PERFORMANCE MODELLING AND HIGH PERFORMANCE BUFFER DESIGN FOR THE SYSTEM WITH NETWORK ON CHIP

By

JIN LIU

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

DECEMBER 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of JIN LIU find it satisfactory and recommend that it be accepted.

____
Chair

____

____

ACKNOWLEDGMENT

I would like to offer my sincerest appreciation to my department and my committee members: Dr. José G. Delgado-Frias, Dr. Kung-Chi Wang, and Dr. Sirisha Medidi for their support, guidance, willingness to work with me, and the suggestions each has offered throughout this process. In particular, I would like to thank my major advisor, Dr. José G. Delgado-Frias, for his consistent support, many helpful discussions that guide me through my whole graduate study and research. I could not have asked for more in terms of his never-ending encouragement and support in this study and make my graduate experience an extremely rewarding and positive one.

I would also like to thank the School of Electrical Engineering and Computer Science which has excellent scholars and students. The School has provided me a great environment and experience through my whole study and life at WSU.

Finally, I must also acknowledge the support and encouragement from my wife and my parents. Words can never express my thankfulness for their love and support.

PERFORMANCE MODELLING AND HIGH PERFORMANCE BUFFER DESIGN FOR THE SYSTEM WITH NETWORK ON CHIP

Abstract

by Jin Liu, Ph.D.
Washington State University
DECEMBER 2007

Chair: José G. Delgado-Frias

High performance novel dynamically allocated multi-queue (DAMQ) buffer schemes for systems with network on chip (NoC) have been proposed and evaluated in this dissertation. An analytical model to predict performance of a NoC where wormhole switching technique and fully adaptive routing protocols has been developed and compared with simulations.

In this dissertation, a novel analytical model for NoC which makes use of simple close form calculations is presented. This model provides accurate network performance prediction in the network stable region. The validity of this model is demonstrated by comparing analytical prediction with simulation results obtained on high-radix *k*-ary 2-cube networks.

Three novel switch buffer schemes, DAMQ_{all}, DAMQ_{min} and DAMQ_{shared}, for system on chip with an interconnection network are also reported. The proposed schemes are based on a DAMQ self-compacting buffer hardware design. These schemes outperform existing approaches. DAMQ_{all} have similar performance using only half of the buffer size used in traditional SAMQ implementations. DAMQ_{min} provides an excellent approach to optimize buffer management providing a good throughput when the network has a larger load. DAMQ_{shared} scheme lets virtual channels from different physical channel share free buffer space. While providing similar performance, DAMQ_{shared} scheme uses only around sixty percent of the buffer size that is used in traditional implementation for NoCs. In addition, using same size buffers, DAMQ$_{shared}$ outperforms existing approaches such as SAMQ and DAMQ$_{all}$ by 1% to 2% in throughput. The proposed schemes also make a better utilization of the available buffer space.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This chapter provides an overall perspective of the research work presented in this dissertation. Section 1 introduces network on chip which is the major topic of this study. Section 2 provides the outline of this dissertation.

1.1 System with Network on Chip

System on chip (SoC) designs are becoming widely used in telecommunication, consumer electronics and multimedia areas. As technology allows greater integration, they are being investigated in greater detail. By the end of this decade SoCs will grow up to four billion transistors [24]. SoCs incorporate a number of components (or modules) including processors, controllers and memory arrays. These components need to communicate to pass data and/or control information. Thus, a successful SoC design largely relies on the ability to interconnect these components to compute a solution efficiently.

One of the major problems for future SoC designs is the non-scalable global wire delays [31]. Global wires can carry signals across a chip, but the wire delays typically increase exponentially or at least linearly by inserting repeaters. Therefore SoC design has to rely on networking paradigms. The shared medium arbitrated bus is the most commonly used on-chip interconnect architecture, here all communicating modules share the same transmission medium. Although the bus architecture features simple topology and low area cost, the relatively long bus to which a number of components are connecting has a quite high parasitic resistance and capacitance [31]. For SoCs consisting of tens or hundreds of IP blocks, the bus-based interconnect architecture will lead to serious bottleneck problems [38]. Another alternative is to have dedicated connections between any given modules, but this design could be extremely complex as the number of modules increases.

To overcome these problems, the use of an interconnection network (direct or indirect network) within a chip has been advocated by researchers. Direct network consists of a set of nodes; each node directly connects to a limited number of other nodes in the network. Two neighboring nodes are connected by a pair of unidirectional channels in opposite directions or a bidirectional channel. Usually these on-chip computational units contain a network interface block called router (switch), which handles communication and connects to neighbor nodes' routers. Thus overlapped computation and communication are realized within each unit. As the total number of nodes in the system increases, the total communication bandwidth, memory bandwidth and processing capability of the system also increase [2]. Therefore direct networks have been a prevalent interconnection architecture.

Similar to the wide area network design, the on-chip interconnection network is proposed to be viewed as a hierarchy of services starting from the physical layer that synchronizes the transfer of bit streams to higher-level protocols layers that perform functions such as packetization, routing etc. [24]. While there is currently no consensus on a standard set of layers for the whole communication system, a three layer model to abstracting the operation of interconnection network is proposed in [2]: the routing layer, the switching layer and the physical layer. Physical layer refers to link-level wiring protocols for transferring messages and managing the physical channels between adjacent routers. Switching layer implements mechanisms for forwarding messages through the network; it determines when and how to connect router inputs to outputs and the time at which message components may be transferred along these paths. Switching layer utilizes flow control mechanisms to synchronize the message transfer; and flow control is tightly coupled with buffer management algorithms that determine how the buffers are used, i.e. how messages are handled when blocked in the network. Routing layer is responsible to make routing decision, determine the output channels at intermediate nodes.

Due to the constraints of being in a single chip, using an interconnection network on chip needs be restricted in terms of area. Thus, it is extremely important to design the schemes that require less hardware resources and still provide a good performance.

The research presented in this dissertation focuses on direct network architecture of NoC. The first part explored the ways to model a directed connected k-ary n-cube. The second part researched the methods to efficiently organize input buffer of switches used in the NoC modules.

1.2 Dissertation Outline

Chapter 2 provides a background on topics related to this dissertation. It describes network topology, switching techniques, flow control, routing algorithms and analytical models for NoC. It also introduces the virtual channel concept and buffer schemes used in NoC switch.

In Chapter 3 a novel analytical model for NoC with wormhole switching and fully adaptive routing is presented. To confirm its validity, the model is compared with results obtained from simulations.

Three novel DAMQ input buffer schemes are presented in Chapter 4. Their performance is determined by means of simulations.

Finally, chapter 5 includes a summary, some conclusion remarks, and a list of contributions of this study as well as directions for future research.

Chapter 2

Background

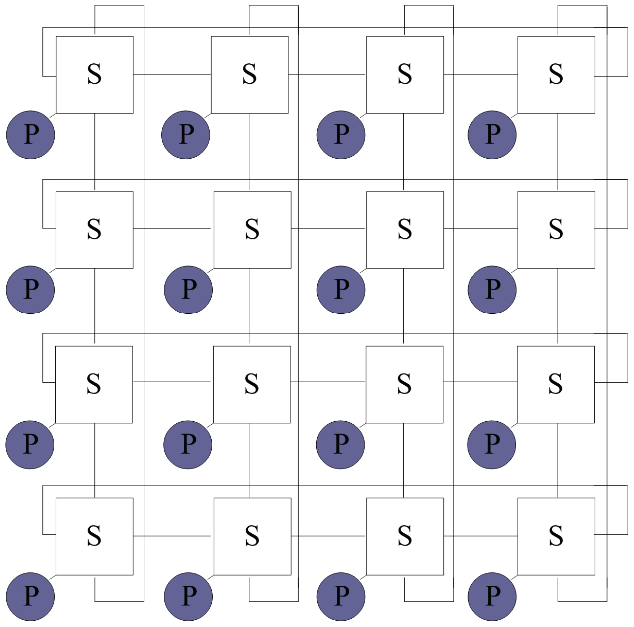
This chapter provides background on topics related to this dissertation work. The related NoC background information includes: network topology, switching techniques, flow control, routing algorithms, analytical models, virtual channel concept and buffer schemes for switch.

2.1 Network Topology

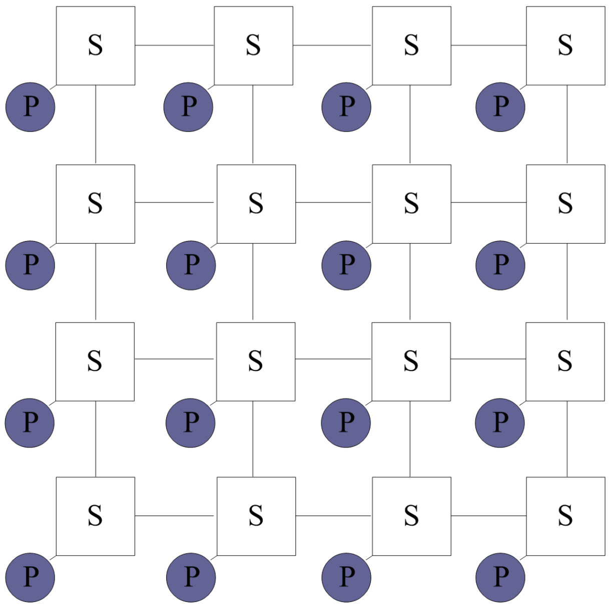
Interconnection networks are usually classified into four major classes based on network topology: shared-medium network, direct network, indirect network, and hybrid network. In shared medium networks, the communication medium is shared by all connected devices. As it was mentioned in chapter 1 the shared bus is an example of this class. Although this architecture is simple, it is not suitable for future NoCs with an increasing number of modules. In direct network, communicating devices are linked to each other by transmission channels. To transmit a message from one device to another, this message needs to traverse through several intermediate devices if the source and destination are not neighboring. On the other hand, an indirect network connects devices by one or more switches, thus any message exchanging requires information transmitting through one or more switches. Finally, hybrid network is also possible by using elements of the previous three paradigms. This research focuses on direct network as most network on chip architectures based on this design. We introduce the common direct network topology used in NoC below.

Network topology can be depicted by a graph in which nodes represent switching points and edges represent communication links [36]. Many direct network topologies have been proposed based on their graph properties. Most of the implemented networks have an orthogonal topology [2]. If every node is connected to every other node then the network is a completely connected network. The completely connected network can use simple routing algorithm and achieves high network performance. However, the cost of the complex architecture makes it impractical for network with large numbers of nodes. In terms of cost and performance, many other orthogonal network topologies such as mesh/torus and hypercube have received more attention. There are a number of network designs with different topology for network on chip reported in the literatures [27-30], among which k-ary n-cubes have been studied and used the most because of their desirable properties, such as ease of implementation, recursive structures, and ability to exploit communication locality to reduce message latency [5].

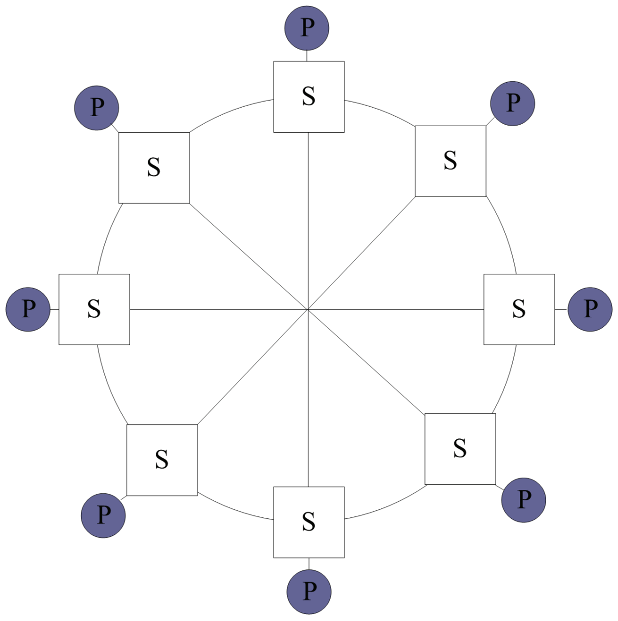
The most commonly used direct network topologies are shown in Fig 1. Each node in these networks is composed of a routing element (switch) and a process element (PE), except the tree like architecture SPIN. In the following, we briefly describe the different NoC topologies proposed recently in the literature.



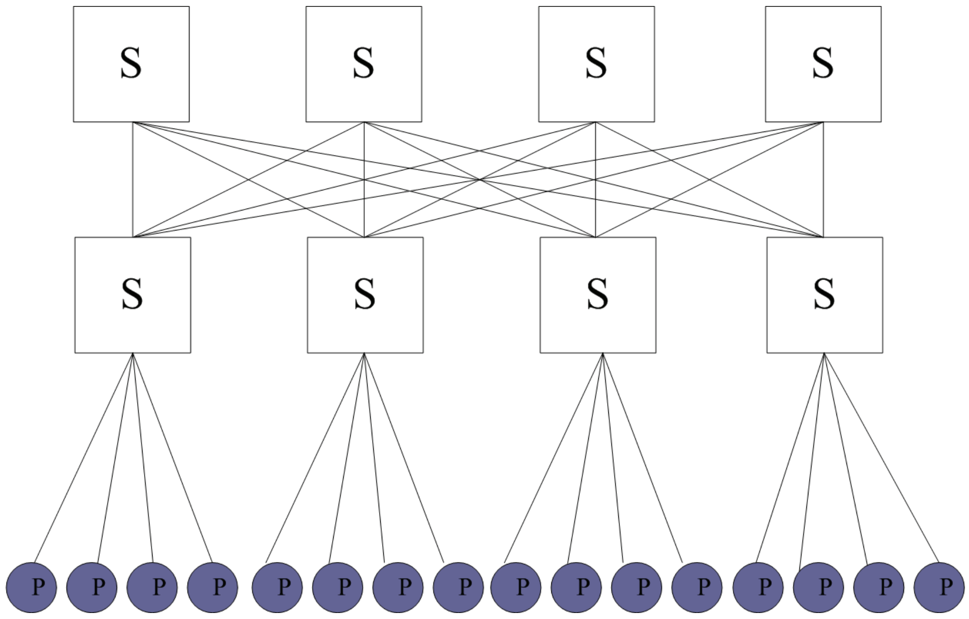
(a) 2D torus (4-ary 2-cube)



(b) CLICHÉ (mesh)



(c) Octagon



(d) SPIN (tree)

Figure 2.1: Examples of NoC topologies.

Dally and Towles [29] have proposed a 2D torus as NoC architecture as shown in Fig. 2.1a. The torus architecture is basically same as a regular mesh [2] except that the switches at the edges are connected to the switches at the opposite edge through wrap-around channels. Every switch has five ports, one connected to the local resource and the others connected to the closest neighboring switches. The long end-around connections can yield excessive delays. However, this can be avoided by folding the torus.

Kumar et al. [28] proposed a mesh-based interconnect architecture, CLICHÉ (Chip-Level Integration of Communicating Heterogeneous Elements). This architecture is based on an m x n mesh network where every switch, except those at the edges, is connected to four neighboring switches and one computation resource (PE) through communication channels. A channel consists of two unidirectional links between two switches or between a switch and a PE. Shown in Fig. 2.1b is a16 functional IP blocks network.

Guerrier and Greiner [27] proposed a tree like generic interconnect template called SPIN (Scalable, Programmable, Integrated Network) for on-chip packet switched interconnections network. A fat-tree architecture is used to interconnect IP blocks. Fig. 2.1d shows the basic SPIN architecture with 16 nodes, representing the number of functional IP blocks in the system. Every node has four children and the parent is replicated four times at any level of the tree. The size of the network grows at ($NlogN$)/8. The functional IP blocks reside at the leaves and the switches reside at the vertices. In this architecture, the number of switches converges to $S = 3N/4$, where N is the system size in terms of number of functional.

Karim et al. proposed the OCTAGON MP-SoC architecture in [30]. Shown in Fig. 1c is an octagon unit consisting of eight nodes and 12 bidirectional links. According to topology, exchanging message between any pair of nodes takes at most two hops. To design a system consisting of more than eight nodes, the octagon can be extended to multidimensional space on a significantly increased wiring complexity.

2.2 Switching Techniques

The switching techniques employed in on-chip interconnection networks initially followed those techniques employed in local and wide area communication networks, e.g. circuit and packet switching.

In circuit switching, a physical path from source to destination must be reserved prior to the data transmission. Physical path is set up by routing header flit which contains destination address and some other information. Once a routing header flit reaches the destination, a complete path is set up and an acknowledgement is transmitted back to the source. Physical path is reserved for the duration of message and may be idle for a period and block other messages. Thus circuit switching is only good for long and infrequent messages.

As an alternative to circuit switching, a message can be partitioned and transmitted as fixed-length packets by packet switching. Packets are individually routed from source to destination. A packet is stored at each intermediate node then forward to next node. Packet switching is good for short and frequent messages [2]. However, unlike in circuit switching where a physical path is reserved for the whole message, each packet of a message has to be routed at each intermediate node. Moreover, splitting a message into packets also makes overhead.

As the applications of the systems spread into more compute-intensive areas, the traditional designs borrowed from LANs become a limiting performance bottleneck. Some new switching techniques, such as virtual cut-through switching (VCT) and wormhole switching, are evolved to practical use.

In packet switching, a packet must be received in the whole at an intermediate node before making a routing decision and forwarding to the destination. However the header of a packet usually arrives to an intermediate node earlier than the tail of a packet by multiple cycles. This is because of the message length and available bandwidth. Thus even a node already knows the needed information to make a routing decision after receiving the header of a message; it still cannot forward the received part of the message to destination. To overcome this drawback, virtual cut-through switching is proposed to forward package immediately after routing decision is made. In absence of blocking, message can be cut through to input port of next node without buffering and the package transmission is pipelined through successive switches. In case a header is blocked at a busy output port, the whole message will be received and buffered at this node.

The requirement to buffer entire packets at a node makes it difficult to build compact and fast routers by using VCT switching technique. To construct small router which resides in an on-chip component, wormhole switching is used.

Wormhole switching has been widely used in practical multi-computer and network on chip where small and faster router is needed [31]. In wormhole switching, message packets are broken up into flits for message transmission flow control. Header flit contains routing information, once a header is received at intermediate node, the routing decision is made and the header is forwarded to next node in absence of blocking. The remaining data flits just follow the route that is determined by the header.

Input or output buffers of a wormhole router only need to be able to store a few flits. For example, message buffer in the Cray T3D are 1 flit deep [2]. The flits are pipelined through the network in a similar manner as VCT in absence of blocking. Because a message is typically too large to be completely stored in a router, in case the required output port is busy, all the flits of the message are stored in situ which implies that a block message occupies buffers in several routers along the path.

An example of a blocked message in wormhole switching network is shown in Fig 2.2 where message A is blocked because message C has occupied south port of node 3 and message B is blocked due to message A's occupancy of west port at node 2. Furthermore, Fig 2 shows that the flits of blocked message A are stored in situ.

This research work is focusing on wormhole switching network.

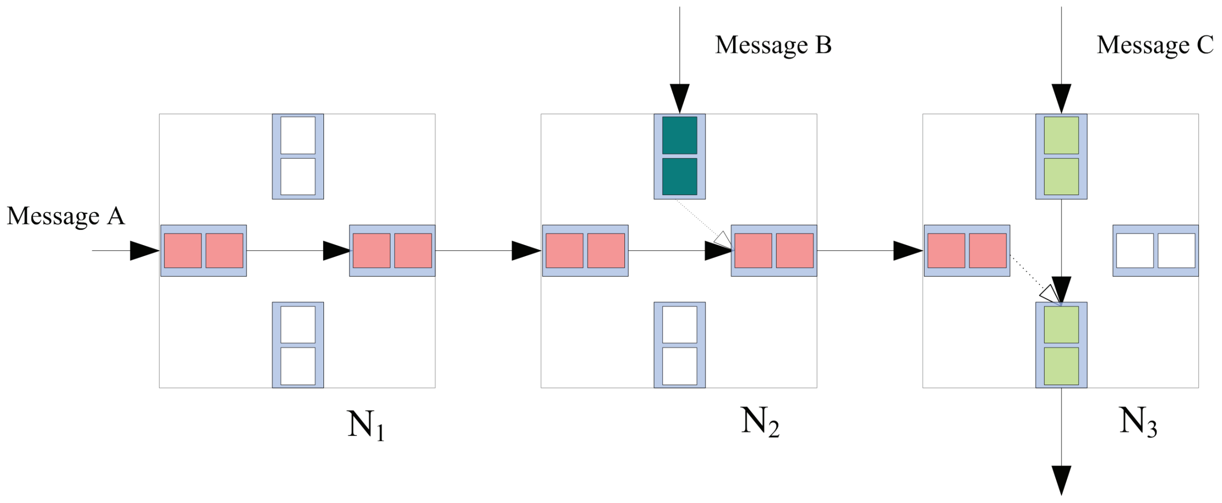


Figure 2.2: Blocked message in a wormhole network

2.3 Virtual Channel

The simple wormhole switching introduced in previous section has a drawback that different messages cannot be interleaved or multiplexed over a physical channel. Once a message starts to be transferred on a channel, this channel will be occupied till the entire message crossing it. During this time period, the occupied channel can not be used by other messages. Virtual channel [2] mechanism is introduced to increase channel utilization.

