{j'} V_{j',c\,|I}} \left\{ \sum_{j}\sum_{k=1}^{L_j} u_{j,c,I}[k] + \sum_{j} u_{j,(c-1)_I}[1] \right\}$$

where the summations in the parentheses sum to the probability that a random message arriving to c from i has to wait for a message from I=*PROC* (which may be in service at c, or blocked on the processor-switch link to c).

The above discussion highlights the main points of this new heuristic used to calculate the waiting time for the first virtual channel. The complete equations are given with the rest of the model in Appendix B.

3.5. Complexity of the Models

The model for wormhole routing with single-flit buffers has $O[L_{max}N^3]$ time complexity and $O[L_{max}N^2]$ space complexity, where L_{max} is the length of the longest message type. As a result, the model with L_{max}=11 and N=64 cannot practically be run on systems with fewer than ten megabytes of main memory. Nevertheless, solving the model is still about ten to a hundred times faster than simulating the wormhole routing protocol under a statistical workload. Furthermore, the model allows us to explore various issues and design trade-offs under the realistic assumptions of arbitrary message sizes, and blocking due to finite buffers. For example, the effects of asymmetric channel loads and hot spot traffic are a direct result of limited buffer space for the channels. Finally, the model with infinite buffers is highly efficient and can be used to explore many of the mesh network design trade-offs for larger systems.

4. Results

In this section we describe the results of extensive analyses of two-dimensional networks using the above models. We assumed a shared-memory workload for these experiments, as discussed below. We first present the ranges of input parameter values used in our study (Section 4.1), and the results of validation experiments (Section 4.2). In Section 4.3, we evaluate the performance and scalability of a baseline system that we use as a reference point for studying further network design issues. In Section 4.4, we study the impact of allowing multiple outstanding requests. In Section 4.5, we compare the alternative mesh topologies. In Section 4.6, we study the performance impact of near-neighbor workloads, and re-evaluate the design issues studied in Sections 4.3-4.5 under such workloads. In Section 4.7, we study the degradation in system performance due to communication hot-spots. Finally, in Section 4.8, we analyze the imbalance in processor efficiencies caused by the asymmetric loads on the virtual channels in the deadlock prevention algorithm (see Section 2.4).

4.1. Model Input Parameter Values and Performance Measures

The measures of system performance we use are individual and average processor efficiency, defined as the fraction of time a processor spends doing useful work:

Table 4.1. Parameter Values used in the Experiments.

Symbol	Range of Values	Symbol	Value	
			Bidirectional	Unidirectional
N	16,64,144-1024	$L_{msg\,1}$	3	2
N_{out}	1 - 8	$L_{resp\,1}$	9	5
τ	20-200	$L_{msg\,2}$	11	6
F_{nn}	0 - 100%	$L_{resp\,2}$	3	2
F_{hot}	0 - 20%	P_1, P_2	0.8, 0.2	0.8, 0.2
		$D_{mem,\,1} = D_{mem,\,2}$	4	4

$$E[i] = \frac{N_{out} \times \tau}{R[i]} \leq 1, \qquad \bar{E} = \frac{1}{N} \sum_{i=1}^{N} E[i].$$

Other measures that are obtained from the model equations include steady state mean channel queue lengths and steady state link utilizations. We validated the accuracy of several of these detailed measures as well.

The ranges of values used for the various model input parameters are given in Table 4.1. Most of the experiments with finite-buffer systems focus on a 64-processor (8×8) mesh, while we use the infinite-buffer model to examine the performance impact of increasing system size for systems as large as 1024 processors (a 32×32 mesh). All processing times and memory access times are specified in units of switch cycles. In many of the graphs, processor efficiency is plotted as a function of $1/\tau$, where $1/\tau$ intuitively measures the average communication rate (e.g., cache misses per cycle per processor). τ was varied from 20 to 200 cycles. Values of τ higher than 200 showed very little further improvement in processor efficiency (for our parameter settings). Also, since average remote access latencies are typically greater than 20 switch cycles in an 8×8 mesh, it is difficult to envisage programs that make requests faster than about one every 20 cycles executing with any reasonable efficiency on this or larger systems. Thus, we believe the above range of τ should allow us to study the performance of a fairly wide range of programs.

Messages are assumed to consist of a header flit, plus address flits (type *msg* 1), data flits (type *resp* 1), address and data flits (type *msg* 2), or acknowledgement flits (type *resp* 2). These interpretations of the message contents and the associated message lengths in Table 4.1 are intended to represent a shared memory workload. Message-passing programs could be expected to exchange larger messages between processes, though less frequently [8]. The models can be modified to study such workloads; however, that is beyond the scope of this paper. The message sizes also reflect the assumption that the channels in the unidirectional torus are twice as wide as in the bidirectional networks with an equal number of wires per switch. Finally, we set $P_1 = 0.8$, P_2=0.2, and $D_{mem,r} = D_{mem,w} = 4$ for all experiments. We do not expect moderate changes in these parameters to significantly alter our results.

4.2. Validation of the Models Against Simulation

We used event-driven simulators to validate the analytical models, for both the single-flit and infinite buffer cases. The simulators use a statistical workload identical to that of the analytical models, but implement the wormhole routing of the flits and the deadlock-free routing algorithm exactly. We present representative results of the validations of the single-flit buffer model in Tables 4.2 and 4.3.

At low to moderate network loads, the average processor efficiency from the analytical model agrees closely with the value obtained by simulation (less than 3% error). Thus our finite buffer model has accuracy similar to the very accurate, less complex models of the infinite buffer case; validations of the infinite buffer model gave results very similar to the model in [22] and are not shown here.

In cases of high network contention (eg., with $N_{out} \geq 4$ and $\tau < 50$), the analytical single-flit buffer model tends to be somewhat optimistic. In these cases some links are nearly saturated and the absolute value of the processor efficiency tends to be very low. The maximum error in average processor efficiency across all our validation experiments was 20%, which is shown for N=16, τ=5 and N_{out} = 8 in Table 4.2(a). In all cases, the predicted efficiencies are qualitatively correct.

We also examined more detailed performance measures, including estimates of the individual, asymmetric processor efficiencies. The maximum, minimum, and ratio of maximum to minimum processor efficiency predicted by the analytical model and simulation for the 8×8 torus are shown in Table 4.3. Again, agreement is very good, particularly for the max/min efficiency ratio. Note also that the ratio of the two efficiencies is always underestimated by the analytical model. Thus, the imbalance estimates, discussed in section 4.8, are generally *conservative*.

4.3. Baseline System Performance

We choose as our "baseline system" (which we will use as a reference point for studying further network design issues), the bidirectional torus with uniform traffic, and processors that block after each request (i.e., N_{out} = 1). In Figure 4.1, the solid lines show the average processor efficiency as a function of request rate ($1/\tau$) for the baseline system with single-flit channel buffers, and system sizes of 16 and 64 processors. The performance of this system is low at moderate or high request rates ($1/\tau$>0.03). The poor performance in this system is chiefly caused by inherent latency of communication rather than by contention in the network. To show this, we also give the efficiency curves assuming there is no contention in the network (the dashed lines in Figure 4.1). Comparing the two sets of curves, we see that the absolute loss in efficiency due to contention is about 5-10%. Thus, the system performance is latency- rather than bandwidth-limited when processors block after each request.

Table 4.2. Comparison of Overall Performance Estimates with Simulation.

(a) Bidirectional 4×4 Torus, Single Channel from Processor to Switch.

Parameters		Processor Efficiency			Network Residence Time		
N_{out}	τ	Simulation	Analytical	% Error	Simulation	Analytical	% Error
1	5	12.37	12.01	-2.9%	22.62	24.70	9.2%
1	25	43.01	42.29	-1.7%	21.07	22.44	6.5%
1	100	76.54	76.02	-0.7%	19.59	20.15	2.8%
2	5	19.56	19.58	0.1%	31.05	33.18	6.8%
2	25	68.93	70.26	1.9%	24.90	26.55	6.6%
2	100	96.34	98.60	2.3%	20.21	20.69	2.4%
4	5	24.11	26.12	8.3%	60.81	57.29	-5.8%
4	25	91.48	98.30	7.4%	33.64	33.63	-0.0%
4	100	99.96	100.0	0.0%	20.44	20.75	1.5%
8	5	25.07	30.08	20.0%	136.79	112.53	-17.7%
8	25	99.56	100.0	0.4%	44.89	34.44	-23.3%
8	100	100.0	100.0	0.0%	20.43	20.76	1.61%

(b) Bidirectional 8×8 Torus, Multiple Channels from Processor to Switch.

Parameters		Processor Efficiency			Network Residence Time		
N_{out}	τ	Simulation	Analytical	% Error	Simulation	Analytical	% Error
1	20	34.6	34.24	1.0 %	32.98	33.54	1.7%
1	33.3	48.9	48.37	-1.1%	30.03	30.72	2.3%
1	50	60.4	59.92	0.8%	27.94	28.61	2.4%
1	100	76.9	76.50	0.5%	25.36	25.88	2.0%
4	20	44.8	49.54	10.5%	144.06	138.57	-3.8%
4	33.3	75.01	84.16	12.2%	105.88	90.55	-14.54%
4	50	97.8	100.0	2.25%	49.20	38.55	-21.6%
4	100	100.0	100.0	0.0%	27.61	27.35	-0.9%

Table 4.3. Comparison of Maximum and Minimum Processor Efficiency Estimates with Simulation.

Bidirectional 8×8 Torus, Single channel from processor to switch.

Parameters		Maximum Efficiency			Minimum Efficiency			Max / Min Efficiency		
N_{out}	τ	Sim	Anal	% Error	Sim	Anal	% Error	Sim	Anal	%Error
1	25	37.78	38.92	3.0%	35.56	38.53	8.4%	1.062	1.010	-4.9%
1	40	50.39	52.55	4.3%	48.93	52.25	6.8%	1.030	1.005	-2.4%
1	100	73.90	76.13	3.0%	72.25	76.03	5.2%	1.023	1.001	-2.1%
4	25	63.76	68.62	7.6%	42.79	47.53	11.1%	1.490	1.444	-3.1%
4	40	88.36	96.00	8.6%	76.77	88.62	15.4%	1.151	1.080	-6.2%
4	100	99.96	100.0	0.0%	99.90	100.0	0.1%	1.001	1.000	-0.1%

Furthermore, because communication latency is the chief cause of low efficiency, *increasing buffer sizes per switch yields very little performance improvement for these system sizes.* In fact, for these systems, the average processor efficiency with infinite channel buffers (shown in Figure 4.2 and discussed below) is almost identical to the performance with single-flit channel buffers.