Multiple virtual channels can multiplex a physical channel. They are implemented by an independently managed pair of buffers at two adjacent nodes as illustrated in Fig 2.3 where we can see two unidirectional virtual channels multiplex one direction of the physical channel. Logically each virtual channel in Fig 2.3 operates as if a distinct physical channel which operates at half the original bandwidth.

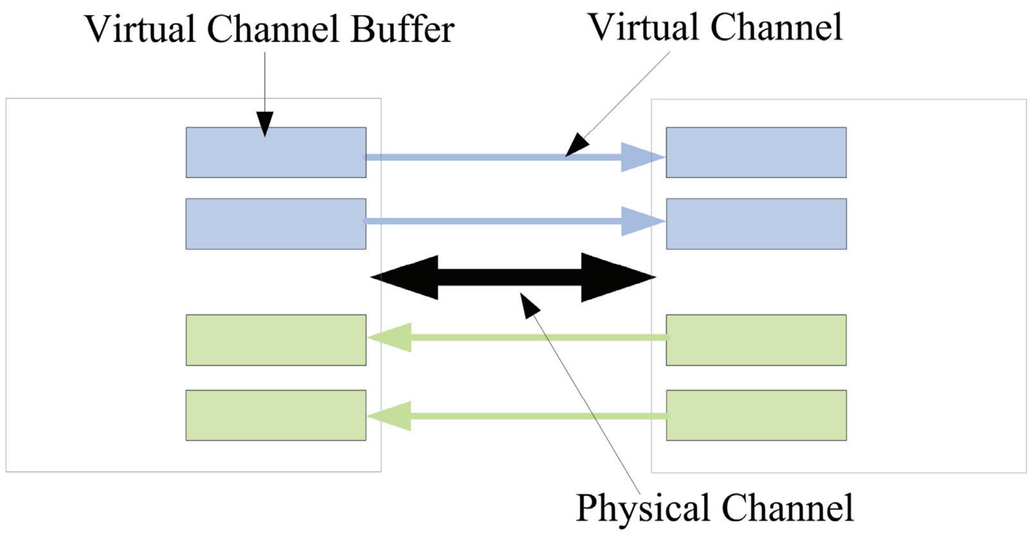


Figure 2.3: Virtual channels multiplexing a physical channel

Virtual channel can be used to improve message latency and network throughput; however it was first introduced to solve the problem of deadlock in wormhole switching network. [2] As shown in Fig 2.4, message A and message B that are blocked in Fig 2.2 can now advance to their next hop node with the help of virtual channel.

Using virtual channels may improve network performance by reducing blocking probability. However the increased channel multiplexing reduces the data transfer rate of messages, thus increases message latency. Therefore, the increase of message latency may eventually overshadow the improvement brought by virtual channels [2]. Moreover, virtual channels also introduce overhead in hardware support for this mechanism. Additional hardware blocks need to be used to make arbitration between multiple virtual channels and allocate buffer space among the virtual channels. Thus, when using virtual channels for a switch, these tradeoffs need to be taken into considerations.

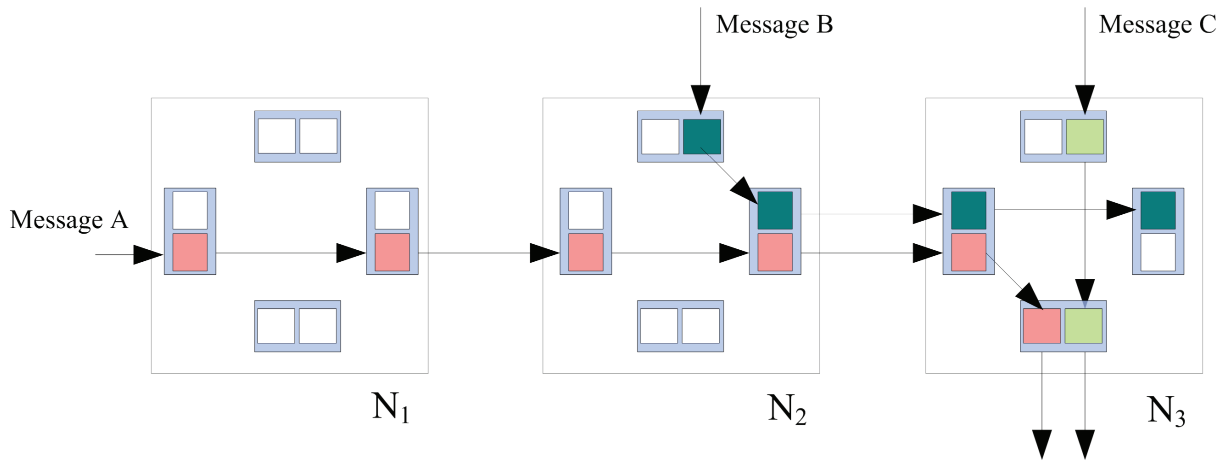


Figure 2.4: Use virtual channel to reduce blocking.

2. 4 Buffer Schemes

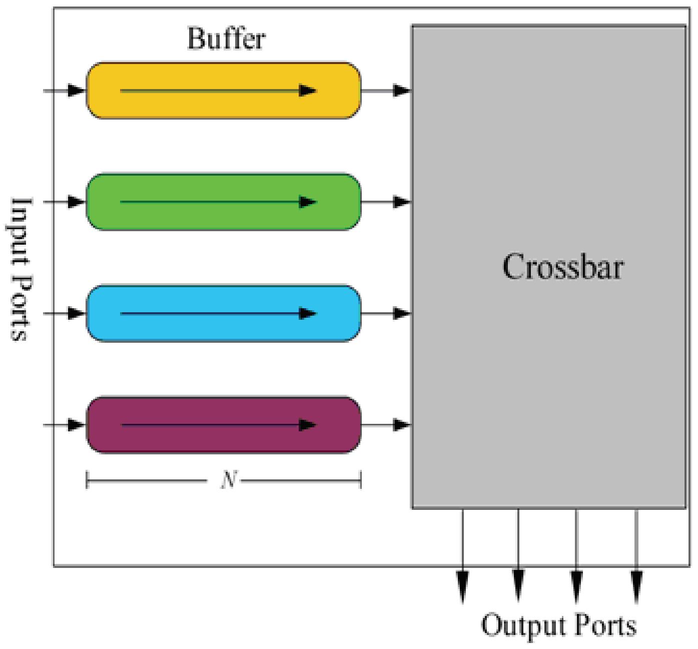
The switch buffers can be associated with input or output ports. Because buffering packets at the input ports can reduce hardware complexity [3], it is preferred over output ports buffering designs [37].

Input buffering schemes can be divided into four categories, namely FIFO (First in First out), SAFC (Statically Allocated Fully Connected), SAMQ (Statically Allocated Multi-Queue), and DAMQ (Dynamically Allocated Multi-Queue). They are illustrated in Fig 2.5.

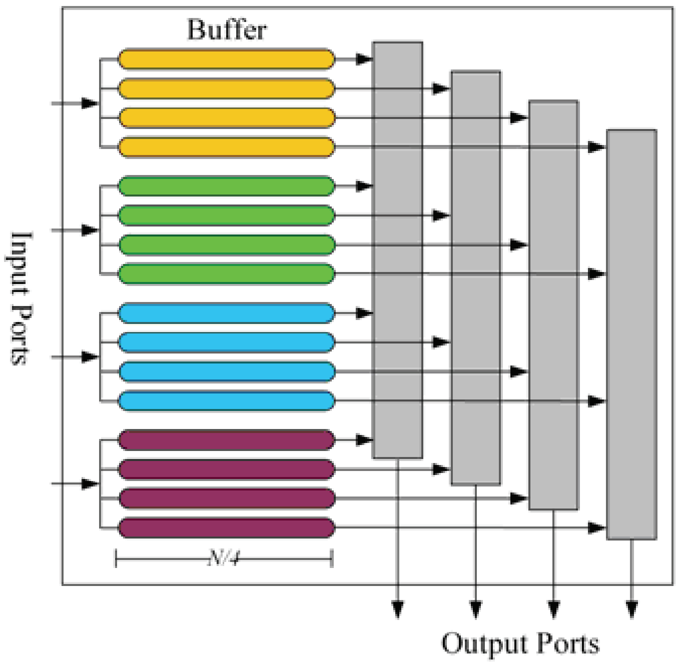
Among these designs, DAMQ efficiently utilized buffer space by dynamically allocating buffer to incoming flits. It has been proved to be a better scheme than others. [3] Following is the DAMQ buffer schemes reported in the literatures.

2.4.1 Linked list buffer scheme

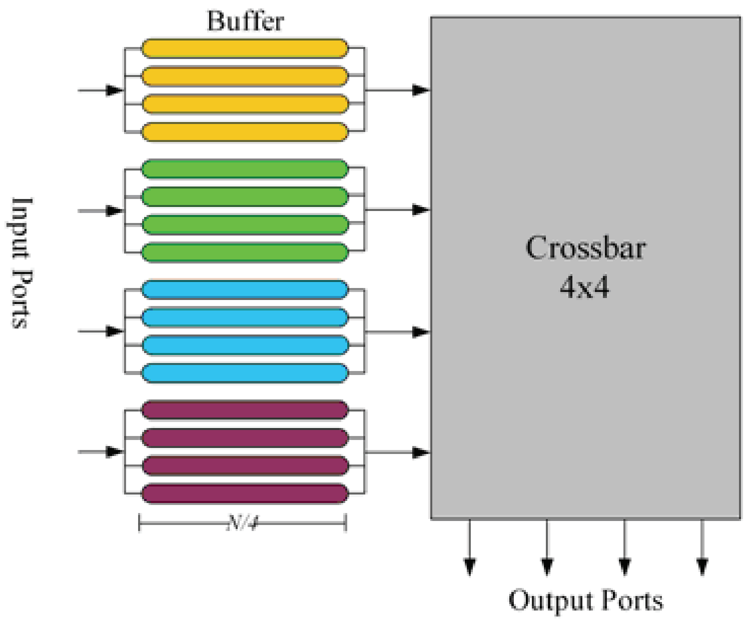
In order to let multiple queues of packets share a DAMQ buffer, linked lists can be used to implement the buffer scheme [37] [39] [40]. The basic idea of this approach is to maintain (k+1) linked lists in each buffer: one list of packets for each one of the (k-1) output ports, one list of packets for the end node interface and one list of free buffer blocks. Corresponding to each linked list there is a head register and a tail register. The head register points to the first block in the queue and the tail register points to the last block. In each output queue, next block information also must be stored in each buffer block to maintain the FIFO ordering.



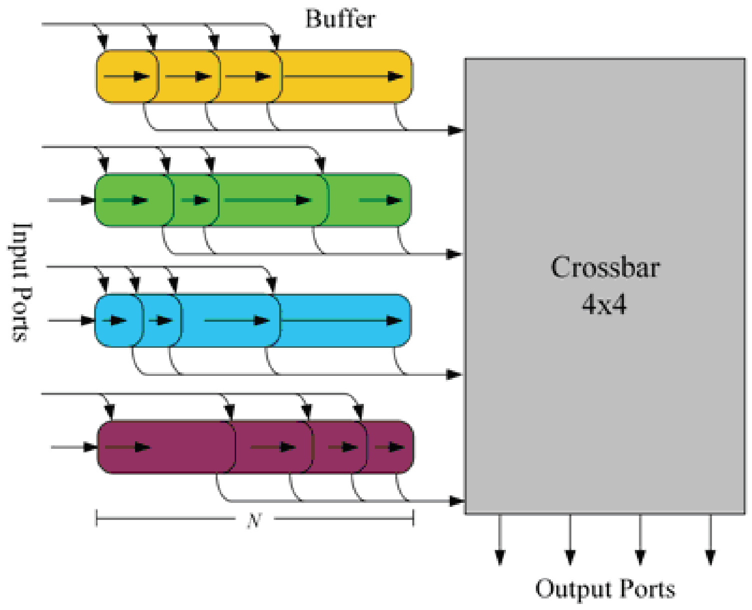
(a) First in First out (FIFO)



(B) Statically Allocated Fully Connected (SAFC)



(c) Statically Allocated Multi-Queue (SAMQ)



(d) Dynamically Allocated Multi-Queue (DAMQ)

Figure 2.5: Input buffer schemes.

2.4.2 Self-compacting buffer scheme.

To reduce the hardware complexity of the linked list scheme, an efficient DAMQ buffer design self-compacting buffer (SCB) was proposed by [40] [41]. The idea for this buffer scheme is to divide the buffer dynamically into regions with every region containing the data associated with a single output channel. If two channels are denoted as i, j with $i < j$, then the addresses of buffer regions for the two channels A_i A_j will be $A_i < A_j$. There is no reserved space dedicated for any channel. Data is stored in a FIFO manner within the region for each channel. When an insertion of the packet requires space in the middle of the buffer, the required space will be created by moving down all the data which reside below the insertion address. Similarly, when a reading operation conducted from the top of a region, data removed from the buffer may result in empty space in the middle of the buffer, then the data below the read address is shifted up to fill the empty space. Our new buffer schemes are based on the SCB scheme.

2.5 Routing algorithms and analytical models

Routing algorithms are used to specify the path from source to destination for each message. They can be implemented in two ways which are either deterministic or adaptive.

2.5.1 Deterministic routing protocols

Deterministic routing protocol chooses the path for a message only by its source and destination. All packets with the same source and destination pair will follow one single path. The packet will be delayed if any channel along this path is loaded with heavy traffic, and if a channel along this path is faulty the packet cannot be delivered. Thus the deterministic routing protocols are prone to suffer from poor use of bandwidth, blocking when alternative paths are available. They are particularly susceptible to component failures. [16] A common deterministic routing algorithm is dimension-order routing [22], in which the packet is routed in one dimension at a time, arriving at the proper coordinate in each dimension before proceeding to the next dimension. We use a dimension-order routing protocol E-Cube to conduct our simulation for two new buffer schemes.

Deterministic routing has been widely used in multi-computers due to its simplicity. [32] [33] [34] and its analytical model has been widely reported in the literature [3] [6] [7] [9] [10] [11] [35].

2.5.2 Adaptive routing protocols

Adaptive routing protocols are proposed to make more efficient use of bandwidth and to improve fault tolerance of interconnection network. In order to achieve this, adaptive routing protocols provide alternative paths for communicating nodes. Thus it can overcome the congested areas in the network. Several adaptive routing algorithms have been proposed, showing that message blocking can be considerably reduced, thus strongly improving throughput.[23] Among them, routing algorithms based on Duato's design methodology [16] are very popular. These routing algorithms split each physical channel into two virtual channel sets, the adaptive and the deterministic channels. When the paths of adaptive channels are blocked, a message uses an escape channel at the congested node. If there is any free adaptive channel available at subsequent nodes, the message can go back to the adaptive channels. Adaptive routing algorithms can be further categorized to progressive and backtracking algorithms. Progressive routing algorithms move the message header forward by reserving a new channel. Backtracking algorithms needs more complex hardware support [2] because they allow the message header to be routed backtrack releasing previously reserved channels. In our model, we assume the routing algorithm is progressive, no backtracking is allowed.

Two analytical models for wormhole switching network using fully adaptive routing protocols are reported in the literature [4] [5]. Y.Boura et al. proposed an analytical model for adaptive routing Hypercube in [4]. In Hypercube network a message traverses at most one channel along a dimension. Due to this topology nature, at an intermediate node, the number of remaining hops for a message to arrive at destination is known in Hypercube. However, *k*-ary *n*-cubes do not have this property. Thus, at a given intermediate node the number of remaining hops that remains on current dimension is not clear. M. Khaoua proposed a model for Duato's fully adaptive routing algorithm in k-ary n-cubes in [5]. Its key idea was to compute the blocking probability at a given node by deriving the blocking probability of the two virtual channel groups in Duato's algorithm i.e. adaptive and deterministic virtual channels groups respectively.

In this dissertation we present a novel performance analysis model for *k*-ary *n*-cubes network on chip with wormhole routing and fully adaptive routing. We use a 2D torus network as the analysis example to compute the average message latency using our new model. The model is based on general queuing theory and probability analysis. The blocking probability for a message at a given channel is derived by the arrival traffic rate and the service rate the message receives. It's simple and yields satisfactory predictions in the network's steady state region.

Chapter 3

Analytical Model for Wormhole Switching NoC

This chapter presents a novel analytical model for easily predicting average message latency of traffics in a network on chip where wormhole switching and fully adaptive routing protocols is used. The model has simple close-form calculations and produces very accurate results in the network stable regions. To validate this model a comparison between analytical prediction and simulation results is performed on high-radix *k*-ary 2-cube networks.

3.1 Network Configuration

We use a 2*D* torus network as our analysis target to illustrate the model for simplicity. Fig. 3.1 shows an 8-ary 2-cube (torus) message exchanging network. Each node consists of a processing element (PE) and a switch. PE is responsible for generating message and consuming message from other nodes. Each switch has 5 input and output channels. PE is connected to switch by the local injection/ejection channel. A node is connected to 4 adjacent neighbouring nodes by bi-directional network channels.

As introduced in chapter 2, virtual channels multiplexing one direction of a duplex physical channel are used in network on chip to enhance throughput and avoid deadlock. Due to the nature of limited hardware resource and to facilitate our model calculation, we assume the buffer size for each end of a virtual channel is 1 flit deep.

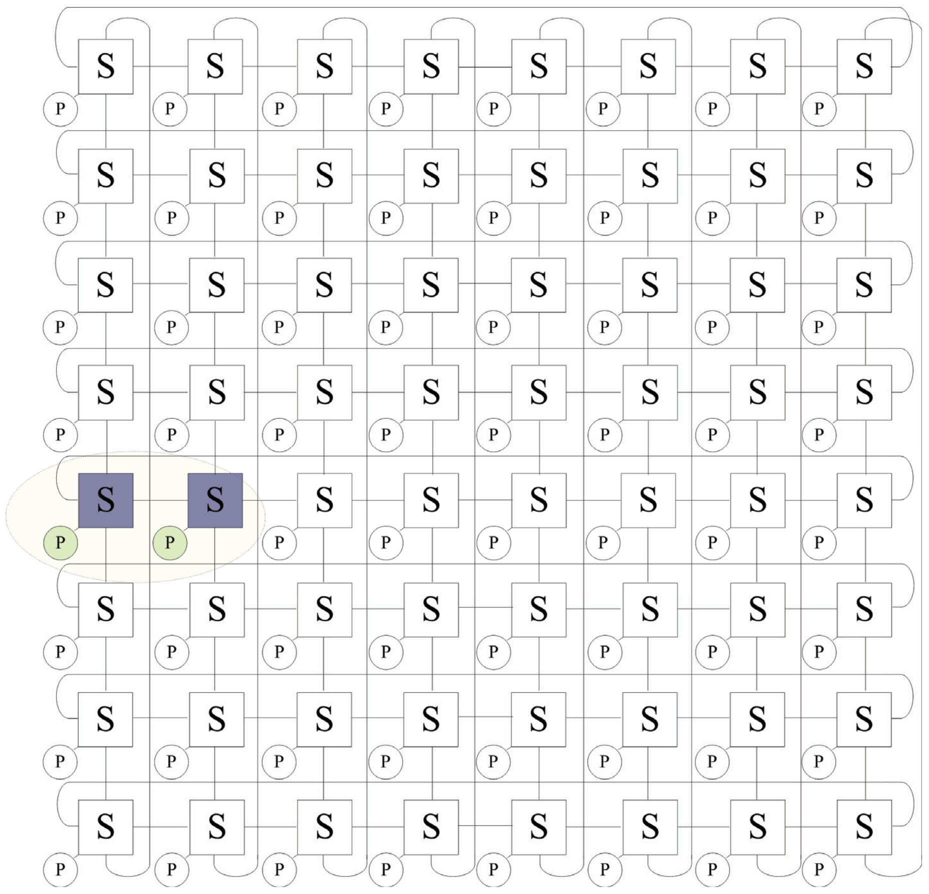


Figure 3.1: 64 nodes torus message exchanging system

A message which is generated from a nodes PE will first be transmitted to the switch by local injection channel. Then, this message will be routed toward its destination. At the destination node, the message is transmitted to PE through local ejection channel. Thus a message has to travel through at least 3 links from the source to destination. For instance, as shown in Fig. 3.2, the message generated in node N_1 has to traverse channels $Ch_{1,L}$, $Ch_{2,W}$ *and* $Ch_{2,L}$ to arrive at its destination PE in N_2.

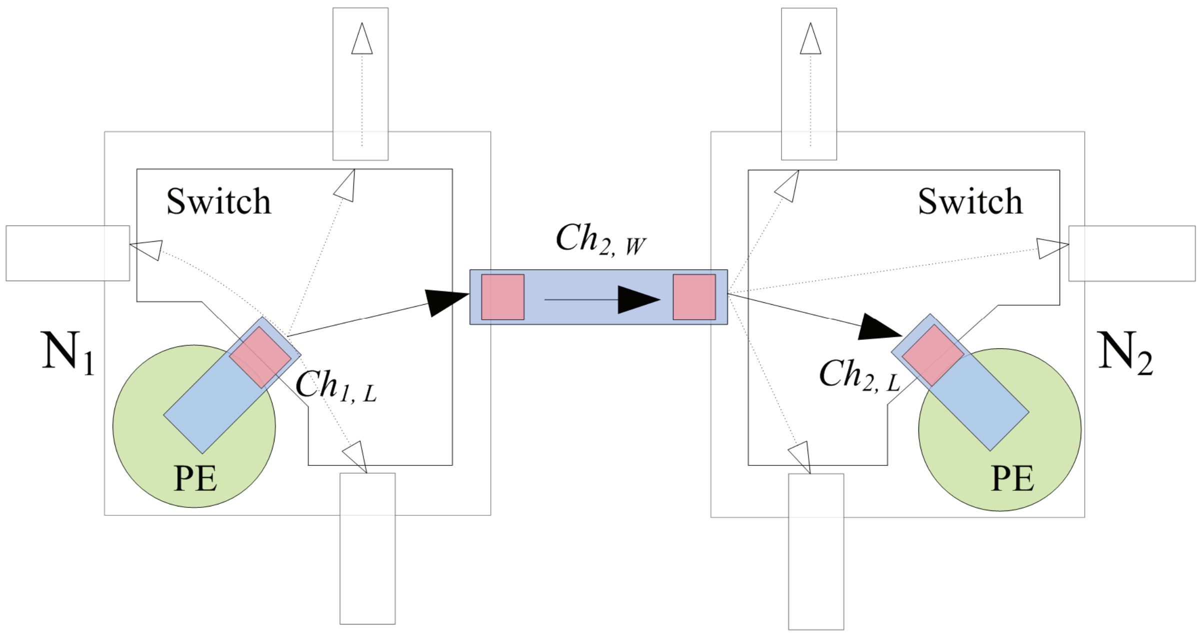


Figure 3.2: Switch to switch path

3.2. Pertinent Assumptions

For this model the following assumptions have been made; these assumptions are commonly accepted in the literature [2-13].

1) *Independent traffic generation.* Each node generates traffic independently with the traffic following a Poisson process on a mean rate of M_{gen} messages per cycle.

2) *Uniform distribution of destinations.* Message destinations are uniformly distributed across the network nodes. Although for an actual application, if node *A* sends a message to node *B* it's highly possible that *B* will send back a message to *A*.

3) *Messages of constant length L flits.* A message is long enough so that its data flits span from source to destination nodes. Moreover, the message lengths can be designated according to any probability distribution where expectation and variance of message lengths are known.

4) *One clock cycle transmission between adjacent nodes.* Each flit requires one cycle to be

transmitted from one node to the next over the physical link between them. Two cycles are needed for a flit to cross a node, i.e. from an input buffer to an output port, in absence of blocking.

5) *Infinite capacity in local injection queue.* The local queue at the injection channel in the source node has infinite capacity. Messages at the destination node are transferred to the local PE one flit at a time through the ejection channel.

6) *Duplex physical channels between any two adjacent nodes.* More than two virtual channels are used for each direction of a physical channel. In Duato's algorithms [15] [16], if there are adaptive virtual channels available, a message can use a random one; for the deterministic virtual channels, although there are two of them, a message uses only one at a time. Therefore we adopt the same strategy as described in [4] to make no distinction between the deterministic and adaptive virtual channels when computing the different virtual channels occupancy probabilities. This simplification also reflects the idea of fully adaptive routing.

3.3 Notations

There are a number of notations involved in this model. We list all of them into Table 3.1 to facilitate the name look up.

Table 3.1. The Parameter Notations

Parameter	**Description**
M_{gen}	Average message generation rate at each node
T_{msg}	The mean latency for all the delivered messages
T_s	Routing (Switching) delay across a node
T_w	Propagation delay across the physical channel
L	Average message length(not including header)
D	Average path length for all the delivered messages
N_q	Average number of intermediate nodes along the path
W_q	Average waiting time for a message at each intermediate nodes along the message's routing path
W_{ej}	Average waiting time for a message on ejection node channel
λ	The average message arrival rate at a channel
μ	The service rate for a message at a physical channel
S	The service time for a message at a channel without contention.
SR	The service rate for a message at a channel without contention
W_{eq}	Average waiting time for a message on last node of the routing path.
μ_r^{dir}	The service rate that a channel observes from the immediate downstream channel on *dir* direction.
μ_r	The service rate that a channel observes from all its immediate downstream channels at nodes on the path.
μ_{rE}	The service rate that a channel observes from all its immediate downstream channels at destination node.
P'_Q	The blocking probability for a message at a physical channel without contention.
P_Q	The blocking probability for a message at a physical channel
v	The virtual channel number of each direction of a physical channel
o	In/output ports number of a node (not including local channel port), in our network, it's 4.
dir	Direction. Each nodes has ports on 5 directions, *i.e.* East, South, West, North and Local
$Ch_{n,\ dir}$	Physical channel in *dir* direction of a message's *n*th hop node.

3.4 Calculation of Average Message Latency

Average message latency is a key metric to evaluate a network's performance. It's defined as the time elapsed since the message transmission is initiated until the message is received at the destination node [2]. In the following, we present how to calculate the average message latency. First we present the notations used in the computing process, and then we describe the calculation process.

The average message latency T_{msg} comprises of the message transmission delay across the network channel t_w, the intrarouter delay t_s, [2] the average contention delay W_q at the network channels and the average delay W_{ej} at nodes' local ejection channel. It can be computed as follows:

$$T_{msg} = \left(N_q - 1\right) \cdot W_q + W_{eq} + D \cdot T_{tr} + L \cdot T_{tr} + W_{ej} \tag{1}$$

Where T_{tr} is given by max(t_s, t_w). It demonstrates the nature of pipelined flits transmission of wormhole switching in absence of contention. DT_{tr} denotes the mean time that a message's header flit needs to travel from source to destination and LT_{tr} denotes the travel time for the data payload of a message. $(N_q -1)W_q$ shows the waiting time that header flit experienced at the $N_q -1$ channels of intermediate nodes. Queuing system at the destination node of a messages routing path, W_{eq}, is separated from other queues on previous nodes because of the different service rate offered by last node. We will discuss the detail later. As the minimum link number that a message travels is 3, the average hops that messages take, D, can be obtained by:

$$D = \sum_{i=3}^{k} p_i \cdot i \tag{2}$$

Where k is the diameter of the network, and p_i denotes the probability that a message's travel path is i links long. Another simple and commonly used estimation of D is:

$$D = \frac{k+3}{2} \tag{3}$$

However, in a low radix network, Eq. (3) is prone to produce error. For example, *D* of the 4 nodes Torus network is 3.33, while Eq. (3) yields 3.5 in this case.

Under the uniform traffic pattern, the average traffic arrival rate λ for each channel is determined by the message generating rate M_{gen}, average routing hops *D* and output channels number of each node o [4].

$$\lambda = \frac{M_{gen} \cdot D}{4} \tag{4}$$

In order to receive service from a link, the message's header flit needs to acquire a virtual channel. Once a virtual channel is assigned to a message, it keeps serving this message and will not be released until all the data flits flow across this node. Because each virtual channel has one flit depth buffer, once it is assigned to a message, no other message can use the same virtual channel till it's free again. When the traffic rate is light, there is no congestion; the service time at each channel can be defined as:

$$S = t_s \cdot (L+1) \tag{5}$$

And the service rate can be derived accordingly:

$$SR = \frac{1}{S} \tag{6}$$

When traffic rate is high enough, congestion appears in the network and waiting queues build up at corresponding bottleneck links. In this case, the service that one channel can provide to the incoming messages is not only determined by its own service capacity, but also by the blocking state of its immediate downstream channels.

In our network, the traffic arrival rates for the four input channel of a node are equal to each other because of the following reasons:

1. Torus network with wrap around links is a strictly symmetric topology.
2. Every node is identical in terms of capacity of generating and consuming message.
3. Traffic is generated randomly from all the nodes.
4. The routing protocol is fully adaptive algorithm.
5. The input channels all have equal service capacity.

So we can treat the queuing systems at each of these channels as identical. Without loss of generality, suppose the node that we analyze is at *n*th hop of a messages routing path. We derive the queuing system model of the channel $Ch_{n,W}$ on the west input port as shown in Fig. 1.

We follow the suggestion in [11] to treat the waiting queue $Q_{n,W}$ at channel $Ch_{n,W}$ as two distinct queues Q_c and Q_d. As shown in Fig. 5, Q_c is result from the delay involved in router (switch) service delay observed by $Ch_{n,w}$, while Q_d is due to the contention that a message may experience when it's to be accepted by a downstream input channels.

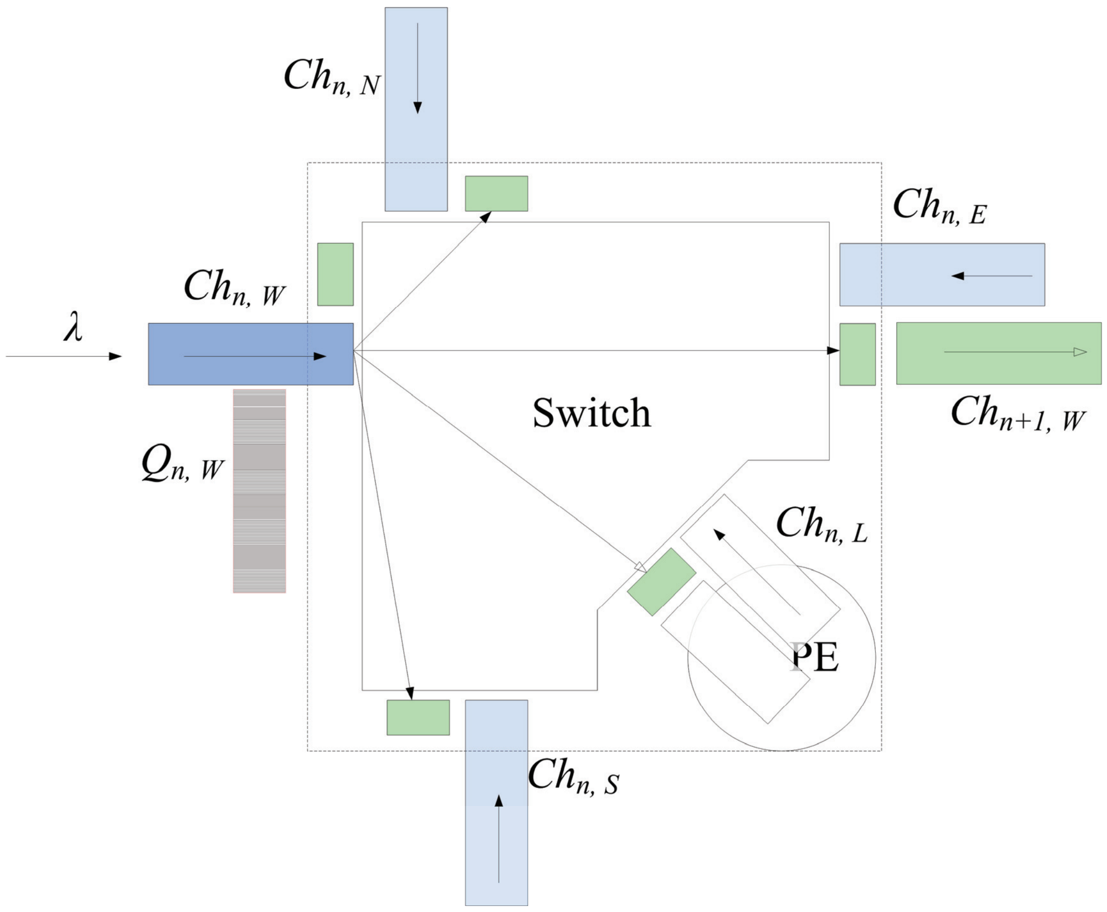


Figure 3.3: Queuing model for an input channel

Q_c is determined by the traffic arrival rate λ and the router self's service rate *SR*. To model it, we use an *M/M/m* queuing system. The first two "*M*"s stands for the Poisson distribution traffic arrival process and the exponential distributed service time respectively. The third "*m*" means there are *m* servers and a message at the head of the queue is routed to any server that is available. Accordingly we have *v* virtual channels per physical link in our network configuration, which are treated as the *v* servers in the queuing model.

The probability that an arrival message will find all virtual channels are busy and will be forced to wait in queue can be obtained by [1]:

$$P_Q^{'} = \frac{p_0 \cdot (v \cdot \rho)^v}{v! \cdot (1-\rho)} \tag{7}$$

Where p_0 is given by:

$$p_0 = \left(\sum_{n=0}^{v-1} \frac{(v \cdot \rho)^n}{n!} + \frac{(v \cdot \rho)^v}{v! \cdot (1-\rho)} \right)^{-1} \tag{8}$$

And ρ is given by:

$$\rho = \frac{\lambda}{v \cdot SR}$$

For the second queue Q_d, the traffic arrival rate is still λ, and the service rate, μ_r, is the service rate that offered by all the immediate downstream channels to $Ch_{n,w}$. However μ_r at destination node is different from those at intermediate nodes along the path. This is because at destination, the message is bound to be delivered into local ejection channel, thus other three possible immediate downstream channels cannot contribute service to μ_{rE} in this case. Correspondingly, at other nodes along the path, local ejection channel won't offer service to a message. We first show how to compute μ_r at intermediate nodes and use it to get the average waiting time for a message at these nodes when contention is taken into consideration. After that, we'll show the calculation for μ_{rE} and the corresponding waiting time at the destination node.

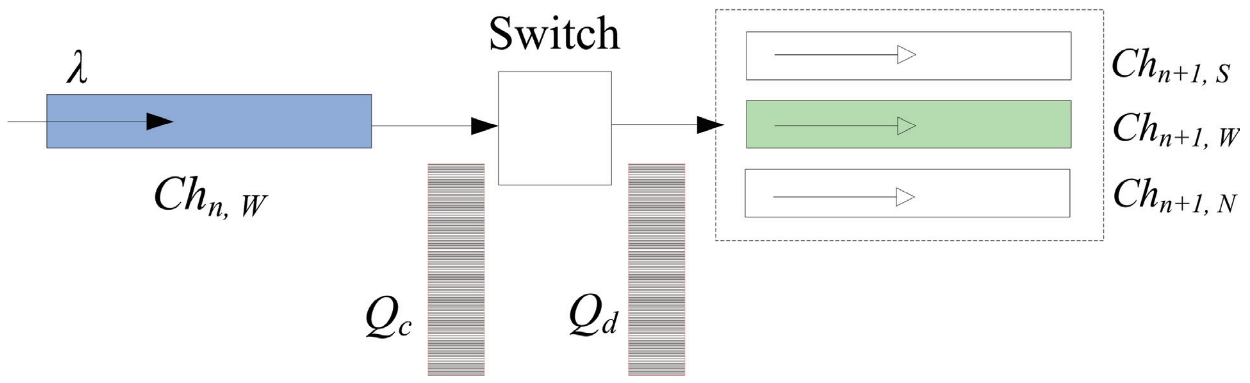


Figure 3.4: Reduced queuing model for an input channel

To obtain μ_r at intermediate nodes, we can further divide Q_d to three queues, each of which is associated with an immediate downstream channel in one of the three possible downstream directions, *i.e.* east, south and north in this case. Note, the routing algorithm that we assume in the model is progressive algorithm, so we don't treat the backtracking channel as an immediate downstream channel. However, if the model needs to be used to model a backtracking routing algorithm, then service from one more possible downstream channel ($Ch_{n-1,E}$ in this case) in the reverse direction can be easily added in.

As we mentioned before, the traffic arrival rates for the three queues on each direction of next hop are equal to each other. Moreover, these downstream network channels have same service capacity, so we can get the analysis results for all these queues by analyzing any one of them. Without loss of generality, let's consider the waiting queue residing at the west downstream channel $Ch_{n+1,W}$. The channel can accept a new message when it has free virtual channel available, while no messages can be accepted when all the virtual channels are occupied. Furthermore, $Ch_{n+1,W}$'s service capability, *i.e.* the number of flits that can be processed within a unit time (clock cycle) is fixed, so with more traffic generated in the network, the probability of one message to be served is less; in another word, the waiting time for this message is longer. Since we can get the average waiting time for a message by: [1]

$$Wq_d = \frac{P_Q^{'} \cdot \rho}{\lambda \cdot (1-\rho)} \tag{9}$$

Then the average time that a message spends at the queuing system of $Ch_{n+1,W}$ is $W_{qd} + S$. In addition, there are v virtual channels in $Ch_{n+1,W}$ and $Ch_{n+1,W}$ has o possible inputs(including local injection channel), therefore we can get μ_r^w as :

$$\mu_r^w = \frac{v}{o \cdot (S + W_{qd})} \tag{10}$$

Hence, we get μ_r as:

$$\mu_r = p_W \mu_r^W + p_S \mu_r^S + p_N \mu_r^N \tag{11}$$

Because of the symmetry of our network, we know that $p_W=p_S=p_N$, therefore $\mu_r=3\,\mu_r^w$. For networks with different topology or traffic pattern, we can get μ_r by deriving the probabilities of the traffic flows to downstream links on different direction. Using Little's Theorem, the average number of messages in the whole queuing system at $Ch_{n,W}$ can be derived as:

$$N = \lambda \cdot T = \frac{\lambda}{\mu} + \frac{\lambda \cdot P_Q}{v \cdot \mu - \lambda} \tag{12}$$

Note that μ in Eq. (12) is the overall service rate that $Ch_{n,W}$ provides to its incoming traffic. It takes into consideration of both with and without contention cases on downstream channels. Also by Little's Theorem, the average number of messages in Q_d is:

$$N_d = \frac{\lambda}{\mu_r - \lambda} \tag{13}$$

And the average number of messages in Q_c is:

$$N_c = \frac{\lambda}{SR} + \frac{\lambda \cdot P_Q}{v \cdot SR - \lambda} \tag{14}$$

As $Q_{n,\,E}$ comprises of Q_c and Q_d, combine Eq. (12), (13) and (14), we get:

$$\frac{\lambda}{\mu} + \frac{\lambda \cdot P_Q}{v \cdot \mu - \lambda} = \frac{\lambda}{SR} + \frac{\lambda \cdot P_Q}{v \cdot SR - \lambda} + \frac{\lambda}{\mu_r - \lambda} \tag{15}$$

From Eq. (7) and (8), we already knew that P_Q in Eq. (15) can be expressed in terms of λ and μ:

$$P_Q = \frac{\left(\sum_{n=0}^{v-1} \frac{(v \cdot \rho)^n}{n!} + \frac{(v \cdot \rho)^v}{v! \cdot (1-\rho)} \right)^{-1} \cdot (v \cdot \rho)^v}{v! \cdot (1-\rho)} \tag{16}$$

Where ρ is given by:

$$\rho = \frac{\lambda}{v \cdot \mu}$$

Replace P_Q in Eq. (15) using Eq. (16), solve the nonlinear equation, we can get the service rate μ at channel $Ch_{n,W}$.

Then, by Little's Theorem again, we can obtain the average time W_q that a message has to wait in queuing system at the intermediate nodes of a message's traveling path:

$$W_q = \frac{P_Q \cdot \rho}{\lambda \cdot (1-\rho)} \tag{17}$$

At the destination node, say N_{des}, μ_{rE} is solely determined by the local ejection channel. But we cannot apply Eq. 9 and 10 directly, because there may be more than one message choose N_{des} as its destination node which implies that local ejection channel has a different arrival traffic rate than network channels. We can derive the message arrival rate from other sources by:

$$\lambda' = \sum_{i=1}^{o-1} \lambda \cdot (n-i)^i \tag{18}$$

Where n is the total nodes number of the network. Since the node's local ejection channel doesn't have any downstream channel dependency; we treat it as an independent *M/M/m* queuing system, where "*m*" also equals to virtual channel number v. Thus we can use Eq. (8) to calculate the average waiting time for a message, W_{qde}, in absence of contention by replacing λ to $\lambda+\lambda'$. So we obtain μ_{rE} as:

$$\mu_{rE} = \frac{v}{S + W_{qde}} \cdot \frac{\lambda}{\lambda + \lambda'} \tag{19}$$

Repeating the computing process expressed in Eq. 12, 13, 14, 15, 16, 18, by providing the ρ for this case:

$$\rho = \frac{\lambda + \lambda'}{v \cdot \mu}$$

We can obtain the waiting time at the input channel of destination node, W_{eq}.

Finally, as we mentioned that the nodes local ejection channel is treated as independent *M/M/m* queuing system; the average waiting time on ejection node channel can be obtained by:

$$W_{ej} = \frac{P_{Q_{ej}} \cdot \rho_{ej}}{(\lambda + \lambda') \cdot (1 - \rho_{ej})} \tag{20}$$

Where $P_{Q_{ej}}$ can be obtained by Eq. (16) with

$$\rho_{ej} = \frac{\lambda + \lambda'}{v \cdot SR} \tag{21}$$

At this point, the message delay T_{msg} defined in Eq. (1) can be easily calculated out as all the unknown variables at the right hand side of the equation are all obtained now.