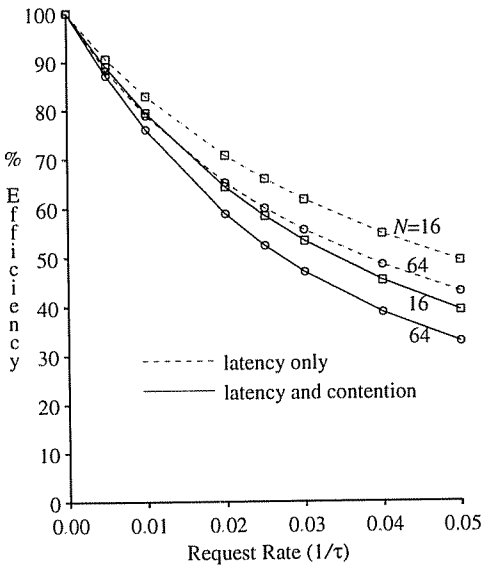


Figure 4.1. Efficiency of baseline system
Bidirectional torus, single flit buffers, uniform traffic, $N_{out} = 1$.

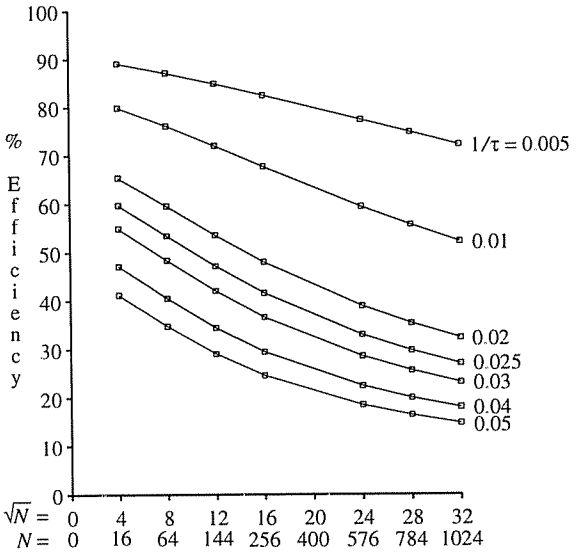


Figure 4.2. Scalability of baseline system
Bidirectional torus, infinite buffers, uniform traffic, $N_{out} = 1$.

To examine how system performance scales with increasing system size, in Figure 4.2 we plot the average processor efficiency as a function of mesh radix ($\sqrt{N}$) for different request rates ($1/\tau$), for the baseline system with infinite channel buffers. The figure shows that the performance of the baseline system scales well (i.e. average processor efficiency decreases slowly) with increasing system size, even though the absolute performance is low. These curves also show that the decrease in efficiency with increasing radix is primarily due to higher latency rather than due to network contention. Specifically, the decrease in efficiency is close to linear, showing that it is primarily due to the increasing number of hops a message must travel, rather than an increase in the delay (due to contention) at each hop. We conjecture that a baseline system with small channel buffers will also show good scalability, based on the low network contention seen in all the cases studied above. (The space complexity of our single-flit buffer model and the time requirements of simulation have prohibited us from testing this directly.)

4.4. Multiple Outstanding Requests

Since baseline system performance is chiefly limited by communication latency rather than contention, a plausible technique for improving processor efficiency is to allow processors to make multiple requests before blocking. The impact of multiple outstanding requests on system performance and scalability are as follows.

Figure 4.3 shows how the performance of an 8×8 baseline system (i.e., a bidirectional torus, uniform traffic) with single-flit or infinite buffers improves as N_{out} increases from 1 to 8. The figure shows that for single-flit

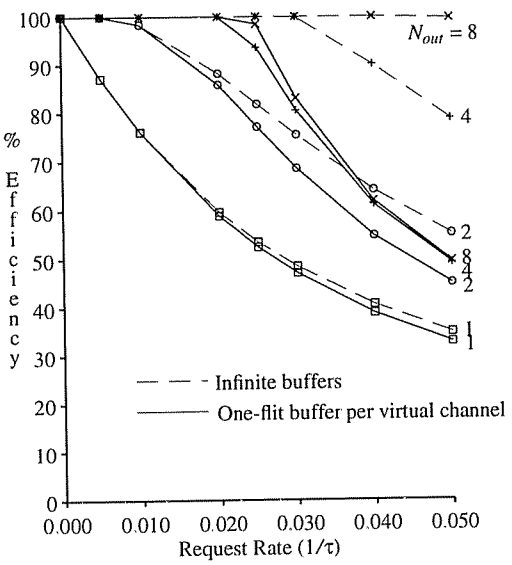


Figure 4.3. Efficiency with multiple outstanding requests
Bidirectional torus, uniform traffic, $N = 64$.

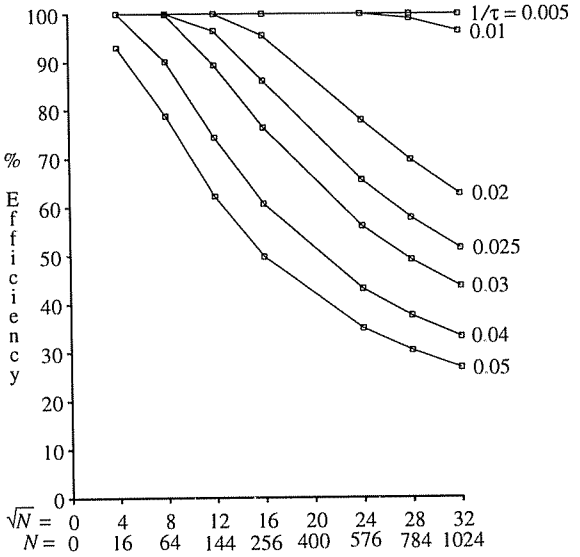


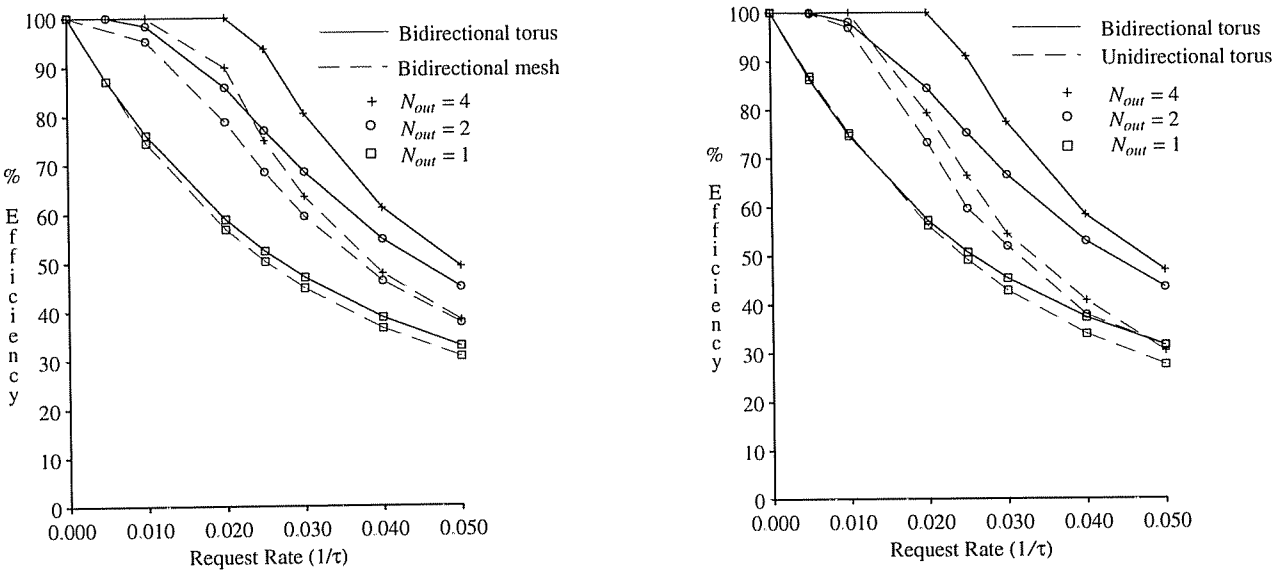
Figure 4.4. Scalability with 4 outstanding requests
Bidirectional torus, uniform traffic, infinite buffers.

channel buffers (solid lines), hiding communication latency with small increases in N_{out} is clearly effective in improving average efficiency, but each additional increase in N_{out} brings diminishing returns because of increasing network contention. In fact, there is a threshold at $N_{out} = 4$ beyond which no appreciable improvement in performance is observed.

For the infinite buffer case (dashed lines), we find that increasing N_{out} up to 8 is worthwhile for this system size. In larger systems with infinite buffers (not shown here) we again found that, beyond some threshold, increasing N_{out} brings little improvement; this threshold is about 8 and 4 for systems with 144 and 1024 processors respectively. *In general, a few contexts per processor or a few prefetches are effective in improving efficiency, but there is a clear threshold at a small value of N_{out} beyond which no further improvement is observed because of increased network contention.* These results further support conclusions in previous papers that a few contexts per processor are sufficient in systems that are being prototyped today [1, 21].

The figure also shows that larger channel buffers become increasingly important as N_{out} is increased, because of the increasing contention. The performance difference between single-flit and infinite channel buffers is significant even for $N_{out} = 2$ and becomes quite large for $N_{out} \geq 4$.

Because of the increased contention, it is important to re-evaluate the scalability of the network with multiple outstanding requests. In Figure 4.4 we plot processor efficiency against mesh radix, $\sqrt{N}$, for $N_{out} = 4$, infinite channel buffers, and various values of $1/\tau$. In contrast with Figure 4.2, efficiency drops sharply for moderate or



(a) Bidirectional torus vs. mesh

(b) Bidirectional torus vs. unidirectional torus

Figure 4.5. Comparison of network topologies.
8×8 system, uniform traffic, single-flit buffers.

high request rates ($1/\tau \geq 0.02$), because of increasing network contention. Thus, *the system with four outstanding requests does not scale well under uniform traffic, even with infinite channel buffers*, because network bandwidth in larger systems does not increase in proportion to the increased communication load. In the next several sections, we focus on systems with $N_{out} = 1$ and $N_{out} = 4$ when studying further network design trade-offs.