3.5 Validation

We carried out a number of simulations to validate the proposed model. The simulator we used is flexsim1.2 [18] [20], which is a flit level simulator for Torus/Mesh network adopting wormhole switching. In flexsim1.2, when uniformly distributed traffic is used in simulations, the source of a message is equally randomly picked among nodes in the network. Thus we can treat the traffic that arrives at a given channel follow Poisson distribution. Parameters of the simulated network are set to conform to the assumptions described in section 3. Validation experiments are carried out on many different combinations of network size, message length and virtual channel number per physical channel. In order to facilitate the illustration of our model's validity, the messages' data payload is set to 32 flits and the number of virtual channels per physical channel is set to 4. All the simulation results are obtained from simulations running for over 1 million cycles simulating time to rule out exceptional results. In the follows, we present average message latency results obtained from both model and simulations on three different size networks, 64, 256 and 1024 nodes two dimensional Torus network. Figures 3.5, 3.6 and 3.7 demonstrate average message latency results predicted by the model against those obtained from the simulations. These results are obtained from 8-ary 2-cube, 16-ary 2-cube and 32-ary 2-cube networks respectively. The figures reveal that simulation results and predictions of our model match well from a very light usage of a network channel to about 50% average utilization, after which point discrepancies are apparent as the network gets into saturation state.

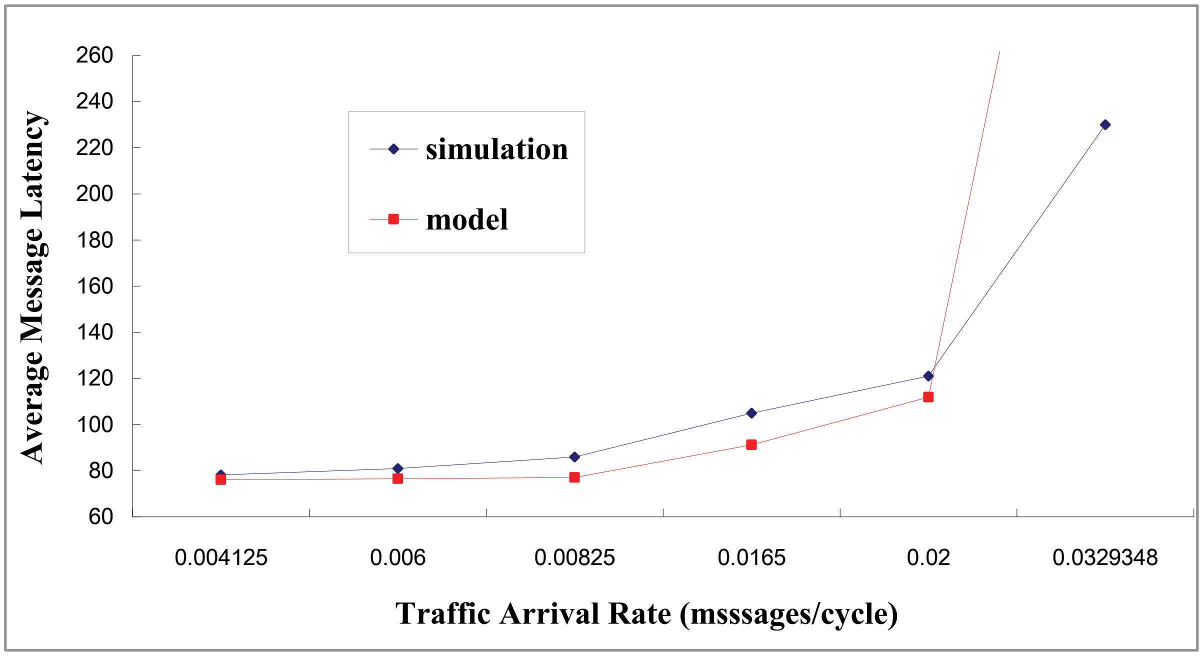


Figure 3.5: Comparison of Model against simulation in 64 nodes torus network

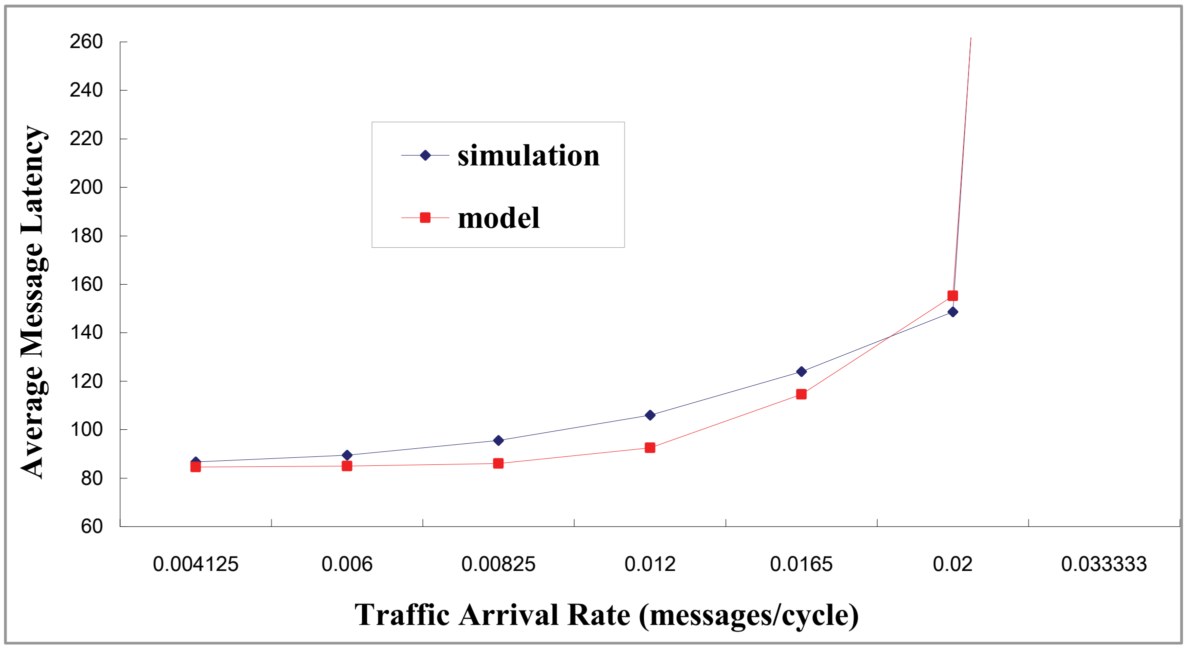


Figure 3.6: Comparison of Model against simulation in 256 nodes torus network

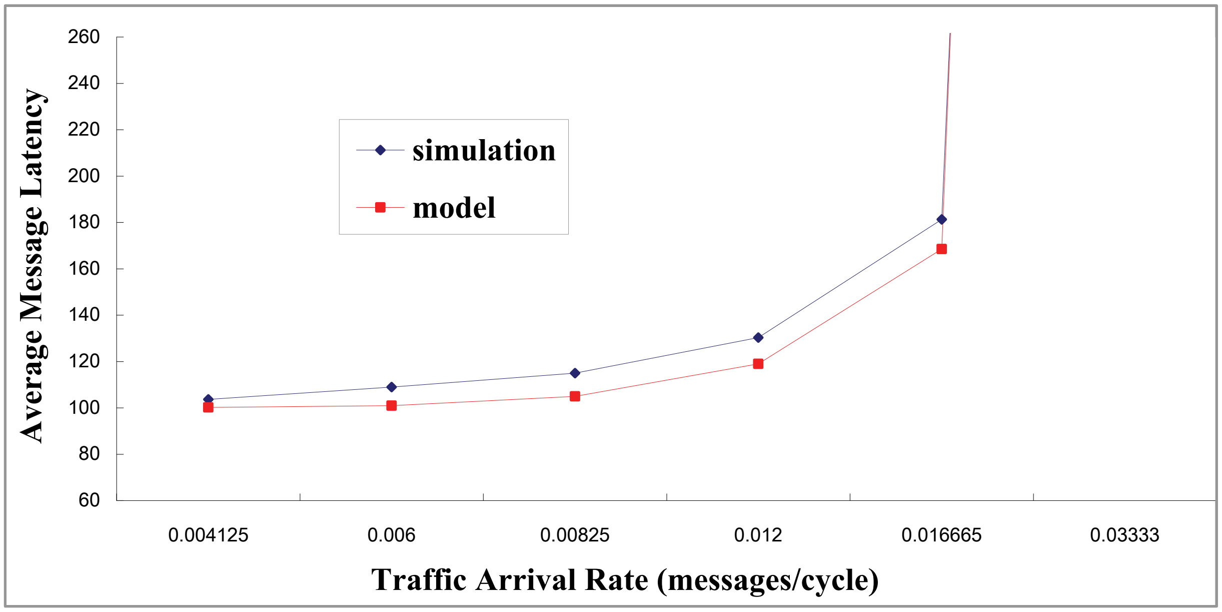


Figure 3.7: Comparison of Model against simulation in 1024 nodes torus network

3.6 Apply model to multiple flit buffer

We have presented the detailed calculation process to use the proposed analytical model predicting performance of a *k*-ary 2-cube network in section 3.4. In section 3.5 a comparison with simulation results is shown. In that calculation process, to make it simpler to illustrate the core idea of the model we assume the buffer size at each virtual channel end is 1 flit deep. Although there are machines that implement 1 flit buffer, it is also possible to have larger virtual channel buffers. In this section we introduce an approach to apply our model to Statically Allocated Multi-Queue (SAMQ) buffers which are more than 1 flit deep.

The following are notations that are needed in this extension of the model:

1. *Buf*: Buffer Size of each virtual channel
2. D_{sm}: Number of nodes needed to store message in virtual channel buffer when message is blocked. It is obtained by *L/Buf* where L is message length
3. D_i: Distance between *i*th intermediate node to destination node.

Due to the pipelined transmission nature of wormhole switching, if there is no congestion in the network, then buffer size has no impact on flowing message because all the flits are cut through to next hop channel without storing in current node's channel buffer.

In case the message is blocked in an intermediate node's channel, all the flits will be stored in the intermediate node buffers. Thus, if there is more buffer space available at each intermediate node's channel, then more flits can be stored in these channels. Therefore, for a channel that are near the source node, its D_{sm} may be smaller than its D_i. In contrast, if each virtual channel has 1 flit buffer, then D_{sm} is equal to D_i. This is to say that the channels near the source node side may be freed right after transmitting flits and the possible congestions of message *M* in subsequent nodes have no impacts to this channel after it finishes transmitting all the flits of *M*. This is because all the flit can be stored in D_{sm} virtual channels that precede the congested node.

In wormhole switching, once a virtual channel is assigned to a message, this virtual channel cannot be used for other messages no matter what's the size of the virtual channel buffer. Therefore, although more buffer space means more data flits can be stored in the channel buffer, the service rate that a channel can provide to the incoming traffic still remains same. In another words, using the notations of analysis in section 3.4, the service rate of a physical channel is still determined by the *v* servers that are the virtual channels multiplexing it. Therefore we can use the average waiting time for a message at each intermediate node, W_q, which is obtained in section 3.4 to estimate the message waiting time in the multiple flit buffer case.

Using the notations in section 3.4, for multiple flit buffer virtual channels, we can obtain the average time that one virtual channel may be occupied by a flowing message *M* as:

$$(L+1) \cdot T_{tr} + \min(D_{sm}, D_i) \cdot W_q \qquad (1)$$

where $(L+1)T_{tr}$ denotes the time that this channel spends to transfer the header and data payload flits of a message *M* and $\min(D_{sm}, D_i)W_q$ shows the time span that flits of *M* stay in this virtual channels buffer. As mentioned earlier, we know that if 1 flit buffer is used, then $D_{sm} = D_i$, therefore, the average time that one virtual channel may be occupied by a flowing message *M* in this case is:

$$(L+1) \cdot T_{tr} + D_i \cdot W_q \qquad (2)$$

Subtract (1) from (2), we get difference between the time that a flowing through message spent in a 1 flit buffer channel and the time it spent in multiple flit buffer channel:

$$(D_i - \min(D_{sm}, D_i)) \cdot W_q \qquad (3)$$

Using equation (2) and (3), together with the parameters of a analysis target network, we can obtain the percentage of the average time reduction of message *M* staying in a virtual channel, which in turn can be used to estimate the improvement in terms of the physical channel service rate. Then we can apply the model presented in section 3.4 to predict the performance of networks with multiple flit virtual channel buffer.

As to apply the model to predict network with Dynamically Allocated Multi-Queue buffer, we need to analyze the pattern of how the buffer space is partitioned among the competing virtual channels. It is more complex than the previously mentioned SAMQ case. However it would be a very good topic for future extension of this research.

3.7 Conclusion

We have presented an analytical model for predicting network performance in this chapter. This model can be applied to wormhole switching network to predict network performance measures, such as average message latency and average link waiting time in a torus network using fully adaptive routing algorithms such as Duato's method. Unlike previously proposed models, this model is based on general queuing theory and probability analysis. It's simple and yields rather accurate predictions in network's steady state region. Compared with the model [5] in the literature for wormhole switching *k*-ary *n*-cubes network with adaptive routing, our model computes the blocking probability using message arrival rate and service rate provided by network channels and can predict network performance accurately from very light traffic load to the saturation point where about 0.2 messages are received/sent at each network node per cycle. While the model proposed in [5] conducted the calculation based on virtual channel groups in Duato’s routing algorithm and predicted network performance in the traffic load region from 0 to about 0.002 messages per node per cycle.

By applying close-form calculations of this model, we can correctly predict high-radix *k*-ary *n*-cubes in steady state regions with a short code snippet. It can be concluded that our model provides an effective and practical evaluation tool. In addition, since this model obtains message waiting time at each queuing system at channels of a routing path, it can be easily adapted for networks with other topologies. For instance, mesh network. The only required information is message arrival rate and service rate at the related channels. These two measures can be computed based on the specific network topology property and routing algorithm.

Chapter 4

High Performance DAMQ Buffer Schemes

In this chapter three novel DAMQ buffer schemes based on self compacting buffer (SCB) are presented. These schemes are proposed to let traffic flow make an efficient use of input buffer that resides in each communicating node. For each of the three schemes, we first describe their organization; then present the simulation results and some concluding remarks.

4.1 DAMQ with Reserved Space for All Virtual Channels

DAMQ dynamically allocate buffer blocks according to the packet received. Compared with statically allocated buffer scheme, the advantage of DAMQ is that it uses efficiently the buffer space by applying free space to any incoming packet regardless its destination output port. Since there is no reserved space dedicated for each output channel, the packets destined to one specific output port may occupy the whole buffer space thus the packets destined to other output ports have no chance to get into the buffer. This is the case especially for small and compact routers with limited buffer space where wormhole switching technique and virtual channel mechanism are commonly used. A unidirectional virtual channel is implemented by an independently managed pair of buffers at two adjacent nodes.

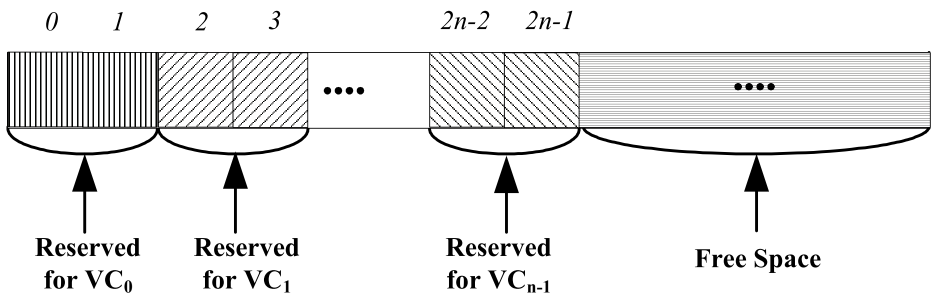


Figure 4.1: DAMQ$_{all}$ Buffer space at the initial state

When several virtual channels multiplex across the physical channel and share a common buffer, the virtual channels which have packets accepted in the buffer prior to other virtual channels may hold the whole buffer space when the output port to next hop node that it destines to is busy. In order to overcome this shortcoming of DAMQ buffer schemes, we implement a new buffer organization scheme, DAMQ with reserved space for all virtual channels (DAMQ$_{all}$). DAMQ$_{all}$ is based on the self-compacting buffer (SCB) scheme. It inherits most features of the SCB. Similarly, the virtual channels multiplexing one direction of a physical channel share a buffer. The new feature is that there is reserved space dedicated for each virtual channel, therefore at any time there is free space for the packets of "late" virtual channels which has not received packet and one virtual channel can never consume the whole buffer. As shown in Figure 4.1, two buffer slots are reserved for each virtual channel before the buffer accepts any incoming flit. The reserved spaces for each virtual channel are arranged sequentially according to the sequence numbers of the virtual channels.

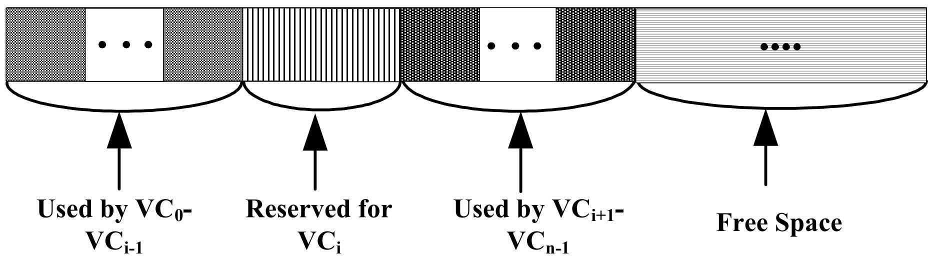


Figure 4.2: DAMQ*all* Buffer space status in operations

One register is used to point to the head of each reserved space, i.e. the head of the buffer region for each virtual channel. If two channels are denoted as V_i, V_j with $i + 1 = j$, then the reserved region for V_j will be placed right after the reserved region for V_i.

The size of reserved space for each virtual channel can be adjusted, however, we have chooses two flits because, according to our simulation experiments statistics, two reserved flits scheme yields satisfactory performance while keeps more free space for sharing. When there is an incoming flit to the buffer, the DAMQ*all* operates as shown in Figure 4.3. When a flit is leaving the buffer, the DAMQ*all* operates as illustrated in Figure 4.4:

. Also as shown in Figure 4.2, the reserved space for each virtual channel is always kept if there is no flit or only one flit in the buffer region for a specific virtual channel.

```
if (first flit for current VC){
  put it into buffer;
  increment counters;
}
else if
(current VC doesn't fill reserved space for it) or
(there is free slot left in buffer){
     if (last flit of current VC is next to first slot of next VC buffer space) {
       shift down all the flits and reserved space of the lower virtual channels one slot;
        increment head pointer for lower VCs;
    }
        put flit into buffer;
        increment counters;
}
```

Figure 4.3: Pseudo-code for in buffer operations

```
if (last flit in current VC){
  write flit to output port;
  decrement counters;
}
else{
  write flit to output port;
  decrement counters;
  shift up remaining flits of current VC;
  if (number of remaining flits of current VC >= reserved space number){
      shift up all the flits and reserved space of following VC one slot;
      decrement head pointer for following VC;
  }
}
```

Figure 4.4: Pseudo-code for out buffer operations

When the buffer performs shift up or shift down operations, the reserved spaces are treated same as the slots which are holding flits. Thus the order of the buffer space for virtual channels is kept conforming to the sequence of virtual channels. And once the number of current flits in buffer plus the number of reserved slots equals to the total amount of buffer slots, no more flit will be accepted unless this flit belongs to a virtual channel which has any reserved space available. Therefore, one or more virtual channels which have the flits come into the buffer at earlier time can never deprive the chance for other virtual channels which get flits later than them to get buffer. Moreover, once the earlier coming packets are blocked in the buffer, since there is still reserved space for other virtual channels, the network traffic will keep flowing, so the performance of the switch is also enhanced. This is the key improvement of DAMQ$_{all}$ scheme over SCB scheme.

4.2 DAMQ with Minimum Reserved Space for Virtual Channels

DAMQ$_{all}$ improved SCB by reserving buffer space for each virtual channel to avoid the situation that a few virtual channels consume the whole buffer then other virtual channels can not get buffer even when those virtual channels which get buffer are blocked. In the simulation experiments, we found that DAMQ$_{all}$ is not the most efficient way to reserve space for virtual channels. In a specific time interval, there may be no packets destined for some virtual channels. Even worse situation is that there may be no packets destined for some virtual channels for a very long time. In either case, the reserved spaces for these idle virtual channels are wasted. In order to reserve the buffer space more efficiently and provide more space for flowing traffic, we implement another buffer organization scheme, DAMQ with minimum reserved space for all virtual channels (DAMQ$_{min}$). DAMQ$_{min}$ is also based on SCB scheme and the virtual channels of one direction of a physical channel still share a buffer. And based on the simulation results, we still set the number of reserved buffer space to two slots. The difference to DAMQ$_{all}$ is that at any time there is at most one reserved space for all the virtual channels. And if every virtual channel have flit present in the buffer, no reserved space will be kept in the buffer anymore. Thus DAMQ$_{min}$ use minimum space for reserve purpose. As shown in Figure 4.5, two buffer slots are reserved for the virtual channel which may firstly claim for buffer.

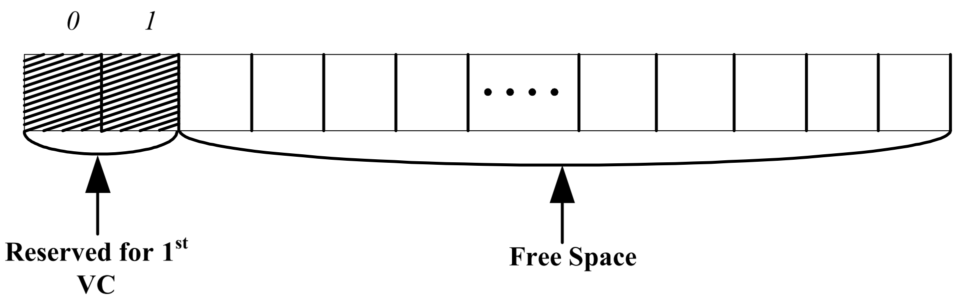


Figure 4.5: DAMQ$_{min}$ Buffer space status at the initial state.

Once a virtual channel has one flit come into the buffer, it will occupy two buffer slots which were reserved space before it comes in, thus there is actually still one slot reserved for it. As shown in Figure 4.8, this virtual channel will hold at least these two buffer slots unless it has no flit left in the buffer any more. Once every flits of a virtual channel moves out the buffer, the header pointer of this virtual channel will be reset to empty and there are no more buffer slots belong to it. Another two slots reserved buffer region may be created if possible. The operations that DAMQ$_{min}$ performs for incoming and leaving flit are shown in Figure 4.6 and 4.7 respectively.

```
if (first flit for current VC) and (there is a reserved space){
  put it into buffer;
  increment counters;
  set head pointer for current VC;
  if ( have enough free space in buffer ) and (not every VC are present)
    set the reserved space pointer to the slot next to current VC reserved space;
}
else if
   (current VC doesn't fill reserved space for it) or
   (there is free slot left in buffer){
     if (last flit of current VC is next to first slot of next VC buffer space) {
         shift down all the flits and reserved space of the lower virtual channels one slot;
         increment head pointers for lower VCs;
     }
     put flit into buffer;
     increment counters;
   }
```

4.6: Pseudo-code of in buffer operations for DAMQ$_{min}$

```
if (last flit in current VC){
  write flit to output port;
  decrement counters;
  reset head pointer for current VC;
}
else{
  write flit to output port;
  decrement counters;
  shift up remaining flits of current VC;
  if (number of remaining flits of current VC >= reserved space number){
      shift up all the flits and reserved space of following VC one slot;
      decrement head pointer for following VC;
   }
}
if ( no reserved space is present in buffer ) and (not every VC are present){
  if (there are enough free buffer slots)
    set the reserved space pointer to the slot next to last VC reserved space;
}
```

Figure 4.7: Pseudo-code of out buffer operations for DAMQ_{min}

Reserved space is always placed right after the buffer region of virtual channel which is the latest one to have flit into the buffer. When the buffer performs shift up or shift down operations, all reserved slots are treated same as the slots which are holding flits.

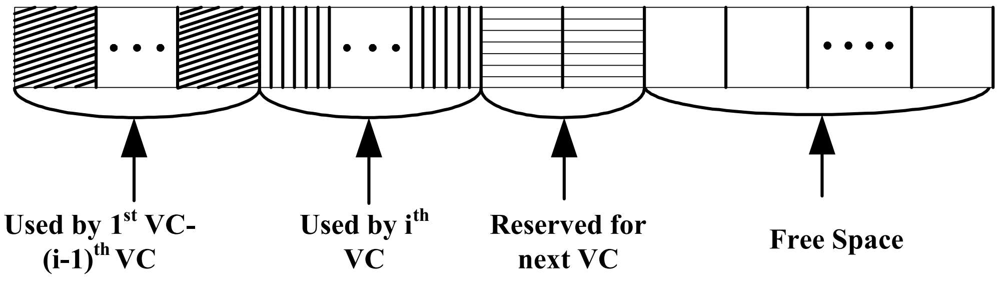


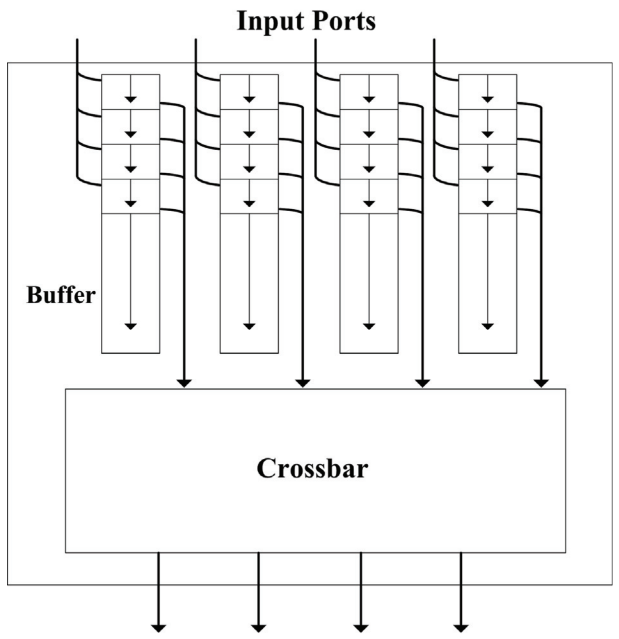
Figure 4.8: DAMQ$_{min}$ Buffer space status in operations.

By dynamically creating reserved space for virtual channels, DAMQ$_{min}$ presents a very efficient way to use buffer space, there is always minimum buffer space used for reserve purpose, so there are more free space available for flowing traffic.

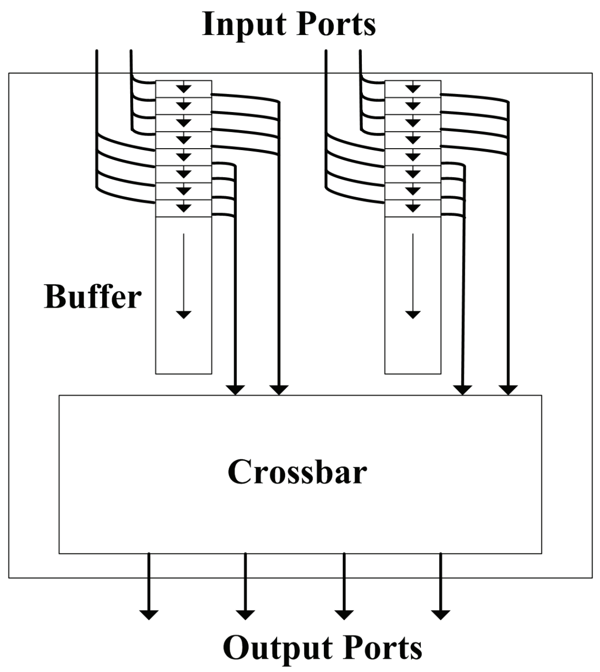
4.3 Shared DAMQ with Reserved Space for All Virtual Channels

DAMQ allocates buffer space when a packet is received. Compared with statically allocated multi-queue (SAMQ) scheme, the advantage of DAMQ is its efficient use of the buffer space by allocating free space to an incoming packet regardless of its destination output port. However, because there is no reserved space dedicated for each output channel, the packets destined to one specific output port may occupy the whole buffer space thus the packets destined to other output ports have no chance to get into the buffer. This is the case especially for small and compact routers with limited buffer space where wormhole switching technique and virtual channel mechanism are used. In order to overcome this shortcoming a new buffer scheme, DAMQ with reserved space for all virtual channels (DAMQ$_{all}$) was proposed in [43], DAMQ$_{all}$ is based on Self-compacting buffer (SCB) scheme, the virtual channels belonging to one direction of a bidirectional physical channel share a buffer as described in previous sections of this chapter.

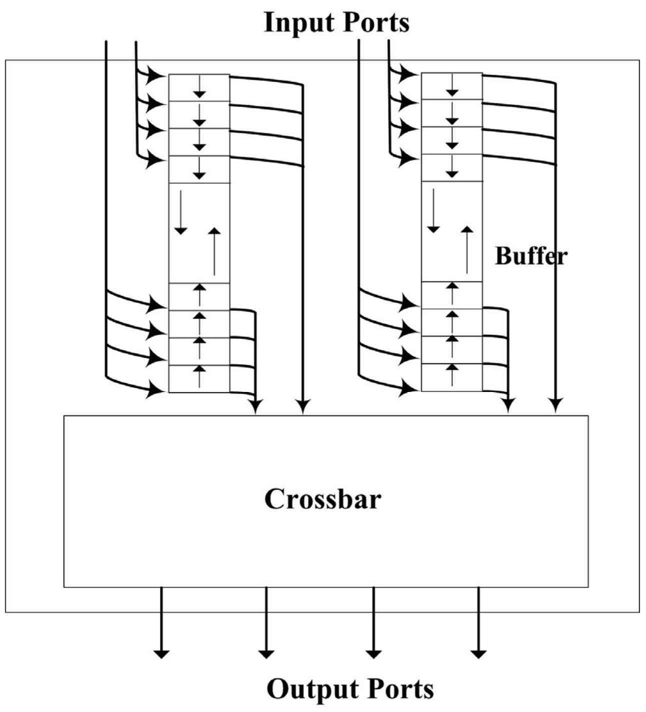
However, in a wormhole-switched network with several virtual channels multiplexing a physical channel, some routing algorithms, for example, the algorithms that pick an available virtual channel sequentially tend to choose one set of virtual channels over others; moreover, even the virtual channels are chosen randomly, the traffic may not evenly distributed into all virtual channels of different physical channel, thus the traffic load usually is not evenly distributed in buffer space of a physical channel and among different physical channels. Therefore, a more efficient approach to use the available buffer space is to let the virtual channels belonging to a physical channel share buffer with virtual channels of another physical channel.



(a) DAMQ$_{all}$ Buffers



(b) DAMQ$_{shared\text{-}1}$ Buffers



(c) DAMQ$_{shared\text{-}2}$ Buffers

Figure 4.9: Switches with DAMQ$_{all}$ buffer and DAMQ$_{shared}$ buffer

As shown in Figure 4.9. (a), the simple switch with four input and four output ports adopts DAMQ$_{all}$ buffer scheme for the input buffer; there is one dedicated buffer per physical channel, i.e. east X, west X, north Y and south Y. Each physical channel buffer has its own read port and write port, the four virtual channels that are multiplexing a physical channel have their own reserved space (RS) in buffer. Our new DAMQ$_{shared}$ buffer combines the buffer for virtual channels from two different physical channels. We combine the buffer space for east X and south Y virtual channels to build one physical buffer, and west X and north Y forms another buffer group. As shown in Figure 4.9. (b) and (c), there are two buffers for four physical channels; each buffer is shared by eight virtual channels, and has two read ports and write ports respectively.

We used two ways to organize the shared space of DAMQ$_{shared}$. The first way, DAMQ$_{shared\text{-}1}$, behaves similarly as DAMQ$_{all}$, the difference is the number of virtual channels sharing buffer is doubled in this case. In the second way, DAMQ$_{shared\text{-}2}$, the shared space is placed in the middle of the two buffer regions of two virtual link groups then the two buffer regions expand towards center of the free buffer space. This way, there will be less data shifts when a flit is saved into buffer because the movement of one region doesn't depend on shift of another group and it closely simulates the behavior of hardware which is shown in Figure 4.10.

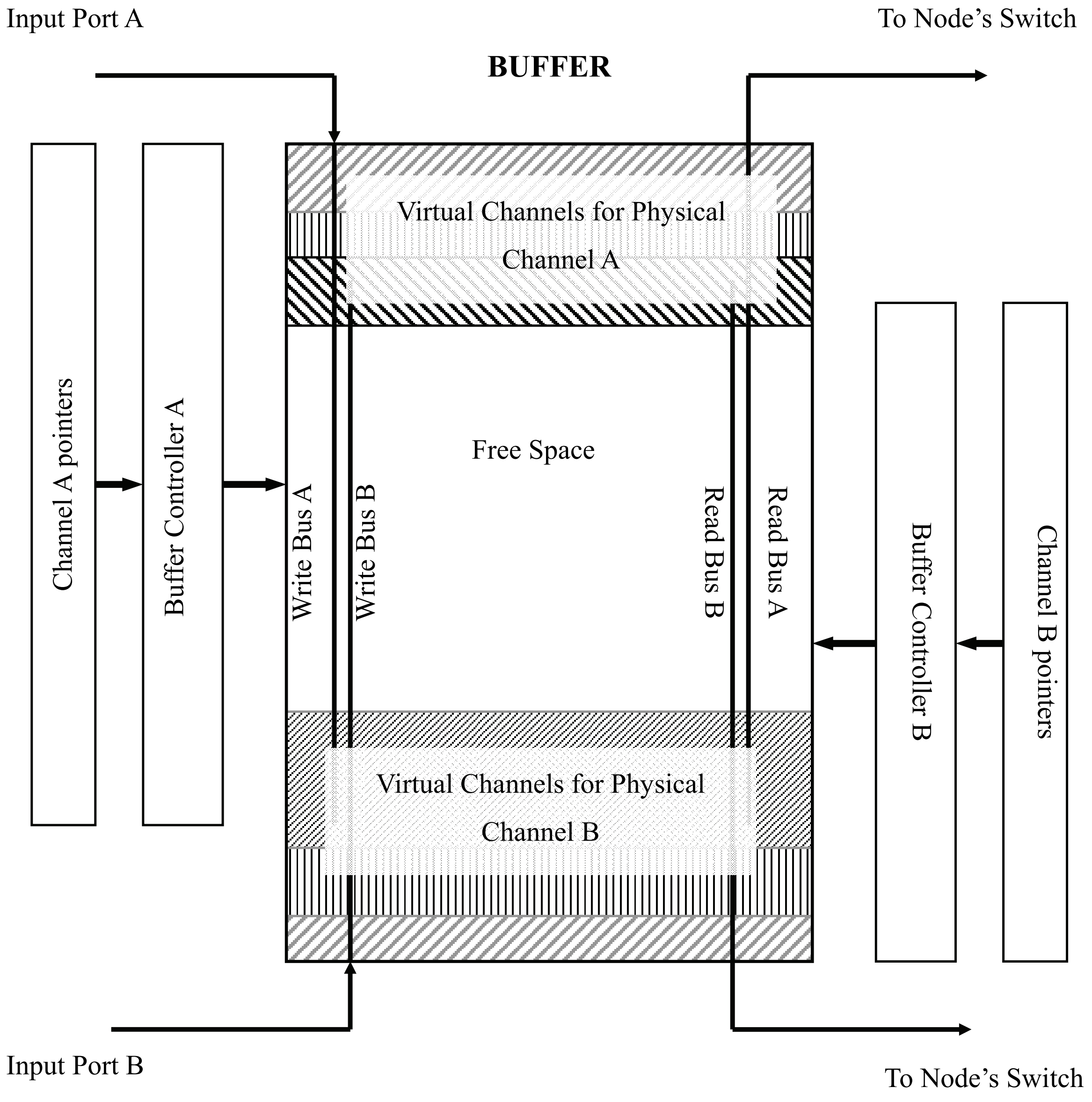
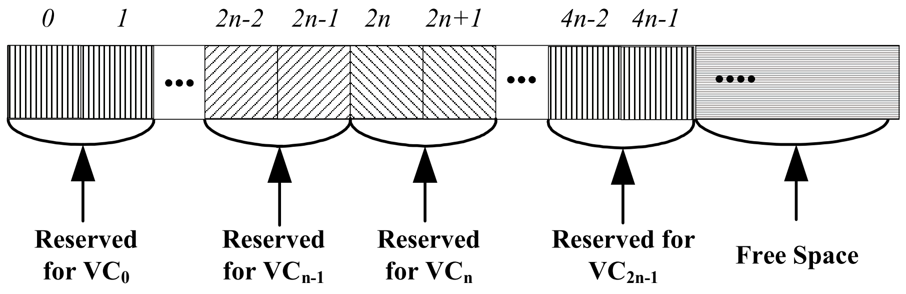


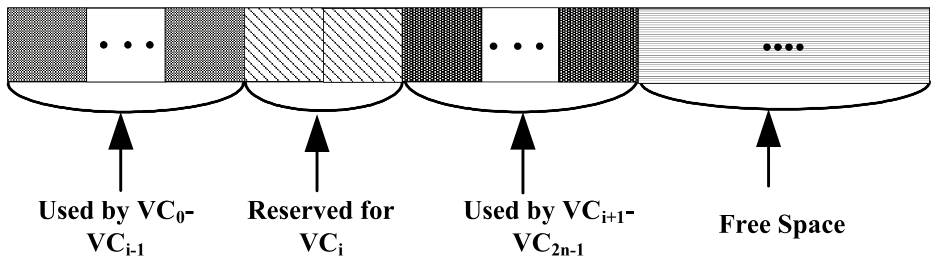
Figure 4.10: DAMQ$_{shared}$ buffer organization

As illustrated in Figure 4.11, two buffer slots are reserved for each virtual channel before any flit comes in the buffer. In DAMQ$_{shared\text{-}1}$, the reserved spaces for each virtual channel are arranged sequentially according to the sequence numbers of the virtual channels. As shown in Fig 4.11 (a), virtual channels on the X dimension have smaller sequence numbers than those on the Y dimension; the first virtual channel on Y dimension is contiguous to the last one on X dimension. One register is used to point to the head of each reserved space, i.e. the head of the buffer region for each virtual channel. If two channels are denoted as V_i, V_j with $i + 1 = j$, then the reserved region for V_j will be placed right after the reserved region for V_i. The reserved space for virtual channels on Y dimension is right after the reserved space for X virtual channels. The reserved space for each virtual channel is always kept if there is no flit or only one flit in the buffer region for a specific virtual channel.

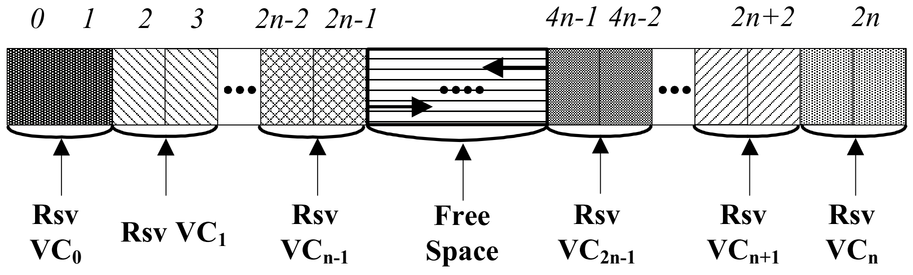
When the buffer performs shift up or shift down operations, the reserved spaces are treated same as the slots which are holding flits. Thus the order of the buffer space for virtual channels is kept conforming to the sequence of virtual channels. Moreover, since two groups of virtual channels are sharing buffer and the reserved spaces for virtual channels from two groups are arranged continuously, if there is no space in the region of first group for an incoming flit headed to a virtual channel in this group, and there is still space in the shared region, then the space will be made by shifting down all the lower addressed regions including space of this virtual channel, space of other virtual channels in this group and space of another virtual channels group. The buffer state when it is in operation is shown in Figure 4.11 (b).



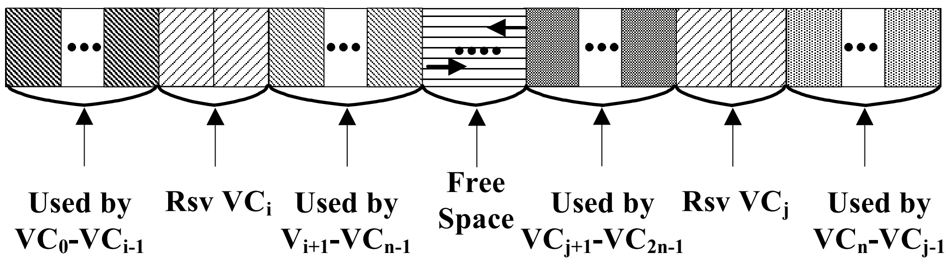
(a) DAMQ$_{shared\text{-}1}$ Buffer space status at initial state.



(b) DAMQ$_{shared\text{-}1}$ Buffer space status in operations.



(c) DAMQ$_{shared\text{-}2}$ Buffer space status at initial state


Used by VC_0-VC_{i-1}
Rsv VC_i
Used by V_{i+1}-VC_{n-1}
Free Space
Used by VC_{j+1}-VC_{2n-1}
Rsv VC_j
Used by VC_n-VC_{j-1}

(d) DAMQ$_{shared\text{-}2}$ Buffer space status in operations

Figure 4.11: DAMQ$_{shared}$ buffer statuses in initial state and operation

For DAMQ$_{shared\text{-}2}$, when the buffer is in operation, the RS (reserved space) is also always kept if there is no flit or only one flit in the buffer region for a specific virtual channel. As shown in Figure 4.11(d), same as previously mentioned schemes, when the buffer performs shift up or shift down operations, the RSs are also shifted. When a virtual channel accepts a flit, it first uses its RS. If RS is used up, buffer space of the lower addressed region in this group expands toward another group's buffer space to produce a slot. Once the boundaries of the two buffer regions encounter, no more flit will be accepted unless this flit goes to a virtual channel which has its own RS available. At any time during operation, the number of current flits in buffer plus the number of reserved slots equals to the total amount of buffer slots. Therefore, one or more virtual channels which have the flits come into the buffer at earlier time can never deprive the chance for other virtual channels which get flits later than them to get buffer. Also, in case the earlier coming packets are blocked in the buffer, since there is still reserved space for other virtual channels, the network traffic through these channels can still keep flowing; therefore the performance of the switch and the whole network can be enhanced.