4.5. Alternate Mesh Network Topologies

We next compare the performance of the different network topologies under uniform communication. In this sub-section, we use the term "mesh" specifically to refer to the network without end-around connections. Our first comparison is between the two bidirectional networks: the torus and the mesh. The average number of hops a message must travel assuming uniform traffic is about 33% larger without the end-around connections. On the other hand, as explained in Section 2.6, the mesh does not require multiple virtual channels per physical link, as required by the deadlock-prevention algorithm for the torus. We therefore use a buffer size of 2 flits per physical link in the mesh network, to ensure a fair comparison with the torus with a single-flit buffer per virtual channel.[6]

6. Since our single-flit-buffer analytical model does not extend to networks with two-flit buffers, we used simulation to estimate the performance in the mesh with two-flit buffers per link.

Figure 4.5(a) plots processor efficiency versus request rate for one, two, and four outstanding requests, for each of the two topologies. The results are for an 8×8 system. For N_{out} = 1, there is only slight benefit to the end-around connections because the higher number of hops in the mesh has only a small effect on latency, due to the pipelined routing of messages and the low contention per hop. For N_{out} = 4, however, the torus has up to 30% higher performance because the higher network contention makes the higher number of hops in the mesh more significant. Thus, *end-around connections significantly improve performance with multiple outstanding requests.* This result should hold as well for larger systems (where the savings in hops for the torus increases) and larger buffer sizes (where network contention is still significant for N_{out} = 4, as shown in Section 4.4).

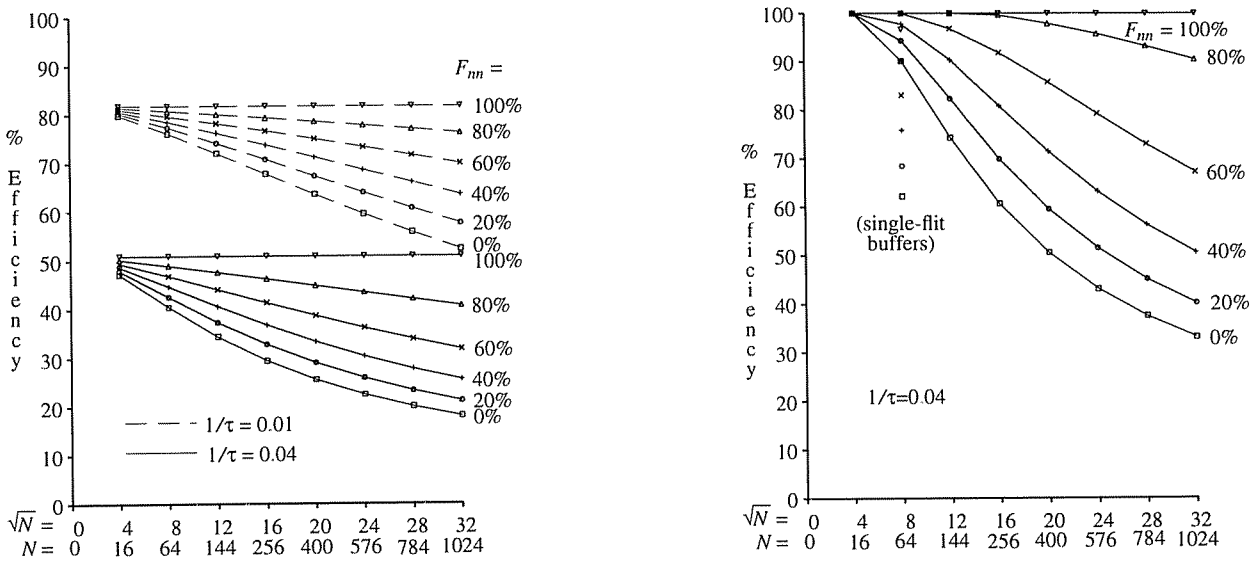
We next compare the bidirectional torus with the unidirectional torus. For the former, we use message lengths of $L_{msg\,1}$ = 4, $L_{resp\,1}$ = 10, $L_{msg\,2}$ = 12 and $L_{resp\,2}$ = 4, rather than 3, 9, 11 and 3 used in all other experiments. This allows us to halve these message lengths for the unidirectional torus, according to our assumption that its links are twice as wide as those in the bidirectional torus. Figure 4.5(b) plots processor efficiency as a function of $1/\tau$ for N_{out} = 1, 2 and 4, for each topology. The results are similar to the comparison against the bidirectional mesh. In particular, *the bidirectional torus performs significantly better than the unidirectional torus with multiple outstanding requests.* Thus, the extra number of hops in the unidirectional torus is not sufficiently offset by the wider channels. These results should hold approximately as system size increases, since the distance as well as the bandwidth scales at the same rate in both topologies.[7]

4.6. Nearest-Neighbor Workloads

The previous experiments assumed uniformly distributed inter-node communication, i.e., each node is equally likely to communicate with each other node. In this section, we investigate how near-neighbor communication locality affects system performance and our previous conclusions about system design trade-offs. We consider near-neighbor traffic patterns in which some fraction, F_{nn}, of the traffic generated by each processor is equally divided among its four nearest neighbors, while its remaining traffic is uniformly distributed to all nodes (including the four neighbors). The uniformly distributed traffic represents non-near-neighbor communication required by the near-neighbor application, as well as other activity on the system such as operating system traffic.

Figure 4.6(a) shows processor efficiency as a function of mesh radix, $\sqrt{N}$, for various values of F_{nn}, for N_{out} = 1. Curves are shown for two values of request rate, $1/\tau$ = 0.01 and 0.04. In both cases, increasing locality of communication improves processor efficiency only gradually. Figure 4.6(b) gives the results for N_{out} = 4 and $1/\tau$ = 0.4. (At $1/\tau$ = 0.01, the efficiency is close to 100% even with uniform traffic, as shown in Figure 4.4.) In this

7. Section 4.8, however, shows that performance imbalance between different parts of the system is higher with the unidirectional network, which may exacerbate the difference in performance at larger system sizes.



(a) $N_{out} = 1$.

(b) $N_{out} = 4$.

Figure 4.6. Effect of communication locality.

Bidirectional torus, infinite buffers except single points in (b).

case, efficiency improves substantially with increasing locality because locality reduces contention as well as latency. For example, the 1024-processor system with F_{nn} = 60% shows more than twice the efficiency of the same system under uniform traffic.

Locality of communication influences many of the design issues that were examined in previous sections assuming uniform traffic distribution. These must now be re-evaluated.

The set of points for single-flit buffers in Figure 4.6(b) shows mean processor efficiency for the 8×8 mesh with single-flit buffers and N_{out} = 4. We observe that, as for uniform traffic, the infinite buffer case is significantly better than the single-flit buffer case even up to F_{nn} = 70-80%.

The results for $1/\tau = 0.04$ in Figures 4.6 (a) and (b) show that the improvement when N_{out} goes from 1 to 4 is much stronger at higher levels of locality, for $N > 64$. Thus, *locality increases the benefit of multiple outstanding requests for large systems*. This is true because network contention is reduced for high values of F_{nn}, so that increasing N_{out} does not cause much higher contention, but does improve performance by overlapping communication with computation.

We can also re-evaluate how system performance scales when communication locality is present. Under uniform traffic we concluded that the mesh network scales well when N_{out} is 1, but scales poorly for N_{out} = 4. In Figure 4.6, we see that increasing locality has a positive effect on the scalability of the network (as expected), but nevertheless, for N_{out} = 4, *system performance only scales well for* $F_{nn} \geq 80\%$. It may be unrealistic to expect such

high levels of locality for real workloads.

Finally, the relative performance of the various mesh topologies may differ under near-neighbor workloads. In particular, we showed in Section 4.5 that the bidirectional torus has a significant performance advantage over the unidirectional torus with multiple outstanding requests. The performance advantage of the bidirectional torus will increase in the presence of locality since, in the unidirectional torus, a round-trip message to a near-neighbor requires $\sqrt{N}$ hops as compared with 2 hops.

To summarize the results of this section, locality of communication improves system performance particularly in large systems with multiple outstanding requests, and increases the benefit of multiple outstanding requests, but only marginally improves the ability of the mesh to support larger system sizes. In particular, the case of $N_{out} = 4$ does not scale well for $F_{nn}<80\%$. Other conclusions of the experiments with uniform workloads are also not altered for workloads with communication locality.

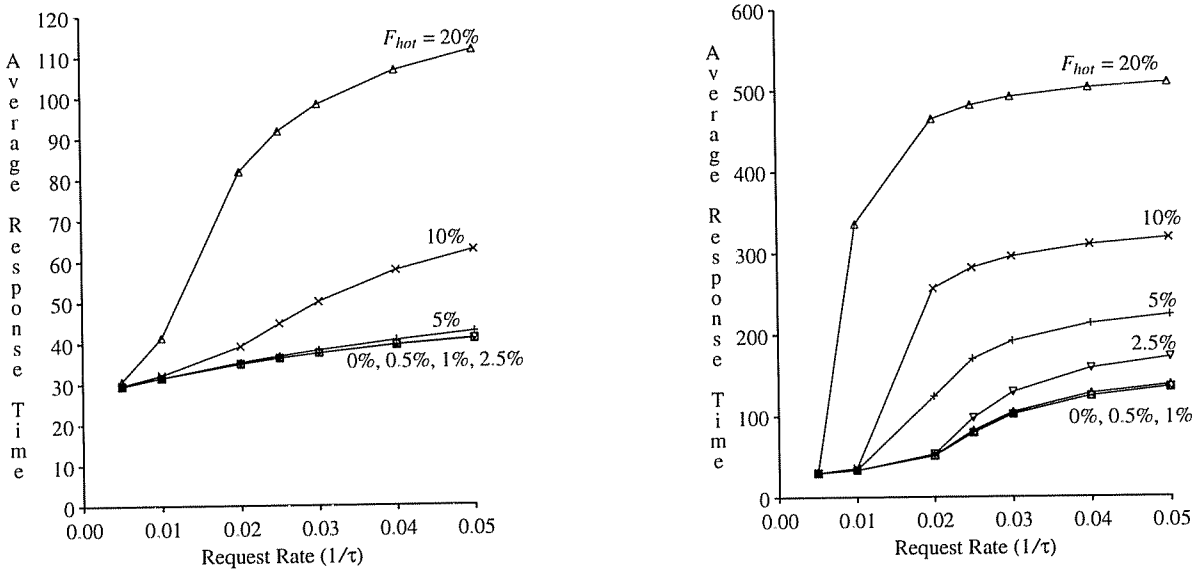
4.7. Hot-spot effects.