Moreover, as virtual channels from two physical channels are sharing the buffer, the buffer space is more efficiently used by the incoming flits. Hence, to achieve same network performance, by using DAMQ$_{shared}$ scheme, a switch can use less buffer space than DAMQ$_{all}$ and traditional buffer schemes. The results will be shown in next section.

4.4 Performance Evaluation

This section presents the results of simulation experiments conducted to evaluate the performance of our novel buffer organization schemes proposed in previous sections. First, our methodology and configuration of simulation environment are described. Then we examine the performance of DAMQ$_{all}$, DAMQ$_{min}$, and DAMQ$_{shared}$ in greater detail.

4.4.1 Simulation Experiments Setup

We have carried out our simulations by using flexsim1.2 [18] which is a simulator for flit-level simulation of torus/mesh networks as introduced in Chapter 3. To compare the performance of DAMQ$_{shared}$ with other schemes including DAMQ$_{all}$, we conducted exhaustive simulations on 16-ary 2-cube, 8-ary 2-cube and 4-ary 2-cube message exchanging systems with wrapped around channels as shown in Figure 4.12. For DAMQ$_{all}$ and DAMQ$_{min}$, the architecture we used to conduct simulations is a 4-ary 2-cube message exchanging system.

In the simulated network system, a switch is attached to each end-node which has one injection channel to the switch and one input channel to receive message from network. Physical channels are duplex channels. To thoroughly examine DAMQ$_{shared}$ scheme and get in depth understanding of how switch buffer schemes impact network performance, two kinds of traffic, uniformly distributed and hotspots traffic are simulated.

To evaluate DAMQ$_{all}$ and DAMQ$_{min}$ schemes, we used uniform traffic as many other researchers also use this traffic mode in their NoC works [31] [42]. When using uniformly distributed network traffic, every end node generates packets with randomly determined destination and injects them into the network. When using hotspot traffic, we randomly pick four hotspots in a 64 nodes 8-ary 2-cube network. In addition, we make these hotspots nodes reside in different row and column of the network to avoid intensively congested region. Other pertinent simulation configuration parameters are listed as follows:

- Routing flit delay is set to 1 cycle.
- Data flit delay is set to 1 cycle.
- Buffers at local end nodes are infinite.
- Packets size is set to 32 flits.
- Switching technique used is wormhole.

As to the routing algorithms used in our simulations, we adopt a static routing protocol E-Cube [12] to conduct the simulations for both DAMQ_{all} and DAMQ_{min}. Adaptive routing protocol is used to conduct our simulations for DAMQ_{shared} as several researchers had reported strong performance brought by adaptive routing [23]. Among the up to date adaptive routing algorithms, we choose Duato's routing methodology in our experiments because it's a well known and extensively used adaptive routing protocol. Dimension order path selection function is used for the Duato's routing protocol. We vary the applied load, buffer size and virtual channels number; study their impact on throughput and message latency for the network. To increase the number of virtual channels multiplexing a physical channel can improve switch performance, but having too many virtual channels not only incurs expensive hardware expense but also increases the message delay. In our simulation experiments, we set the number of virtual channels of one direction of a duplex physical channel to four or eight, thus the advantages brought by virtual channel mechanism will not be overshadowed by its shortcoming.

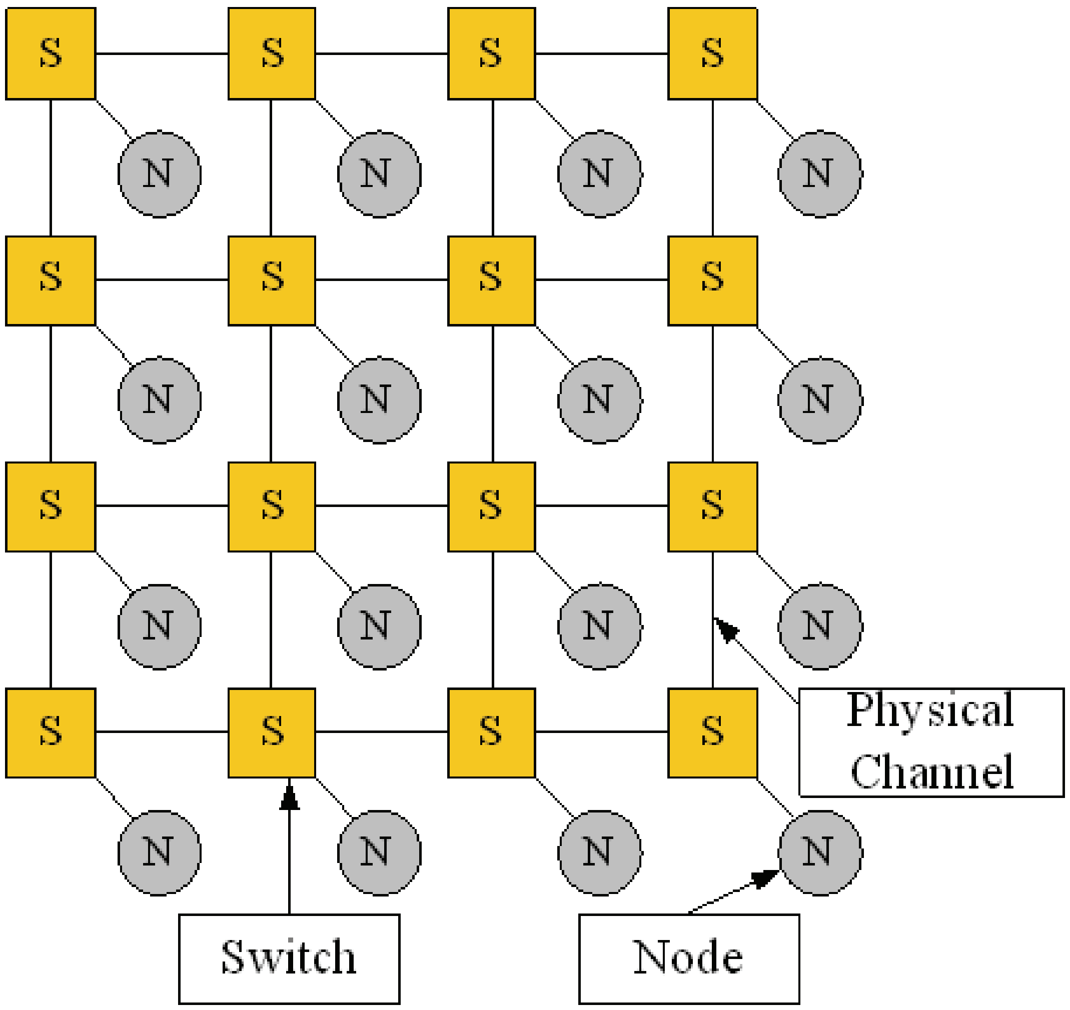


Figure 4.12: The base system for our simulations

4.4.2 Examined Performance Metrics

Since messages are divided to flits when transmitting, to increase message length has the similar effect on increasing traffic load as to shorten the average injection period for each node. We set the message length at fixed 32 flits and make the network into saturation state by shortening injection period for each node namely by increasing the traffic load rate. Traffic load rate is derived by the following formula [18]:

$$LR \times Ch = N \times (ML / IP) \times Dst$$

Where *LR* is load rate, *CH* is the number of channels, *N* is the number of nodes, *ML* is message length, *IP* is average injection period, and *DST* is average routing distance.

We compare our new buffer scheme DAMQ$_{shared}$ to DAMQ$_{all}$ [43] and the traditional statically allocated buffer scheme (SAMQ) used for virtual channels which is reported in [44]. We didn't include traditional DAMQ scheme in the comparisons, because during the simulation process we found the whole buffer space for one port is easily occupied by the blocked messages which incur deadlocks, when the network becomes congested.

Two most important metrics, network throughput and message latency are compared among these three buffer schemes. The network throughput (*TP*) is defined as the number of flits received per node per cycle as follows:

$$TP = \frac{MSGN \times ML}{N \times ST}$$

Where *MSGN* is the total number of delivered messages; *ML* is the message length, *N* is the total number of nodes and *ST* is the simulation time (in cycles).

The message latency is measured as the average time span (in cycles) for every packet between the moment it was generated and the reception of the whole packet at destination. We use average latency (LTN_{avg}) of all the injected messages as the performance metric in our simulations. This latency is defined as follows:

$$LTN_{avg} = \frac{1}{MN} \times \sum_{i=1}^{MN} LTN_i$$

Where LTN_i is latency for $Message_i$ and *MN* is the total injected message number.

In addition, when evaluating DAMQ$_{shared}$ scheme, we compared buffer utilizations among these buffer schemes to get an in-depth understanding of buffer usage efficiency. We use average stored flits ($FLIT_{avg}$) in the buffers of all the nodes as a metric in our simulations. It is defined in the following formula:

$$FLIT_{avg} = \frac{1}{ST} \times \sum_{1}^{ST} \sum_{i=1}^{VN} FLIT_i$$

Where $FLIT_{avg}$ is the average number of flits that are stored in the buffer space of all the nodes, $FLIT_i$ is the number of stored flits in *VirtualChannel$_i$'s* buffer space and *VN* is the total virtual channel count in the network.

4.4.3 Simulation Results

DAMQ$_{all}$ and DAMQ$_{min}$ on four virtual channels and 16 nodes In this part we present the simulation results obtained when 4 virtual channels are multiplexing a physical channel network is 16 nodes torus. We examine the performance for three different size buffers on SAMQ and two different size buffers on DAMQ$_{all}$ and DAMQ$_{min}$, The throughput and message latency are shown in Tables 4.1 and 4.2, respectively. It is well known that there is no significant difference for different buffer scheme while the network is not saturated. In our simulation experiments, we keep increasing the traffic load so that we can compare the performance of different buffer schemes when the network is in saturation status. As shown in Figure 4.13, along with the network saturation process, our DAMQ$_{all}$ and DAMQ$_{min}$ have higher throughputs than SAMQ if they all use same size buffer. DAMQ$_{all}$ achieves approximate the same maximum throughput as SAMQ. However it only uses half of the buffer space used by SAMQ. DAMQ$_{min}$ gets even higher maximum throughput than DAMQ$_{all}$, because the former saves more buffer slots ready to be used than the latter. Furthermore, at the same maximum throughput level, both DAMQ$_{all}$ and

DAMQ$_{min}$ have less latency than SAMQ as shown in Figure 4.14.

DAMQ$_{all}$ and DAMQ$_{min}$ on eight virtual channels and 16 nodes We present the simulation results obtained when 8 virtual channels are multiplexing a physical channel in this section and network is 16 nodes torus. Because more virtual channels are used, the throughput and latency are higher than 4 virtual channels situation when the network is saturated. The throughput and message latency are shown in Table 4.3 and Table 4.4 respectively. DAMQ$_{all}$ and DAMQ$_{min}$ again get better performance over SAMQ. As shown in Figure 4.15, with only half size buffer, DAMQ$_{min}$ even get higher maximum throughput than SAMQ and DAMQ$_{all}$ get same throughput as SAMQ. The reason is when there are more virtual channels involved, DAMQ$_{min}$ and DAMQ$_{min}$ can have more free buffer space to use and DAMQ$_{min}$ make the most efficient usage on the buffer. Also as shown in Figure 4.16, there is no unwanted message latency introduced for DAMQ$_{all}$ and DAMQ$_{min}$, they have approximate same latency as SAMQ that are using double size buffer.

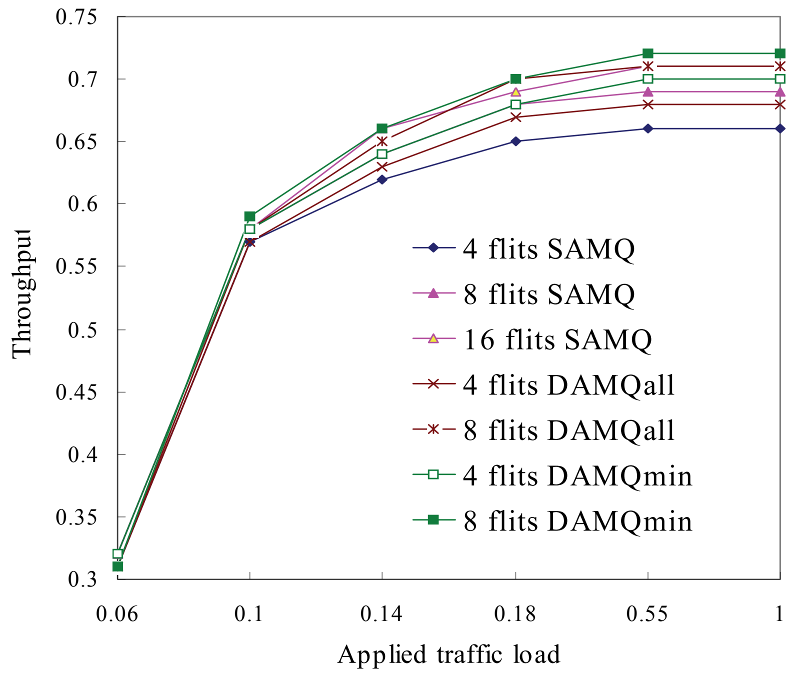


Figure 4.13: Comparison on throughput between 4/ 8 flit-buffer DAMQall / DAMQmin and

4/ 8/ 16 flit-buffer SAMQ

Table 4.1

The Performance 4-ary, 2-cube network composed of block switches. Shown is the throughput obtained from simulations where 4 virtual channels used

.

Buffer Type	BS per VC	Throughput Versus Applied Traffic Load Rate						
		0.02	0.06	0.10	0.14	0.18	0.55	1
SAMQ	4	0.16	0.31	0.57	0.62	0.65	0.66	0.66
	8	0.15	0.32	0.58	0.64	0.68	0.69	0.69
	16	0.15	0.32	0.58	0.66	0.69	0.71	0.71
DAMQ $_{all}$	4	0.16	0.31	0.57	0.63	0.67	0.68	0.68
	8	0.16	0.31	0.58	0.65	0.70	0.71	0.71
DAMQ$_{min}$	4	0.16	0.32	0.58	0.64	0.68	0.69	0.69
	8	0.16	0.31	0.59	0.66	0.70	0.72	0.72

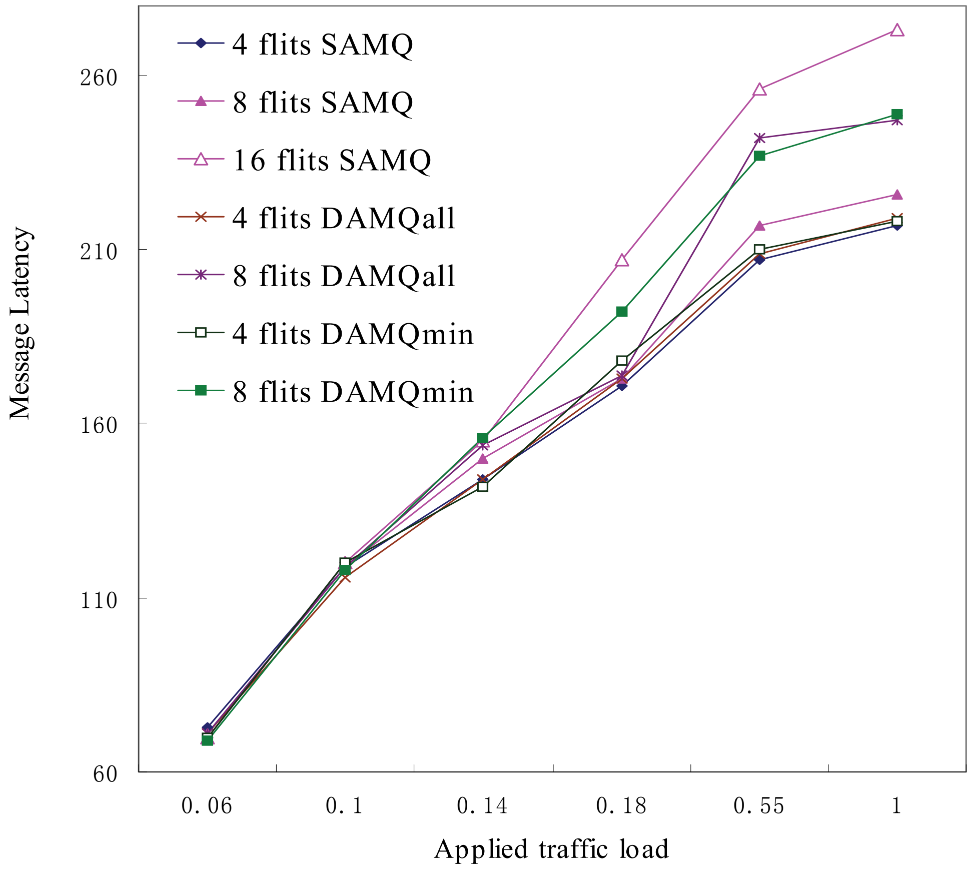


Figure 4.14: Comparison on Latency between 4/ 8 flit-buffer DAMQ$_{all}$ / DAMQ$_{min}$ and 4/ 8/ 16 flit-buffer SAMQ

Table 4.2

The Performance 4-ary, 2-cube network composed of block switches. shown is the message latency obtained from simulations where 4 virtual channels used.

Buffer Type	BS per VC	Message Latency Versus Applied Traffic Load Rate						
		0.02	0.06	0.10	0.14	0.18	0.55	1
SAMQ	4	54	73	119	144	171	207	217
	8	53	71	119	150	173	217	226
	16	53	70	120	155	207	256	273
DAMQ$_{all}$	4	55	71	116	144	173	209	219
	8	54	71	119	154	174	242	247
DAMQ$_{min}$	4	53	70	120	142	178	210	218
	8	54	69	118	156	192	237	249

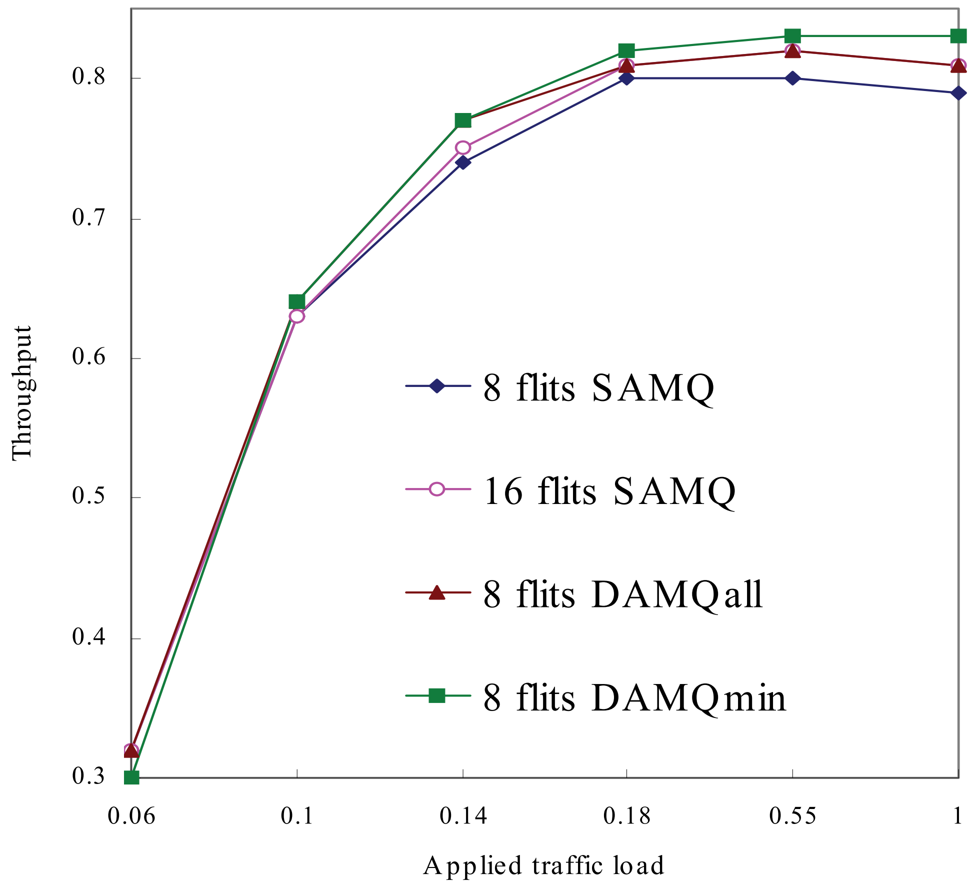


Figure 4.15: Comparison on Throughput between 8 flit buffer DAMQ$_{all}$ / DAMQ$_{min}$ and 8/ 16 flit-buffer SAMQ

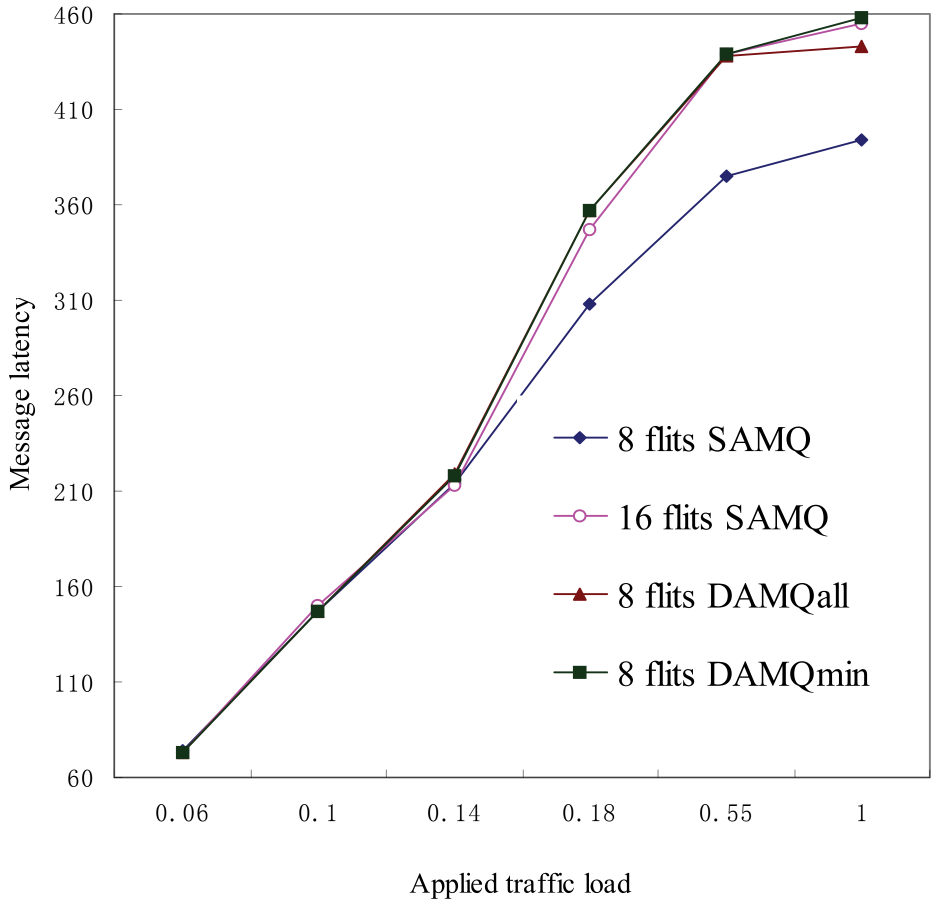


Figure 4.16: Comparison on Latency between 8 flit-buffer DAMQ$_{all}$ / DAMQ$_{min}$ and 8/ 16 flit-buffer SAMQ

Table 4.3

The Performance 4-ary, 2-cube network composed of block switches. Shown is the throughput obtained from simulation where 8 virtual channels used..

Buffer Type	BS per VC	Throughput Versus Applied Traffic Load Rate						
		0.02	0.06	0.10	0.14	0.18	0.55	1
SAMQ	8	0.16	0.32	0.63	0.74	0.80	0.80	0.79
	16	0.16	0.32	0.63	0.75	0.81	0.82	0.81
DAMQ$_{all}$	8	0.16	0.32	0.64	0.77	0.81	0.82	0.81
DAMQ$_{min}$	8	0.16	0.30	0.64	0.77	0.82	0.83	0.83

DAMQ$_{shared}$ on four virtual channels, 64 nodes and uniform traffic. Because of the hardware constrains for network on chip systems, a single buffer in NoC systems usually does not hold a full message. We set the buffer size for each virtual channel to 4 flits when DAMQ$_{all}$ and SAMQ are used. Since four virtual channels are multiplexing cross one physical channel, the buffer size for each direction of a duplex physical channel is 16 flits when these two buffer schemes are evaluated. To examine the performance of DAMQ$_{shared}$ with regard to the relationship between buffer size and performance, we use six different size buffers from 11 to 16 flits buffers for each direction of a duplex physical channel.

Table 4.4

The Performance 4-ary, 2-cube network composed of block switches. Shown is the message latency obtained from simulations where 8 virtual channels used

Buffer Type	BS per VC	Message Latency Versus Applied Traffic Load Rate						
		0.02	0.06	0.10	0.14	0.18	0.55	1
SAMQ	8	55	74	147	214	308	375	394
	16	54	73	150	213	347	439	455
DAMQ$_{all}$	8	55	73	147	219	357	438	443
DAMQ$_{min}$	8	54	73	147	218	357	439	458

The simulation results for throughput and message latency are shown in Tables 4.5 and 4.6, respectively. We compare the performance of different buffer schemes when the network starts to saturate on about 0.28 traffic load until it gets saturated after about 0.4 traffic load is applied. As shown in Figure 4.17, along with the network saturation process, our new DAMQ$_{shared}$ has higher throughput than both DAMQ$_{all}$ and SAMQ when they all use same size of 16 flits buffer. When the network gets saturated, DAMQ$_{shared}$ achieves the highest throughput with same size 16-flit buffer is used. Furthermore, 12-flit DAMQ$_{shared}$ achieves approximately the same maximum throughput as SAMQ using 16 flits buffer as shown in Figure 4.17. Also, 14-flit DAMQ$_{shared}$ achieves approximately the same maximum throughput as SAMQ using 16 flits buffer. In sum, DAMQ$_{shared}$ achieves best performance among the three buffer schemes we tested.

DAMQ$_{shared}$ tends to provide a more efficient method for flits to share buffer space than DAMQ$_{all}$ which has already shown advantages over traditional SAMQ scheme.

DAMQ$_{shared}$ achieves the best performance among the three buffer schemes we tested. DAMQ$_{shared}$ tends to provide a more efficient method for flits to share buffer space than DAMQ$_{all}$ which has already shown advantages over traditional SAMQ scheme.

As to the message latency, DAMQ$_{shared}$ managed to hold a similar latency as SAMQ until the network is severely saturated after a load of about 0.36 is applied, as shown in Figure 4.18. When we further increase the traffic load after the network starts getting saturated, DAMQ$_{shared}$ shows higher latency than both DAMQ$_{all}$ and SAMQ. The reason is DAMQ$_{shared}$ holds much more flits in the buffer than other schemes.

The numbers of average flits that are stored in the buffer are presented in Table 4.7. The total buffer space (BUF_{total}) can be obtained by the following formula:

$$BUF_{total} = N \times PHY \times 2 \times VC \times VB$$

Where *PHY* is the physical channel corresponding to a node port, *VC* is the count of virtual channel multiplexing a physical channel and *VB* is the buffer size of a virtual channel. The total available buffer space is 4096 flits in our simulations. Thus, under traffic load 0.38, DAMQ$_{shared}$ utilizes 65% of the whole buffer space while DAMQ$_{all}$ and SAMQ uses 51% and 37% respectively as shown in Figure 4.19. Because message latency is a time span average on all the flits that are injected into the network, more flits stored in the buffer results in longer time for them in the waiting queues as the network has become drastically saturated.

It has been shown that DAMQ$_{shared}$ provides a better use of the buffer space. In addition, it should be pointed out that a 12-flit DAMQ$_{shared}$ buffer achieves approximately the same maximum throughput as a 16-flit SAMQ buffer as shown in Figure 4.17. Also, a 14-flit DAMQ*shared* achieves approximately the same maximum throughput as 16-flit DAMQ*all* buffer. This is to say, to provide a similar network performance on very limited buffer resource, DAMQ*shared* achieves similar throughput with 25% and 12.5% less buffer space than SAMQ and DAMQ*all*, respectively. And the control units overhead is negligible as mentioned in Section 4.3.

Table 4.5

The Performance of 8-Ary, 2-cube Network Composed Of Block Switches. Shown is the Throughput Obtained for Simulations Where 4 Virtual Channels per Physical Channel Used When Uniform Traffic is applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.20	*.23*	*.29*	*0.31*	*.33*	*.35*	*.38*
SAMQ	*16*	.391	.442	.530	.548	.556	.558	.555
DAMQ *all*	*16*	.391	.440	.541	.562	.571	.570	.565
DAMQ*shared*	*11*	.392	.447	.535	.553	.554	.553	.542
	12	.392	.450	.539	.558	.562	.560	.546
	13	.391	.449	.540	.564	.568	.565	.548
	14	.392	.447	.541	.565	.575	.570	.552
	15	.392	.449	.542	.571	.578	.576	.568
	16	.392	.449	.541	.573	.585	.585	.575

Table 4.6

The Performance of 8-ary, 2-cube Network Composed of Block Switches. Shown is the Message Latency Obtained from Simulations Where 4 Virtual Channels Per Physical Channel Used When Uniform Traffic is Applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.20	*.23*	*.29*	*.31*	*.33*	*.35*	*.38*
SAMQ	*16*	103	113	140	157	168	179	199
DAMQ $_{all}$	*16*	102	110	142	158	175	184	198
DAMQ$_{shared}$	*11*	103	117	151	169	190	200	223
	12	104	116	149	171	186	201	228
	13	103	116	149	165	184	203	235
	14	103	114	145	164	183	205	238
	15	103	113	145	164	185	203	240
	16	103	113	143	158	180	205	244

Table 4.7

The Performance of 8-ary, 2-cube Network Composed of Block Switches. Shown is the Buffer Usage Obtained from Simulations Where 4 Virtual Channels Per Physical Channel Used When Uniform Traffic is Applied.

Buffer Type	BS per PC	Applied Traffic Load Rate						
		.20	*.23*	*.29*	*.31*	*.33*	*.35*	*.38*
SAMQ	*16*	356	493	873	1053	1224	1324	1501
DAMQ $_{all}$	*16*	368	543	1019	1291	1611	1798	2096
DAMQ$_{shared}$	*11*	446	592	1020	1223	1341	1462	1537
	12	452	626	1086	1305	1548	1651	1793
	13	461	672	1156	1417	1675	1857	2024
	14	501	662	1201	1541	1864	2063	2262
	15	491	689	1224	1619	1931	2172	2489
	16	501	685	1219	1618	2002	2239	2669

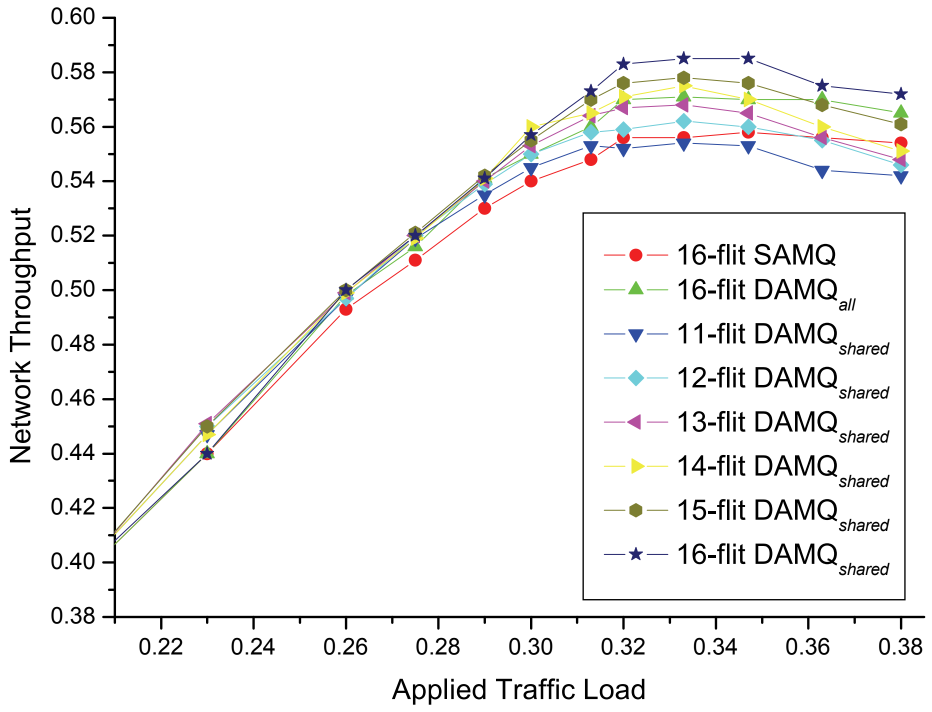


Figure 4.17. Comparison on throughput between 11-16 flit buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 16 flit-buffer SAMQ under uniform traffic.

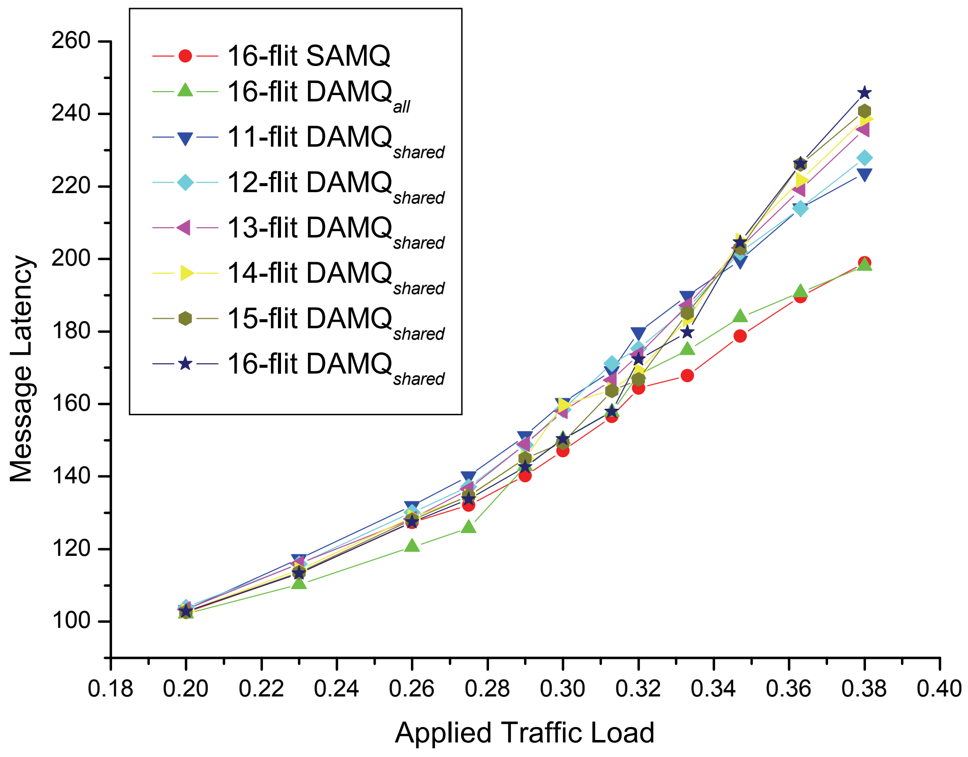


Figure 4.18: Comparison on latency between 11-16 flit-buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 16 flit-buffer SAMQ under uniform traffic.

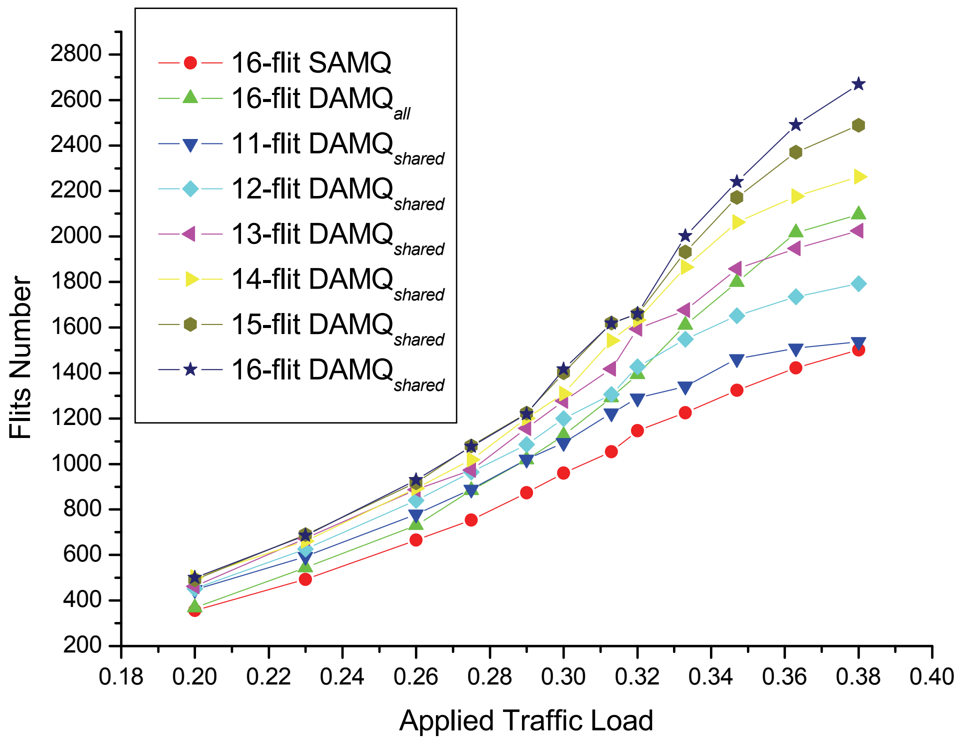


Figure 4.19: Comparison on buffer usages between 11-16 flit-buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 16 flit-buffer SAMQ under uniform traffic

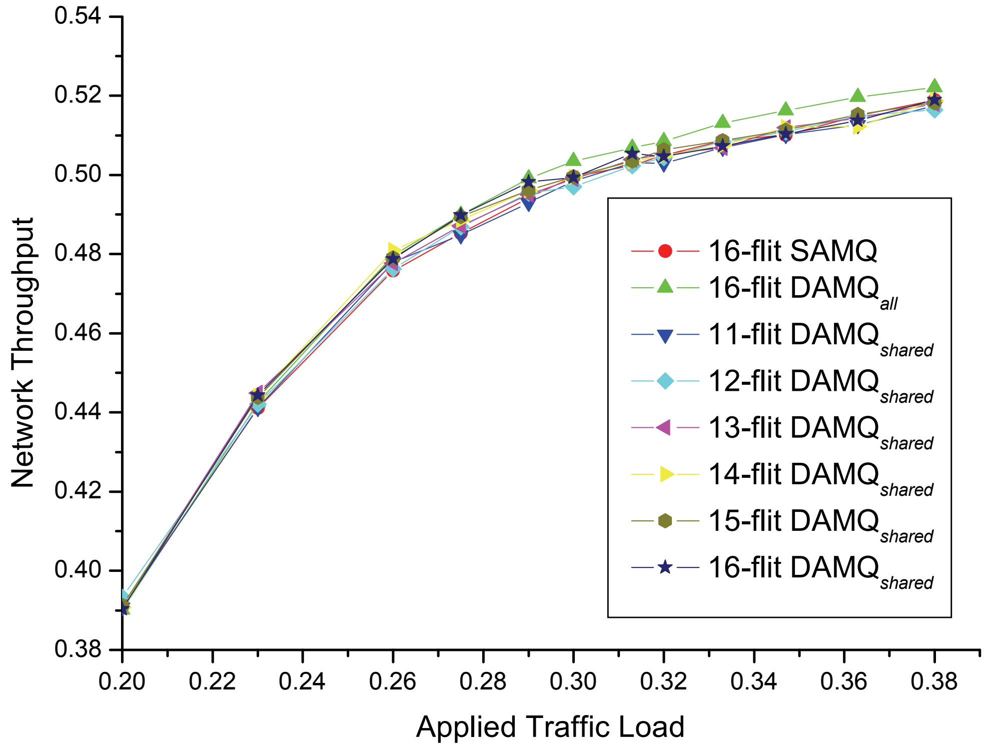


Figure 4.20: Comparison on throughput between 11-16 flit-buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 12/16 flit-buffer SAMQ under Hotspots traffic

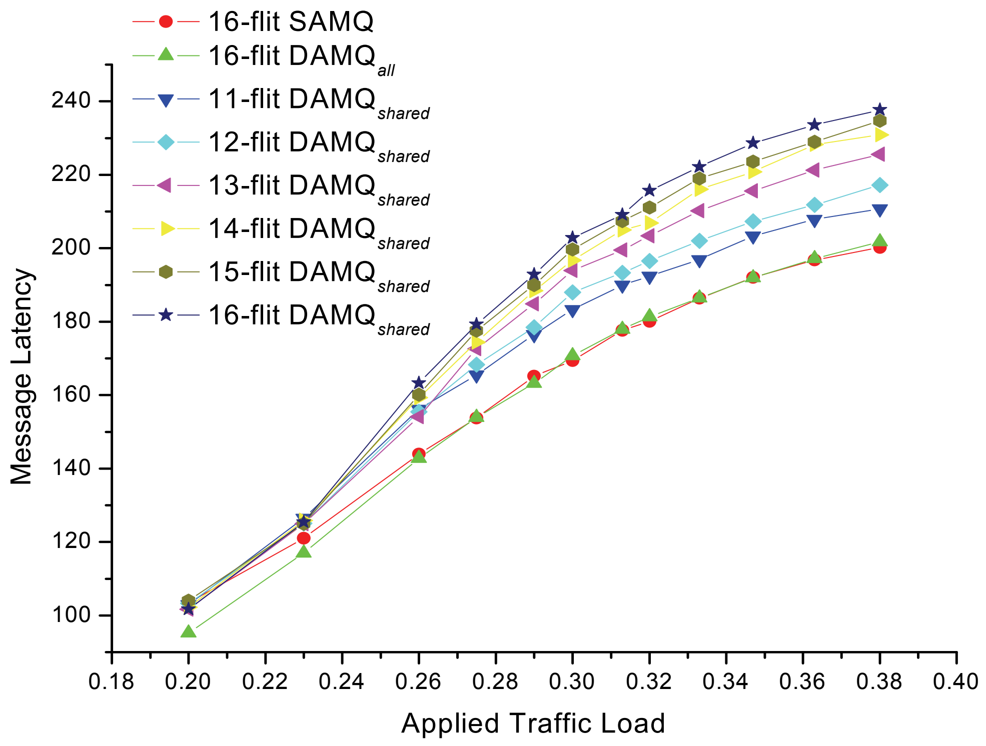


Figure 4.21: Comparison on latency between 11-16 flit-buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 12/ 16 flit-buffer SAMQ under hotspots traffic.