Hot-spots are a form of non-uniform communication that can strongly impact system performance. Hot-spots can arise, for example, when a number of processors make a significant fraction of their requests to a single memory module or to a single node in a multiprocessor. The issue has been studied using open queueing models (i.e., assuming non-blocking processors) in the context of multistage interconnection networks [13, 17, 23].

We examine the effect of hot-spots in mesh networks by assigning some fraction, F_{hot}, of requests from each processor to a particular node in the system, while the remaining fraction $1-F_{hot}$ is distributed uniformly across all processors. Figure 4.7 plots the mean response time (sum of average network plus remote-node residence times) versus request rate, $1/\tau$, for various values of F_{hot} in a bidirectional 8×8 torus with single-flit buffers. For $N_{out} = 1$ (Figure 4.7(a)), there is very little increase in mean round-trip time for $F_{hot} \leq 10\%$. For $N_{out} = 4$, however, much smaller fractions (about 2.5%) of hot traffic cause significant increases in mean round-trip time. (Note the larger range on the y-axis in Figure 4.7(b).) This result indicates that *the effect of a hot-spot is very sensitive to* N_{out}. In particular, this suggests that open traffic models ($N_{out} = \infty$) may yield extremely pessimistic results.

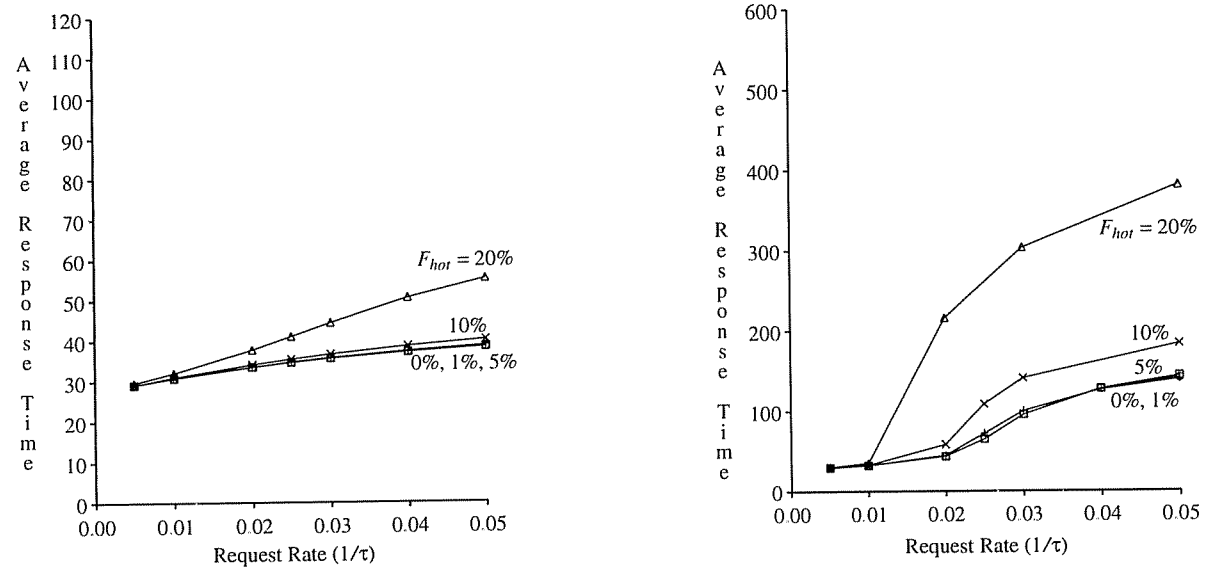
The above hot-spot experiments assumed a single node-to-switch link, as in all previous experiments. This link in the hot node is a bottleneck in the system, and substantial queues build up at the link because we have assumed unlimited buffer space for it. Hence, traffic in the rest of the network sees almost no contention. Using multiple links from node to switch (for example, one node-to-switch link per outgoing virtual channel) alleviates this bottleneck. The average response times for this case are shown in Figure 4.8.[8] The figure shows that the

8. The curves for $N_{out} = 4$ and $F_{hot} = 10\%$ and 20% in Figure 4.8(b) were plotted using data from simulations because the analytical model did not converge in this case. (The results of the analytical models for lower values of F_{hot} were in good agreement with simulations.)



(a) $N_{out} = 1$. (b) $N_{out} = 4$.

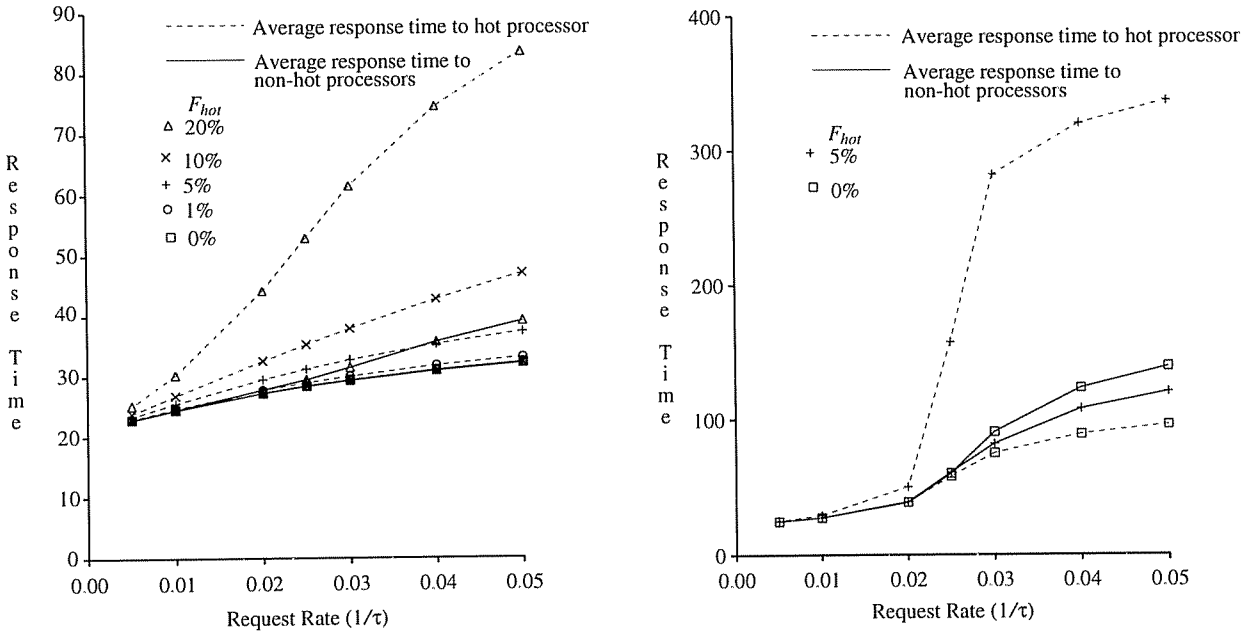
Figure 4.7. The effect of hot-spots on overall mean response times.
Bidirectional 8×8 torus, single-flit buffers.



(a) $N_{out} = 1$. (b) $N_{out} = 4$.

Figure 4.8. Overall mean response times under hot-spot traffic with multiple node-to-switch links.
Bidirectional 8×8 torus, single-flit buffers.

average round-trip time has reduced considerably for the cases that showed non-negligible increases in response time due to hot-spots in Figure 4.7. Although Figure 4.8 assumes one link per outgoing channel, we would expect to see approximately the same performance if the eight processor-switch channels were multiplexed onto 2-3



(a) $N_{out} = 1$.

(b) $N_{out} = 4$.

Figure 4.9. The effect of hot-spots on mean response times to *non-hot* processors.
Bidirectional 8×8 torus, single-flit buffers, multiple node-to-switch links.

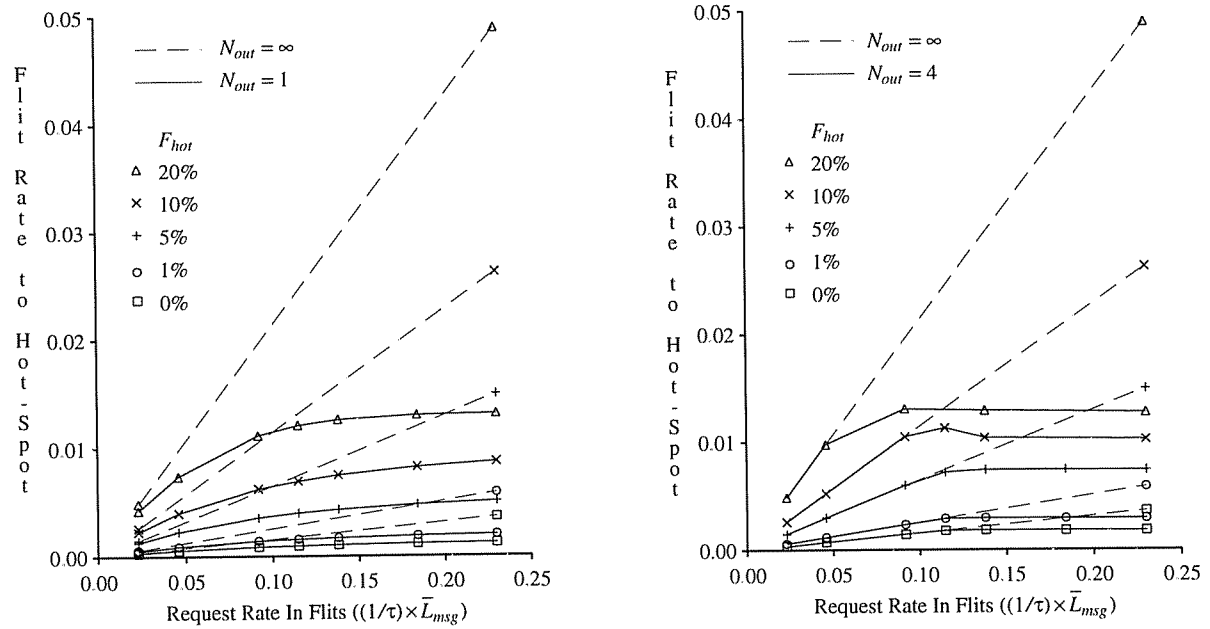


Figure 4.10. Effective flit rate to hot node with blocking and non-blocking processors.
Bidirectional 8×8 torus, single-flit buffers, multiple node-to-switch links.

physical links since a very high fraction of the outgoing traffic at each node uses only two or three of the eight outgoing virtual channels. (7/8 of the total traffic out a node must go out first on the column, and is further restricted to only two or three of these four outgoing virtual channels by the deadlock-avoidance algorithm.)

When the bottleneck on the node-to-switch link in the hot node is alleviated, the switch-to-node link becomes the new bottleneck in the system. Now, channel buffers on paths leading to this bottleneck link can get filled up, affecting messages to non-hot nodes as well. Hot-spot studies in indirect networks have shown that traffic to memory modules other than the hot module is slowed down as much as traffic to the hot module itself [17]. This phenomenon has been called *tree saturation.* To analyze the corresponding effect in mesh networks, we plot in Figure 4.9 the average response time for messages to the hot processor (dashed lines) and to all other processors (solid lines), assuming multiple node-to-switch links.[9] For $N_{out} = 1$, we see that traffic to non-hot processors does not see significant increase in response time even when F_{hot} is as high as 20%. For $N_{out} = 4$, mean response time to the non-hot processors actually decreases slightly when F_{hot} is increased from 0 – 5%. In this case, contention at the hot node has significantly decreased overall network throughput, offsetting any tree-saturation effect.

The above results suggest that the presence of hot-spots in mesh networks does not significantly increase response times for non-hot traffic, in systems of this size. This is different from the conclusions of Pfister and Norton [17] for *systems of the same size* based on multistage interconnection networks. The principal reason for the difference in results is that Pfister and Norton assumed an open model, in which processors generate requests continuously without blocking for responses to return (i.e., $N_{out} = \infty$). To illustrate the effect of this assumption, Figure 4.10 shows the actual request rate in flits per processor to the hot module for $N_{out} = \infty$ (dashed lines) as well as $N_{out} = 1$ and $N_{out} = 4$ (solid lines) as a function of the input rate (in flits), $(1/\tau) \times \bar{L}_{msg}$. The request rate to the hot module is significantly higher for $N_{out} = \infty$ than for finite values of N_{out}, and this would also be true for the multistage interconnection network.

The hot-spot experiments described in this section have focused on 64-processor systems. The degradation due to a hot-spot will become more severe with increasing system size. However, finite-buffer models are necessary for realistic hot-spot studies and we have been unable to use our analytical finite-buffer model to study large systems quantitatively. (Simulating these systems is even more difficult.) Nevertheless, we believe the results of this section provide insights that would be valuable in studying large systems. Qualitatively, we expect multiple node-to-switch links to significantly alleviate the effect of a hot-spot for larger systems as well. We also expect

9. Note that for $N_{out} = 4$, with uniform traffic ($F_{hot} = 0\%$) the round-trip time to the "hot" processor is actually lower than to other processors. This is a direct result of the asymmetric loads on the virtual channels described in Sections 2.4 and studied in Section 4.8. The hot processor chosen for these experiments (processor [2,2]) is located at a point in the mesh where loads on the outgoing virtual channels are balanced.

Table 4.4. Efficiencies of Individual Processors in the Mesh.

Ⓧ : Minimum Efficiency Value

[X] : Maximum Efficiency Value

(a) Unidirectional Torus (N_{out}=4, 1/τ=0.02)

	Col 0	1	2	3	4	5	6	7
Row 0	[91]	89	88	87	88	89	89	89
1	89	86	83	82	84	86	87	87
2	85	79	72	71	76	83	85	85
3	81	67	55	55	63	78	83	83
4	81	62	Ⓧ46	Ⓧ46	57	78	83	83
5	83	74	62	61	70	81	84	84
6	84	81	78	78	79	82	83	83
7	85	84	84	83	83	83	83	83

(b) Bidirectional Torus (N_{out}=4, 1/τ=0.04)

	Col 0	1	2	3	4	5	6	7
Row 0	Ⓧ48	52	57	60	60	59	55	49
1	52	55	60	62	63	61	58	53
2	59	61	64	66	66	65	63	60
3	64	65	67	[68]	[68]	67	66	64
4	64	65	67	[68]	[68]	67	66	65
5	60	62	64	65	65	65	63	60
6	53	57	61	62	62	61	58	54
7	49	53	58	60	60	58	55	49

that larger systems with blocking processors (finite N_{out}) will be able to support much higher levels of hot-spot traffic without introducing tree-saturation than would be predicted using open system models (i.e., assuming $N_{out} = \infty$).

4.8. Performance Imbalance Caused by the Deadlock-Free Routing Algorithm

The analyses of torus performance using the single-flit buffer analytical model show significant differences among processor efficiencies at different locations in the system, and these observations are corroborated by simulation (see Section 4.1). To illustrate the imbalance, Table 4.4(a) gives the efficiencies of the individual processors in the unidirectional 8×8 torus, (N_{out} = 4, 1/τ = 0.02) under *uniform* traffic, i.e., with equal loads on all physical links. The table shows that the processors near the corners have high efficiencies, while the ones near the center of the mesh have much lower efficiencies. Table 4.4(b) shows the processor efficiencies for the bidirectional 8×8 torus (N_{out} = 4, 1/τ = 0.04) under uniform traffic. Again imbalance is observed; however, this time the processors near the corners have low efficiency. Note that these two cases represent operating points with moderate to high *average* processor efficiency (79% and 58% respectively), yet also with significant performance imbalance across the system.

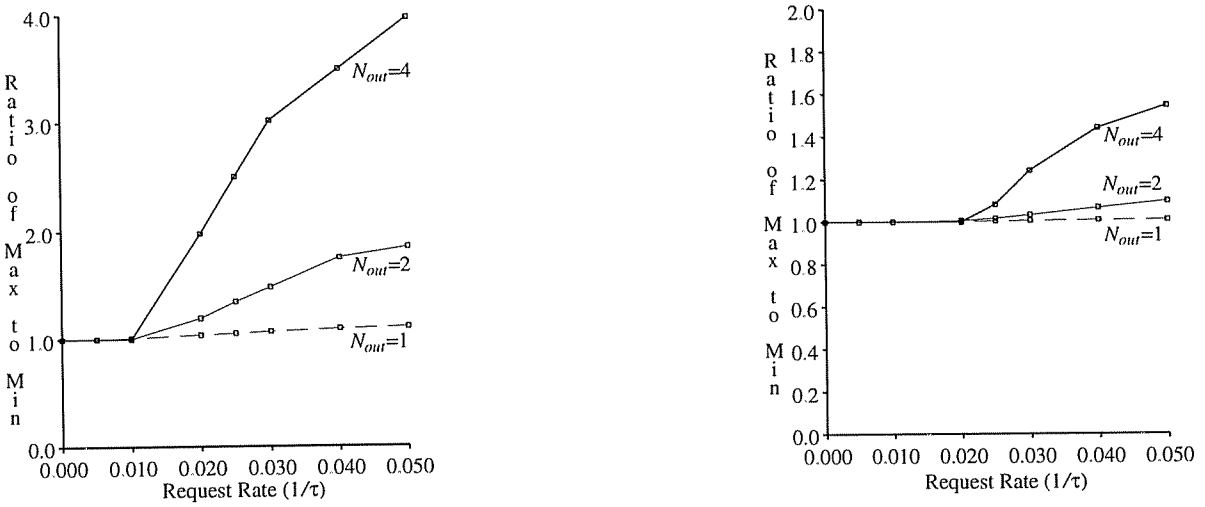
To quantify the performance imbalance at particular parameter settings, we use the ratio of the maximum processor efficiency to the minimum processor efficiency. Figure 4.11 plots this ratio as a function of the request rate for the unidirectional and bidirectional 8×8 tori, for N_{out} = 1, 2, and 4. The figure shows that the imbalance becomes significant when network contention is moderately high, but includes cases that represent reasonable operating points (i.e., average efficiencies greater than 50%). For example, the 64-processor system with N_{out} = 4 has average efficiency greater than 50% at most request rates, as shown in Figure 4.5(b); however, the imbalance is as high as 1.5 for the bidirectional torus and 4.0 for the unidirectional torus. Finally, comparing Figures 4.11 (a) and (b), we also see that the imbalance is much greater in the unidirectional torus than in the bidirectional torus.

The results described above suggest that the imbalance in system performance can be significant and needs to be considered during the design of the system. The imbalance in processor performance may have significant implications, for example, for parallel programs that synchronize via barriers. Whether the imbalance is significant for any particular system depends on several factors, including buffer size, message lengths and request rate.

Recent studies have shown that mesh networks *without end-around connections* also have significant, symmetric imbalances in processor performance even under uniform communication [3, 7]. However, these imbalances occur because of unequal traffic requirements on the physical links, which arises from edge effects due to the lack of end-around connections. In torus networks, physical link loads are balanced under uniform traffic; thus the source of the observed imbalance in processor performance must be sought elsewhere.

A potential source of imbalance in torus networks is the asymmetric virtual channel loading by the deadlock avoidance algorithm, described in Section 2.4. In attempting to determine whether and how this asymmetry causes the imbalance, an obvious guess is that the round-trip communication by some nodes makes greater use of high-load virtual channels (i.e., channels that carry a high fraction of their links' traffic). However, this explanation cannot account for an observed peculiarity in the pattern of imbalance in the unidirectional case, namely, that *processors with poor performance are not necessarily located where greatest asymmetry in channel loading occurs*. In fact, channels near the edges have the greatest traffic asymmetry (Figure 2.4) but processors near the edges have the best performance (Table 4.4(a)). To understand why the above explanation is invalid in general, note that for outgoing requests that use high-load channels, the responses will use low-load channels on the return trip (and vice-versa) due to symmetries in the routing algorithm. (This is easiest to reason about for the unidirectional torus.) Some careful thought reveals that the differences between nodes that place somewhat greater load on balanced virtual channels versus nodes that place somewhat greater load on high- and low-load channels are not likely to account for the fairly large observed imbalance in processor efficiencies.

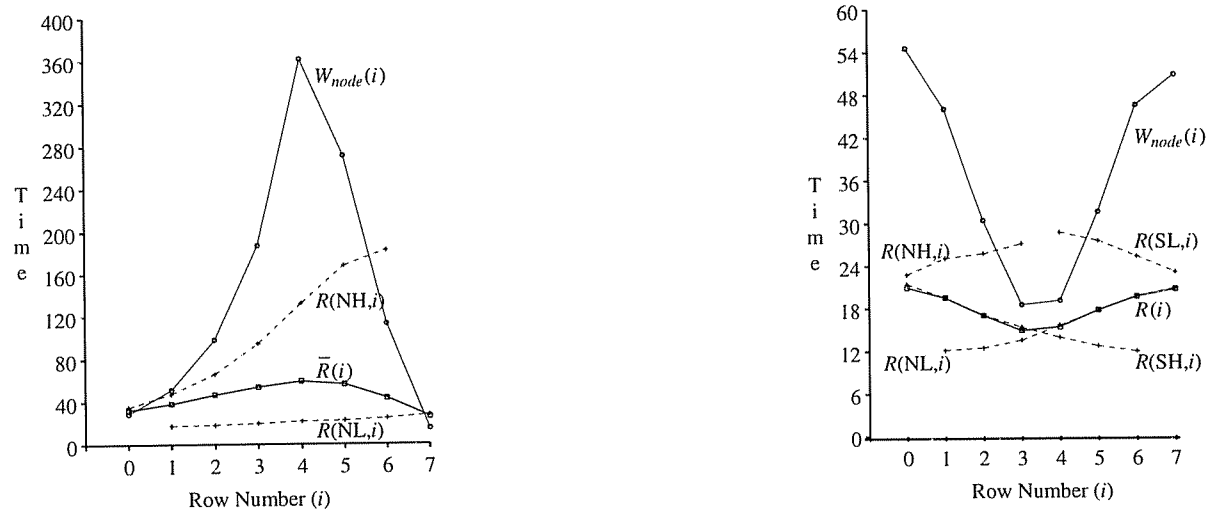
More careful consideration of message *pipelining and blocking* behavior reveals a different and potentially substantial impact of the asymmetric loads that can also explain the particular patterns of imbalance observed in Table 4.4. Specifically, certain nodes' outgoing messages (both requests and responses *to* other nodes) experience relatively severe blocking due to high-load virtual channels, after leaving the node. Such nodes will see significantly higher contention for their *node-to-switch link* because of the pipelining and blocking of messages. Since each processor is the heaviest user of its own node-to-switch link, increased contention on this link results in lower efficiency for the processor at such a node. Furthermore, in the unidirectional torus, it is the nodes near the center of the network whose outgoing messages experience most severe blocking due to high-load channels, whereas in the bidirectional torus it is the nodes near the edges. This leads to the different patterns of imbalance in the two cases.



(a) Unidirectional 8×8 torus

(b) Bidirectional 8×8 torus

Figure 4.11. Maximum performance imbalance.
Single-flit buffers, uniform traffic



(a) Unidirectional 8×8 torus

(b) Bidirectional 8×8 torus

Figure 4.12. Residence time on first virtual channel.
Nodes on column 3; single-flit buffers, uniform traffic.

To demonstrate the phenomenon quantitatively, consider the nodes on some fixed column (as in Figure 2.4) of a mesh network. For the node on row i, define

$R(c,i) \equiv$ mean residence time on outgoing virtual channel c for a message out of the node (i.e., for a message that was transferred over the node-to-switch link into the buffer for channel c),

$\bar{R}(i)$ ≡ average of $R(c,i)$, averaged over all outgoing channels c,

$W_{node}(i)$ ≡ mean waiting time for the node-to-switch link at the node.

Figure 4.12 (a) plots $R(c,i)$, for c = NH and NL, $\bar{R}(i)$, and $W_{node}(i)$ as functions of i (row number) for column 3 of a unidirectional 8×8 torus (NH and NL denote the High and Low channels in the North direction). $\bar{R}(i)$ is higher near the middle of the column than near the edges, showing that outgoing messages from the nodes near the middle experience much more severe blocking, as described above. Now, high $\bar{R}(i)$ also implies a high residence time on the node-to-switch link, and hence a high waiting time, $W_{node}(i)$, just as shown in the figure. This leads to poorer performance for these nodes. For the bidirectional case, Figure 4.12 (b) plots $R(c,i)$ for $c \in \{NH,NL,SH,SL\}$, $\bar{R}(i)$, and $W_{node}(i)$. In this case, $\bar{R}(i)$ and hence $W_{node}(i)$ are higher near the edges of the torus, and thus the efficiency is lower.

Thus, the ultimate effect of high-load channels is the same in both networks: they cause more severe blocking for outgoing messages of some nodes, which produces much greater contention for the node-to-switch link at these nodes. However, the pattern of use of the high-load channels by outgoing messages is different in the two networks. In the bidirectional case, blocking of outgoing messages is more severe for nodes near the edges because these nodes' outgoing messages make much greater use of high-load virtual channels compared to nodes near the center. For example, a node on row 7 in an 8×8 bidirectional network sends all its outgoing messages on channels that carry 100% of their links' traffic (NL and SL), whereas the outgoing messages for a node on row 3 are mostly concentrated on channels with low to moderate load (NL and SH). In the unidirectional case, however, the nodes near the center, which have poor performance, make only slightly greater use of high-load virtual channels than nodes near the edges. In these networks, it appears more significant that outgoing messages from nodes near the center travel from channels with a low to moderate load into channels with a high load, whereas for nodes near the edges the opposite is true (refer to Figure 2.4). The pipelined routing causes significantly greater blocking for the former than for the latter.

As the preceding discussion indicates, the precise explanation of the relationship between the asymmetric channel loadings and the performance imbalance is fairly subtle and non-intuitive. Formulating and validating the explanation required significant insight as well as analysis detailed metrics obtained from the analytical model. Furthermore, the imbalance cannot be detected or analyzed using models or simulations that ignore the virtual channel loadings or the finite switch buffers. Finally, note that the waiting times for the node-to-switch links, and hence the imbalance itself, might be reduced by the use of multiple physical node-to-switch links and/or multiple virtual node-to-switch channels.

5. Conclusions.

We have developed accurate, approximate Mean Value Analysis models for k-ary n-cube interconnection networks with wormhole routing, with single-flit and infinite buffers at the switches. Interesting aspects of the model include the techniques used to estimate mean message blocking times, mean message queueing times at the processors, and the mean queue lengths seen at the first outgoing link when multiple channels connect each processor to its switch output channels. Many of the experimental results would not have been possible with simpler analytical models that do not represent the message blocking and details of the routing. The equations for channel waiting time, which form the foundation of the single-flit-buffer model, use recurrence relations to model the dependencies among flit blocking times within a single message, yet use random arrival instant assumptions to model interference by other messages. The models were shown to be quite accurate by extensive validations with simulation. We are not aware of any previous work that has used similar models of blocking in these networks. We believe the validation results are important evidence that approximate mean value analysis is a viable technique for modeling complex systems.

We used the models to analyze various issues that arise in the design of 2-dimensional (mesh) networks. These results (summarized below) should prove useful for engineering high-performance systems based on low-dimensional k-ary n-cube networks.

Some of our results confirm and quantify existing intuition about mesh interconnection networks. With processors that block on every request we have shown that contention in the network is low, and the three network topologies (bidirectional and unidirectional torus and bidirectional mesh) show little difference in performance. Multiple outstanding requests can help increase performance, but also cause increased contention. Thus, in this case, substantial performance gain is achievable by increasing buffer size.

We also gained new intuition from some of our results, including:

- Under uniform workloads absolute performance is higher with multiple outstanding requests, but network performance does not scale well with increasing system size.
- Communication locality improves system performance, particularly for multiple outstanding requests, but at least 70-80% of each processor's traffic must be directed to its nearest neighbors before the case of 4 outstanding requests scales well.
- With multiple outstanding requests, the bidirectional torus performs significantly better than the other two topologies. Furthermore, it exhibits much lower performance imbalance (see below) due to the deadlock-free routing algorithm than the unidirectional torus.
- Open system models can yield extremely pessimistic results in hot-spot studies. When processors block after making a few requests, only high fractions of hot-spot traffic cause significant performance degradation in 64-processor systems with single-flit buffers. Furthermore, traffic to the non-hot processors is not much affected by hot-spot traffic in these systems (i.e., tree-saturation is not observed).
- At some plausible operating points (i.e., in cases where average processor efficiency is reasonably high), there is perceptible difference in the efficiencies of processors at different locations in the mesh. This imbalance is due to asymmetric loads on the virtual channels by the deadlock-avoidance algorithm.

A number of related issues for k-ary n-cube networks remain to be studied. The models developed in this paper can be used to study network performance for message-passing and hierarchical systems. The conclusions from the experiments need to be examined for 3-dimensional networks. The result that a communication hot-spot does not significantly slow down other traffic in the system needs to be re-examined for larger systems. A related question that needs to be answered is what buffer sizes are required to approximate infinite buffer performance, under various workload assumptions. However, for interconnection networks with pipelined routing in particular, modeling the performance with larger finite buffers is a difficult problem. (Previous analytical models of interconnection networks that allow finite buffer sizes are based on a decomposition approximation in which each queue is analyzed in isolation, thus *ignoring the dependencies between network stages* caused by the blocking and pipelining of messages; for example, see [15] and the references therein.) Finally, it would be worthwhile to develop a deadlock-free routing algorithm for the mesh network that does not lead to the imbalance in processor efficiencies that we have observed.

6. Acknowledgements.

We thank Amarnath Mukherjee and Derek Eager for valuable discussions during the development of the model. We also thank Sarita Adve, Anant Agarwal, Derek Eager, Mark Hill, Susan Owicki, Gurinder Sohi and an anonymous referee for valuable comments on earlier drafts of this paper.

References

[1] A. Agarwal, B. Lim, D. Kranz and J. Kubiatowicz, APRIL: A Processor Architecture for Multiprocessing, *17th Annual International Symposium on Computer Architecture*, May 1990, 104-114.

[2] A. Agarwal, Limits on Interconnection Network Performance, *IEEE Trans. on Parallel and Distributed Systems 2*, 4 (October 1991), 398-412.

[3] K. Bolding and L. Snyder, Mesh and torus chaotic routing, *Advanced Research in VLSI and Parallel Systems: Proceedings of the Brown/MIT Conference*, March 1992, 333-347.

[4] K. Bolding, Non-Uniformities Introduced by Virtual Channel Deadlock Prevention, Technical Report 92-07-07, Department of Computer Science and Engineering, University of Washington , July 1992.

[5] S. Borkar, iWarp: An Integrated Solution to High-Speed Parallel Computation, *Proceedings of Supercomputing '88*, November 1988.

[6] K. M. Chandy and D. Neuse, Linearizer: A Heuristic Algorithm for Queueing Network Models of Computer Systems, *Communications of the ACM 25*(1982), 126-134.

[7] S. Chittor and R. Enbody, Performance Degradation in Large Wormhole-Routed Interprocessor Communication Networks, *Proc. 1990 International Conference on Parallel Processing*, 1990, I-424 - I-428.

[8] W. J. Dally, *A VLSI Architecture for Concurrent Data Structures*, Ph.D. Thesis, California Institute of Technology, March 1986.

[9] W. J. Dally and C. L. Seitz, Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, *IEEE Trans. on Computers C-36*, 5 (May 1987), 547-553.

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

[11] E. Gelenbe, Performance Analysis of the Connection Machine, *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems 18*, 1 (May 1990), 183-191.

[12] P. Kermani and L. Kleinrock, Virtual Cut-through: A New Computer Communication Switching Technique, *Computer Networks 3*(October 1979), 267-286.

[13] G. Lee, C. P. Kruskal and D. J. Kuck, The Effectiveness of Combining in Shared-Memory Parallel Computers in the Presence of 'Hotspots', *Proc. International Conference on Parallel Processing*, 1986, 35-41.

[14] D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz and M. Lam, The Stanford DASH Multiprocessor, *IEEE Computer 25*, 3 (March 1992), 63-79.

[15] T. Lin and L. Kleinrock, Performance Analysis of Finite-Buffered Multistage Interconnection Networks with a General Traffic Pattern, *Proc. 1991 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, San Diego, California, USA, May 21-24, 1991.

[16] M. D. Noakes, D. A. Wallach and W. J. Dally, The J-Machine Multicomputer: An Architectural Evaluation, *20th Annual International Symposium on Computer Architecture*, May 1993, 224-235.

[17] G. F. Pfister and V. A. Norton, 'Hot Spot' Contention and Combining in Multistage Interconnection Networks, *IEEE Trans. on Computers C-34*, 10 (October 1985), .

[18] P. Schweitzer, Approximate Analysis of Multiclass Closed Networks of Queues, *International Conference on Stochastic Control and Optimization*, 1979.

[19] H. S. Stone, *High Performance Computer Architecture*, Addison-Wesley Publishing Company, 1987.

[20] M. K. Vernon, E. D. Lazowska and J. Zahorjan, An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols, *Proc. 15th International Symposium on Computer Architecture*, June 1988.

[21] W. Weber and A. Gupta, Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results, *The 16th. Annual International Symposium on Computer Architecture*, May 1989, 273-280.

[22] D. L. Willick and D. L. Eager, An Analytic Model of Multistage Interconnection Networks, *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1990, 192-202.

[23] P. Yew, N. Tzeng and D. H. Lawrie, Distributing Hot-Spot Addressing in Large-Scale Multiprocessors, *IEEE Trans. on Computers C-36*, 4 (April 1987), .

Appendix A. Notation used in the Model.

We use the following convention for integer subscripts in the model equations: i, s, d, q denote node numbers. (In this usage, i always appears within brackets, i.e., "$[i]$" and "(i,n)"). j denotes message type. k denotes flit number. c denotes a virtual channel. i, I denote input ports. (In this case, i always appears as a subscript, e.g., $r_{j,c,i}$).

Term	Definition
	Terms common to the entire model
$R[i]$	Mean round-trip time for customer of class i.
$r_{network}[i]$, $r_{proc}[i]$, $r_{remote}[i]$	Mean residence time for a customer of class i in the network, at the processor, and at the remote node respectively.
$r_{j,sd}$	Mean residence time in the network for a message of type j from s to d.
$(c \pm k)_{sd}$	The virtual channel which is k steps after (or before) virtual channel c on the path from to s to d.
$\overline{c}$	The virtual channel that shares the same physical link as c.
	Queueing for the channels
$b_{node,j,s\,\|q}[k]$, $q_{node,j,s\,\|q}$	For a message class q arriving to the link from processor to switch at node s: respectively, the probability that the link is busy serving the k^{th} flit of a request of type j, and the mean number of waiting requests of type j.
$r_{node,j,s}[k]$	Mean residence time for the k^{th} flit of a request of type j, on the link from processor to switch at node s.
$D_{c,i\,\|j}(s)$	Set $\{d \mid$ messages from s to d, or responses from d to s if j is a reply message, visit c via input port $i\}$.
$r_{j,c,I}[k]$, $u_{j,c,I}[k]$	Respectively, the mean residence time of the k^{th} flit of messages of type j on channel c, which arrive to c via input port I, and the mean utilization of channel c by such flits.
$r_{j,c,sd}[k]$	Mean residence time on channel c of the k^{th} flit of a message of type j from s to d.
$u_{link,\overline{c}}$	Utilization of the physical link corresponding to channel c by messages on companion channel $\overline{c}$.
$w_{node,s\,\|q}$	Mean waiting time for the link from processor to switch at node s, for the header flit of a message of class q.
$w_{c\,\|I}$	Mean waiting time for channel c, for a message arriving to c via input port I.
	Queueing for the processors
$q_{proc}(i,n)$, $u_{proc}(i,n)$, $r_{proc}(i,n)$	Steady state mean queue length, residence time, and utilization of processor i when there are n customers in class i.
res_{proc}	Mean residual service time of a customer found in service by a message arriving to a processor, as seen by the tail flit of the message, conditioned on the header flit finding the processor busy.
	Queueing at a remote node
$B_{mem,j,s\,\|d}$, $Q_{mem,j,s\,\|d}$, $W_{mem,j,s\,\|d}$	For an arriving request from s at remote node d: respectively, the probability d is busy serving a request of type j', the mean number of waiting requests of type j', and the mean waiting time.
$res_{mem,j'\,\|j}$	Mean residual service time of a request of type j at a remote node as seen by the tail flit of a type j' request, conditioned on the header flit finding the type j request in service.
	Queueing for the first network virtual channel on a path, with multiple processor → switch channels.
$b'_{j,c,i\,\|q}[k]$, $q'_{j,c,i\,\|q}$	For a customer of class q arriving to channel c via input port i: respectively, the probability c is serving the k^{th} flit of a request of type j, and the mean number of waiting requests of type j.
$V_{j,c,I}$	Total visit ratio of all customer classes to channel c as type j messages arriving via input port i.
$w'_{c\,\|q}$	Mean waiting time for channel c by a customer of class q, where c is the first network virtual channel on its path.

Appendix B. The Model.

The equations of the model are given in detail here. Throughout the development of the model, we call a message from s to d or a response from d to s a message of *class s*. Also, in a summation over all four types j, we write $\sum_j$ instead of $\sum_{j \in \{msg1,msg2,resp1,resp2\}}$.

$$R[i] = r_{proc}[i] + r_{network}[i] + r_{remote}[i], \quad i=1 \cdots N. \tag{1}$$

Queueing in the network.

$$r_{network}[s] = \sum_{d \neq s} F_{sd}(P_1(r_{msg\,1,sd}+r_{resp\,1,ds}) + P_2(r_{msg\,2,sd}+r_{resp\,2,ds})), \quad i=1 \cdots N. \tag{2}$$

$$r_{j,sd} = w_{node,sd\,|\,j} + \sum_c r_{j,c,sd}[1] + T_{catchup,j,sd}, \qquad j \in \{msg1,msg2,resp1,resp2\}, \tag{3}$$

where the sum is over all channels c on the path from s to d.

$$r_{j,c,sd}[k] = \begin{cases} w_{(c+1)_{sd}\,|\,I} + u_{link,\bar{c}} + 1 & k = 1 \\ r_{(c+k-1)_{sd}}[1] + u_{link,\bar{c}} & k > 1, d \text{ is } k \text{ or more steps away from } c. \\ 1+u_{link,\bar{c}} & \text{otherwise} \end{cases} \tag{4,}$$

$T_{catchup}$ is the length of the message less 1 (for the header) plus the delay due to sharing the physical links with traffic on other virtual channels, summed over the links the tail flit must traverse *after the header flit reaches the destination node*:

$$T_{catchup,j,sd} = L_j - 1 + \sum_{\text{c: c is} < L_j \text{hops from d}} u_{link,\bar{c}}.$$

The equation for the waiting time, $w_{c\,|\,I}$, has been explained in Section 3.2.

$$w_{c\,|\,I} = \sum_{i \neq I} \sum_j \sum_{k=1}^{L_j} u_{j,c,i}[k] \left[\frac{1}{2} r_{j,c,i}[k] + \sum_{l=k+1}^{L_j} r_{j,c,i}[l] \right]$$
$$+ \sum_j \left[\frac{u_{j,c,I}(L_j)}{1 - \sum_{j'} \sum_{l=1}^{L_{j'}-1} u_{j',c,I}[l]} \times \frac{r_{j,c,I}[L_j]}{2} \right] + \sum_{i \neq I} \sum_j \left[u_{j,(c-1)_i}[1] \sum_{l=1}^{L_j} r_{j,c,i}[l] \right] \tag{6}$$

$r_{j,c,i}[k]$, $u_{j,c,i}[k]$, and $u_{link,c}$ remain to be calculated. These can be expressed in terms of $r_{j,c,sd}[k]$ in a straightforward manner. First, define $D_{c,i\,|\,j}(s) \equiv \{ d \mid$ messages from s to d visit c via input port $i \}$. Then,

$$r_{j,c,I}[k] = \frac{\sum_{s=1}^{N} \sum_{d \in D_{c,i\,|\,j}(s)} F_{sd} P_j r_{j,c,sd}[k]}{\sum_{s=1}^{N} \sum_{d \in D_{c,i\,|\,j}(s)} F_{sd} P_j}, \tag{7}$$

$$u_{j,c,I}[k] = \sum_{s=1}^{N} \sum_{d \in D_{c,i\,|\,j}(s)} \frac{N_{out} F_{sd} P_j r_{j,c,sd}[k]}{R[s]}. \tag{8}$$

$$u_{link,c} = \sum_{j} \sum_{s=1}^{N} \sum_{d \in D_{c|j}(s)} \frac{N_{out} F_{sd} P_j L_j}{R[s]}, \quad \text{where } D_{c|j}(s) \equiv \bigcup_{i} D_{c,i|j}(s). \tag{9}$$

Waiting time for the processor link

The remaining term in Equation (3), $w_{node,sd|j}$, is calculated next. For request messages, $w_{node,sd|j}$ is the average waiting time for the P-link at node s seen by a class s message ($w_{node,s|s}$), and for reply messages $w_{node,sd|j}$ is the average waiting time for the P-link at node s seen by a class d message ($w_{node,s|d}$). In general, $w_{node,s|q}$ is calculated in a manner very similar to (6) for $w_{c|l}$.

$$w_{node,s|q} = \sum_{j} \left\{ \sum_{k=1}^{L_j} \left\{ b_{node,j,s|q}[k] \left(\frac{r_{node,j,s}[k]}{2} + \sum_{l=k+1}^{L_j} r_{node,j,s}[l] \right) \right\} + q_{node,j,s|q} \sum_{l=1}^{L_j} r_{node,j,s}[l] \right\}. \tag{10}$$

The main differences from equation 6 for $w_{c|l}$ are:

i. the second term in equation 6 is no longer required and the first term does not have a summation over input ports. In both cases the reason is that the buffer capacity for this link is unbounded.

ii. The third term in equation 6 used the probability that there is a waiting request, for each input port i. Now, however, we require an actual queue length, $q_{node,j,s|q}$, denoting the average number of requests of type j waiting for the processor link in node s when a request of class q arrives. This can be calculated as follows:

$$q_{node,j,s|q} = \begin{cases} \sum_{d \neq s} \dfrac{(N_{out} - \mathbf{1}_{\{s=q\}}) F_{sd} P_j w_{node,s|s}}{R[s]}, & j \in \{msg1, msg2\} \\ \sum_{s' \neq s} \dfrac{(N_{out} - \mathbf{1}_{\{s=q\}}) F_{s'q} P_j w_{node,s'|q}}{R[s']}, & j \in \{resp1, resp2\} \end{cases} \tag{11}$$

where $\mathbf{1}_{\{s=q\}}$ is 1 if $s=q$, and 0 otherwise. We can calculate $r_{node,j,s}[k]$ and $b_{node,j,s|q}[k]$ similarly.

$$b_{node,j,s|q}[k] = \begin{cases} \sum_{d \neq s} \dfrac{(N_{out} - \mathbf{1}_{\{s=q\}}) F_{sd} P_j}{R[s]}, & j \in \{msg1, msg2\} \\ \sum_{s' \neq s} \dfrac{(N_{out} - \mathbf{1}_{\{s'=q\}}) F_{s'q} P_j}{R[} s'], & j \in \{resp1, resp2\} \end{cases} \tag{12}$$

$$r_{node,j,s}[k] = \begin{cases} \dfrac{\sum_{d=1}^{N} F_{sd} P_j r_{j,node,sd}[k]}{\sum_{d=1}^{N} F_{sd} P_j}, & j \in \{msg1, msg2\} \\ \dfrac{\sum_{s'=1}^{N} F_{s's} P_j r_{j,node,s'p}[k]}{\sum_{s'=1}^{N} F_{s'p} P_j}, & j \in \{resp1, resp2\} \end{cases} \tag{13}$$

Queueing at the processor.

As defined in Appendix A, $r_{proc}(i,n)$ is the average residence time at the processor for a customer of class i, when there are n customers in its class. Hence $r_{proc}[i]=r_{proc}(i,N_{out})$ by definition. $r_{proc}(i,n)$ is calculated by recursion on n.

$$r_{proc}(i,n) = [q_{proc}(i,n-1) - u_{proc}(i,n-1)] \times \tau + u_{proc}(i,n-1) \times res_{proc}, \quad n > 1, \tag{14}$$

$$q_{proc}(i,n) = \frac{n \times r_{proc}(i,n)}{r_{proc}(i,n) + r_{network}[i] + r_{remote}[i]}, \tag{15}$$

$$u_{proc}(i,n) = \frac{n \times \tau}{r_{proc}(i,n) + r_{network}[i] + r_{remote}[i]}, \text{ and} \tag{16}$$

$$q_{proc}(i,1) = u_{proc}(i,1) = \frac{\tau}{\tau + r_{network}[i] + r_{remote}[i]}, \tag{17}$$

$$res_{proc} = (\tau-1) \times (P_1 \gamma^{L_{resp1}-1} + P_2 \gamma^{L_{resp2}-1}), \quad \gamma = 1 - (1/\tau). \tag{18}$$

Residence time at the remote node.

The equations for the residence time at the remote node are developed assuming that the request is serviced at the memory of the remote node without interrupting the remote processor (we denote r_{remote} as r_{mem} here). For example, in a shared-memory system remote memory accesses could be of this type, with *msg* 1, *msg* 2, *resp* 1 and *resp* 2 corresponding to *read*, *write*, *data* and *acknowledgement* messages respectively. The time to access one word is $D_{mem,1}$ for a read request and $D_{mem,2}$ for a write request. We assume that a request is queued for a memory module only when its last flit is received at the node. We also assume that the memory at each node is interleaved and, to simplify the analysis, all accesses read or write the first byte from the first module, the second byte from the second module and so on. This implies that $D_{mem,1}$ ($D_{mem,2}$) cycles after a read (write) request begins service, the next memory request can begin service. Further, for a read request the response (data message) is queued up to be transmitted as soon as the first word of data is read out from the first memory module, with subsequent words being transmitted one per cycle. An acknowledgement in response to a write request is queued when the last byte has been written, i.e. after L_{msg2} cycles. We do not limit the number of requests that can simultaneously be queued up for a given module.

The method for calculating the mean residence time at memory is the same as in [22].

$$r_{mem}[s] = \sum_{d} \sum_{j \in \{1,2\}} F_{sd} P_j (D_{mem,j} + w_{mem,j,s|d}), \tag{19}$$

$$w_{mem,j,s|d} = \sum_{j' \in \{1,2\}} \left\{ q_{mem,j',s|d} D_{mem,j'} + b_{mem,j',s|d} res_{mem,j'|j} \right\} \tag{20}$$

$$q_{mem,j',s'|d} = \sum_{s \neq d} (N_{out} - \mathbf{1}_{\{s=s'\}}) F_{sd} P_{j'} \frac{w_{mem,j',s|d}}{R[s]}. \tag{21}$$

$$b_{mem,j',s'|d} = \sum_{s \neq d} (N_{out} - \mathbf{1}_{\{s=s'\}}) F_{sd} P_{j'} \frac{D_{mem,j'}}{R[s]}. \tag{22}$$

Finally, just as in the processor queueing equations, the residual life of a memory request in service has to be

calculated as seen by the tail flit rather than as seen by the head. Defining $res_{mem,j'|j}$ to be the residual service time of a type $j' \in \{1,2\}$ request as seen by the tail flit of a message of type $j \in \{msg1, msg2\}$, we have:

$$res_{mem,j'|j} = (D_{mem,j'} - L_j + 1) \times (D_{mem,j'} - L_j) \,/\, (2 \times D_{mem,j'}). \tag{23}$$

Waiting time for the first virtual channel in the network, with multiple processor-switch channels.

The equations of the model described so far have been the same for single or multiple processor → switch channels. The only exception is (10) for $w_{node,s|q}$, which now would be denoted by $w_{node,c|q}$, and must be calculated separately for each outgoing channel c from node s. Similarly, $q_{node,j,c|q}$, $b_{node,j,c|q}[k]$, and $r_{node,j,c}[k]$ have to be calculated separately for each c; however, in all cases the equations remain essentially the same.

The waiting time for the buffer in the first switch is the only part that needs to be calculated somewhat differently, as described in Section 3.5. The equations are as follows.

$$\begin{aligned} w'_{c|q} = \sum_j \Bigg\{ & \sum_{i \neq PROC} \sum_{k=1}^{L_j} b'_{j,c,i|q}[k] \left[\frac{r_{j,c,i}[k]}{2} + \sum_{l=k+1}^{L_j} r_{j,c,i}[l] \right] \\ & + \left[\sum_{k=1}^{L_j} b_{node,j,c|q}[k] \right] (r_{j,c|PROC}[L_j] - 1) \\ & + \sum_{i \neq PROC} \left[q'_{j,c,i|q} \sum_{k=1}^{L_j} r_{j,c,i}[k] \right] \Bigg\} \end{aligned} \tag{24}$$

The first term in Equation (24) has been explained in Section 3.5, and the remaining two terms are similar to the second and third terms in Equation (6). Then, as explained in Section 3.5, $b'_{j,c,i|q}[k]$ is calculated as

$$b'_{j,c,i|q}[k] = \left\{ \frac{1 - \sum_j \sum_{k=1}^{L_j} b_{node,j,c|q}[k]}{1 - \sum_j \sum_{k=1}^{L_j} u_{j,c,PROC}[k]} \right\} \times u_{j,c,i}[k] \tag{25}$$

Finally, the equation for $q'_{j,c,i|q}$ is:

$$\begin{aligned} q'_{j,c,i|q} = & \left\{ \sum_j \sum_{k=1}^{L_j} b_{node,j,c|q}[k] \right\} \times \frac{V_{j,c,i}}{\sum_{j'} V_{j',c,PROC}} \left\{ \sum_j \sum_{k=1}^{L_j} U_{j,c,PROC}[k] + \sum_j u_{j,(c-1)_{PROC}}[1] \right\} + \\ & \sum_{\substack{i' \neq PROC \\ i' \neq i}} \left\{ \sum_j \sum_{k=1}^{L_j} b'_{j,c,i|q}[k] \right\} \times \frac{V_{j,c,i}}{\sum_{\substack{i' \neq PROC \\ i' \neq i}} \sum_{j'} V_{j',c|i'}} \times \sum_{\substack{i' \neq PROC \\ i' \neq i}} \sum_j \sum_{k=1}^{L_j} u_{j,c,i'}[k] \,. \end{aligned} \tag{26}$$

The first line of (26) corresponds to waiting for messages that were blocked on input port $i \neq PROC$ by the preceding message on the processor-to-switch link, when the processor-to-switch link is found busy. When it is found idle but channel c is busy serving a message that arrived from input port $i' \neq i, i' \neq PROC$, the tagged message also has to wait for messages on input port i that were blocked by the message occupying c. This is the second line of (26).

This completes the equations for the case with multiple processor switch channels, and the description of the model.