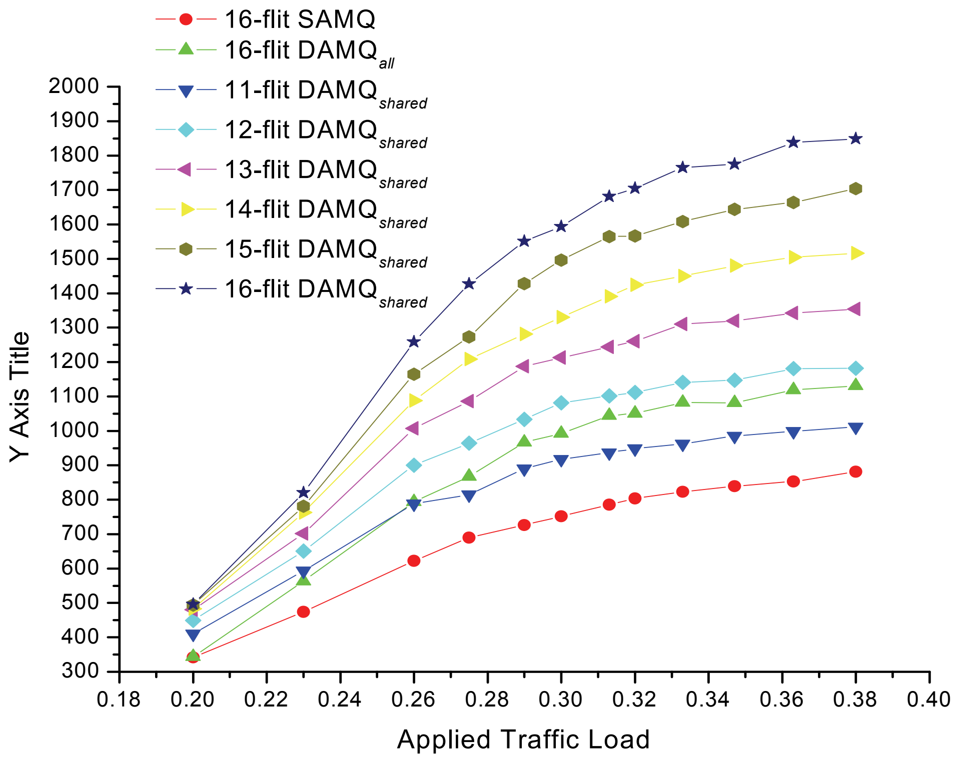


Figure 4.22: Comparison on buffer usages between 11-16 flit-buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 12/16 flit-buffer SAMQ under Hotspots traffic

$DAMQ_{shared}$ on four virtual channels, 64 nodes and hotspots traffic As motioned earlier, four hot spots are randomly generated in the 64 nodes network when we conduct the simulations on hotspots traffic. After we got the hot spots locations, they are fixed for all the simulations on different buffer schemes. Five percent of the traffic is directed to the hot spots. Other network configurations are same as simulations for uniform traffic.

The simulation results for throughput and message latency are shown in Tables 4.8 and 4.9, respectively. As shown in Figure 4.20, because hot spots become the bottlenecks for the entire network, we can observe all the three buffer schemes provides similar throughput along with the network's saturating process. Also we can find that buffer space doesn't play an important role in this scenario; there is no significant throughput difference between 12-flit and 16-flit SAMQ. The results on different sized $DAMQ_{shared}$ buffer are very similar as well. This is because when network gets saturated, hot spots become bottlenecks and they are the determining factor for the whole network throughput. As shown in Figure 4.21, message latency on $DAMQ_{shared}$ is about twenty and ten percent higher than same sized SAMQ and $DAMQ_{all}$ respectively.

Moreover, the message latency is proportional to the buffer size when $DAMQ_{shared}$ are used. 11-flit $DAMQ_{shared}$ buffer has very close message latency to $DAMQ_{all}$ and SAMQ; larger sized $DAMQ_{shared}$ buffer has greater latency while 16-flit $DAMQ_{shared}$ buffer has the greatest one. Similar to simulations under uniform traffic, the reason for higher message latency on $DAMQ_{shared}$ buffer is that it holds much more flits than other two buffers, the numbers are shown in Table 4.10. In Figure 4.22, we can see that under traffic load 0.38, 16-flits $DAMQ_{shared}$ makes use of 45% of the whole buffer space while same sized $DAMQ_{all}$ and SAMQ uses 27% and 20% respectively. Again the 12-flits $DAMQ_{share}$ buffer can be used to provide similar performance as other SAMQ or $DAMQ_{all}$ and save the hardware cost by 25% with very little control logic cost.

Table 4.8

The performance 8-ary, 2-cube Network Composed of Block Switches. Shown is the Throughput Obtained from Simulations Where 4 virtual Channels Per Physical Channel Used When Hotspots Traffic is Applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.20	*.23*	*.29*	*.31*	*.33*	*.35*	*.38*
SAMQ	*16*	.391	.441	.494	.502	.508	.510	.518
DAMQ $_{all}$	*16*	.390	.442	.499	.506	.513	.516	.522
DAMQ$_{shared}$	*11*	.391	.441	.493	.503	.508	.510	.516
	12	.392	.442	.495	.502	.508	.510	.516
	13	.391	.444	.495	.503	.506	.512	.517
	14	.391	.444	.496	.503	.507	.512	.518
	15	.391	.444	.496	.503	.508	.511	.518
	16	.390	.444	.498	.505	.507	.510	.518

Table 4.9

The performance of 8-ary, 2-cube Network Composed of Block Switches. Shown is the Message Latency Obtained from Simulations Where 4 virtual Channels Per Physical Channel Used When Hotspots Traffic is Applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.20	*.23*	*.29*	*.31*	*.33*	*.35*	*.38*
SAMQ	*16*	103	121	165	177	186	192	200
DAMQ $_{all}$	*16*	95	117	163	178	186	192	202
DAMQ$_{shared}$	*11*	103	126	176	190	197	203	210
	12	103	125	178	193	202	207	217
	13	101	124	185	199	210	215	225
	14	102	125	188	204	216	220	230
	15	103	125	190	207	219	223	234
	16	101	125	192	209	222	228	237

Table 4.10

The performance of 8-ary, 2-cube Network Composed of Block Switches. Shown is the Buffer Usage Obtained from Simulations Where 4 virtual Channels Per Physical Channel Used When Hotspots Traffic is Applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.20	*.23*	*.29*	*.31*	*.33*	*.35*	*.38*
SAMQ	*16*	341	474	726	785	822	839	880
DAMQ $_{all}$	*16*	344	562	966	1043	1082	1081	1130
DAMQ$_{shared}$	*11*	410	592	890	937	962	985	1011
	12	449	651	1033	1102	1140	1147	1182
	13	480	701	1187	1243	1310	1319	1354
	14	483	762	1280	1390	1449	1479	1516
	15	494	780	1427	1564	1607	1643	1704
	16	495	820	1550	1681	1764	1775	1848

DAMQ$_{shared}$ on four virtual channels, 4/6/8 flit buffer per VC , 64 nodes and uniform traffic In this section we examine the network performances for 4 and 8 flits buffer for each virtual channel on SAMQ; fixed 4 flits buffer for each virtual channel on DAMQ$_{all}$. Three different size buffers, 4, 6 and 8 flits buffers for each virtual channel are used on DAMQ$_{shared}$ so that we can examine its performance in detail. The simulation results for throughput and message latency are shown in Table 4.11 and 4.12, respectively. We compare the performance of different buffer schemes until the network is saturated after about 0.4 traffic load is applied.

As shown in Figure 4.23, along with the network saturation process, our DAMQ$_{shared}$ has significant higher throughputs than both DAMQ$_{all}$ and SAMQ when they all use same size of 4 flits buffer and DAMQ$_{all}$ beats SAMQ. DAMQ$_{shared}$ not only achieves the highest throughput when same size buffer is used, but also achieves approximately the same maximum throughput as SAMQ using 8 flits buffer, which double the number of buffer DAMQ$_{shared}$ is used. When we use 6 flits buffer for DAMQ$_{shared}$, it achieves a significantly higher throughput than SAMQ with 8 flits buffer as shown in Figure 4.23. The max throughput is achieved by DAMQ$_{shared}$, when 8 flits buffer is used for it. In sum, DAMQ$_{shared}$ achieves best performance among the three buffer schemes we tested. The reason, we believe, is DAMQ$_{shared}$ provides more efficient methods for flits to share buffer space than DAMQ$_{all}$ which has already shown advantages over traditional SAMQ scheme. Furthermore, at the same maximum throughput level, DAMQ$_{shared}$ also has a similar latency as SAMQ as shown in Figure 4.24.

Table 4.11

The performance of 8-ary, 2-cube Network Composed of Block Switches. Shown is the Throughput Obtained from Simulations Where 4 virtual Channels Per Physical Channel Used.

Buffer Type	**BS per VC**	**Applied Traffic Load Rate**						
		0.20	0.23	0.29	0.31	0.33	0.35	0.38
SAMQ	4	0.39	0.44	0.53	0.54	0.55	0.55	0.55
	8	0.39	0.44	0.54	0.56	0.58	0.59	0.59
DAMQ $_{all}$	4	0.39	0.44	0.54	0.56	0.57	0.57	0.57
DAMQ$_{shared}$	4	0.39	0.44	0.54	0.57	0.58	0.59	0.59
	6	0.39	0.44	0.55	0.58	0.6	0.61	0.61
	8	0.39	0.44	0.55	0.58	0.6	0.62	0.62

Table 4.12

The performance of 8-ary, 2-cube Network Composed of Block Switches. Shown is the Message Latency Obtained from Simulations Where 4 virtual Channels Per Phy Channel Used.

Buffer Type	**BS per VC**	**Applied Traffic Load Rate**						
		0.20	0.23	0.29	0.31	0.33	0.35	0.38
SAMQ	4	102	113	140	156	167	174	186
	8	102	112	141	157	175	183	204
DAMQ *all*	4	102	110	142	157	175	184	198
DAMQ*shared*	4	102	113	142	158	178	187	207
	6	101	110	140	157	169	188	216
	8	102	110	139	155	169	185	225

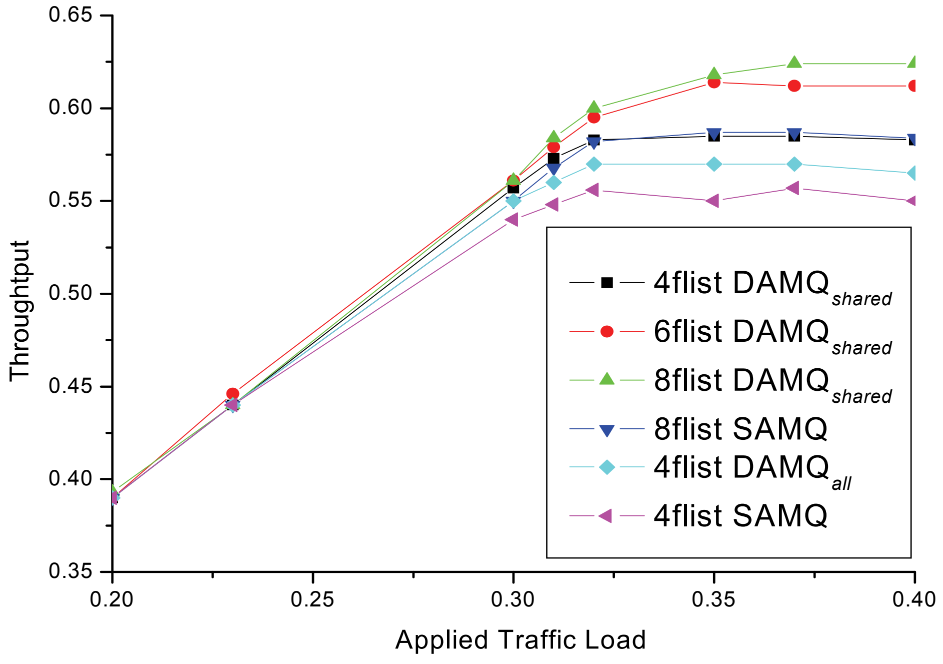


Figure 4.23: Comparison on throughput between 4/6/8 flit-buffer DAMQ$_{shared}$, 4 flit-buffer DAMQ$_{all}$ and 4/ 8 flit-buffer SAMQ

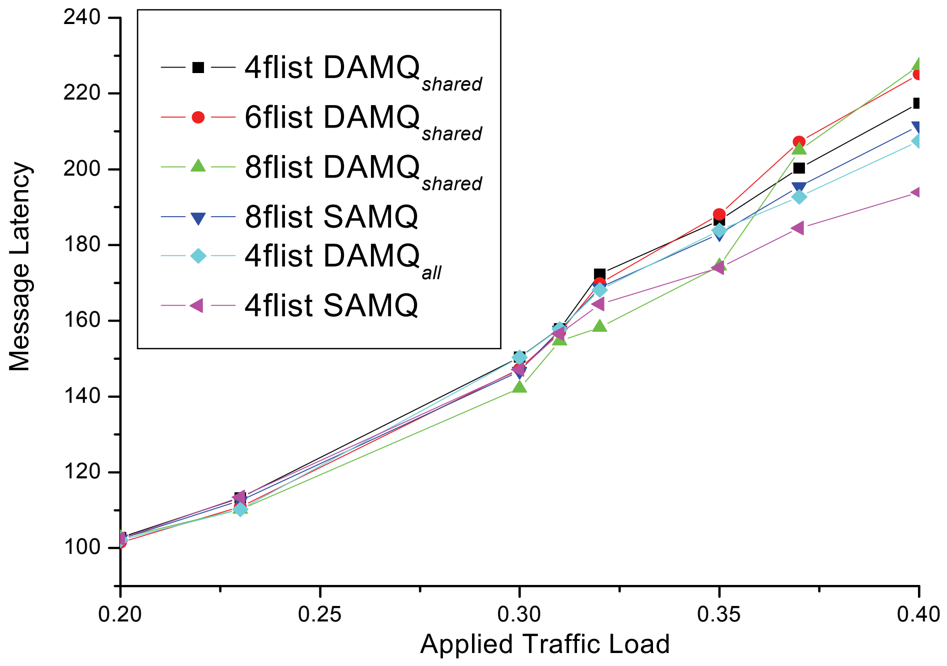


Figure 4.24: Comparison on Latency between 4/6/8 flit-buffer DAMQ$_{shared}$, 4 flit-buffer DAMQ$_{all}$ and 4/8 flit-buffer SAMQ

DAMQ$_{shared}$ on four virtual channels, 10/12/14/16 flit buffer per PC, 16 nodes and uniform traffic According to other researchers works in the literature [42] [43], we set the buffer size for each virtual channel to 4 flits when DAMQ$_{all}$ and SAMQ are used. Since four virtual channels are multiplexing cross one physical channel, the buffer size for each direction of a duplex physical channel is 16 flits when these two buffer schemes are evaluated. In order to examine the performance of DAMQ$_{shared}$ with regard to the relationship between buffer size and network performance, we use four different size 10, 12, 14 and16 flits buffer. The simulation results of network throughput and message latency are shown in Tables 4.13 and 4.14, respectively.

We compare the performance of different buffer schemes when the network starts to saturate on traffic load rate 0.2 until it gets severely saturated after about 0.5 traffic load is applied. As shown in Figure 4.25, along with the network saturation process, our new DAMQ$_{shared}$ has higher throughput than both DAMQ$_{all}$ and SAMQ when they all use same size 16-flits buffer. When the network gets saturated, DAMQ$_{shared}$ achieves the highest throughput while using same size 16-flit buffer. Furthermore, DAMQ$_{shared}$ with 10-flit buffer achieves approximately the same maximum throughput as SAMQ using 16-flit buffer. With 14-flit buffer, it also yields approximately the same throughput as 16-flit DAMQ$_{all}$ along with the network saturating process. In summary, DAMQ$_{shared}$ achieves the best performance among the three buffer schemes we studied . It provides a more efficient method for virtual channels to share buffer space than DAMQ$_{all}$ which also showed advantages over traditional SAMQ scheme.

As to the message latency, DAMQ$_{shared}$ managed to hold a similar latency as SAMQ until the network is congested after about 0.3 traffic load is applied. This is shown in Figure 4.26. When we further increase the traffic load after the network gets saturated, DAMQ$_{shared}$ shows higher latency than both DAMQ$_{all}$ and SAMQ. And the reason is mentioned earlier.

Under a traffic load of 0.5, when the three buffer schemes use same size 16-flit buffer, DAMQ$_{shared}$ utilizes 56% of the whole buffer space while DAMQ$_{all}$ and SAMQ use 43% and 30% respectively as shown in Figure 4.27. In addition, with 12-flit buffer, DAMQ$_{shared}$ already contains more flits in buffers than SAMQ along with the increasing of traffic load. It has also been shown that 10-flit DAMQ$_{shared}$ contains a similar number of flits as 16-flit

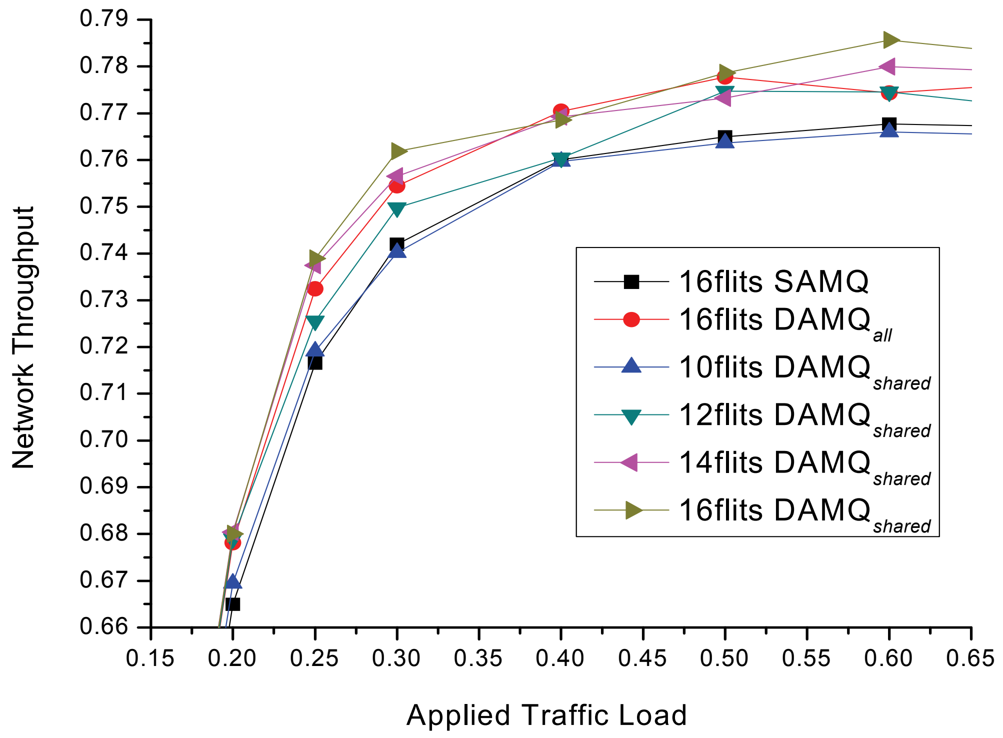


Figure 4.25: Comparison on throughput between 10-16 flit buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 16 flit-buffer SAMQ under uniform traffic.

Table 4.13

The Performance of 4-ary, 2-cube Network Composed of Block Switches. Shown is the Throughput Obtained from Simulations Where 4 Virtual Channels Per Physical Channel Used When Uniform Traffic is Applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.15	*.20*	*.25*	*.30*	*.40*	*.50*	*.60*
SAMQ	*16*	.554	.665	.716	.742	.760	.765	.767
DAMQ $_{all}$	*16*	.563	.678	.732	.754	.770	.777	.774
DAMQ$_{shared}$	*10*	.560	.670	.719	.741	.759	.762	.766
	12	.562	.679	.725	.749	.761	.774	.774
	14	.563	.680	.737	.756	.769	.773	.780
	16	.564	.680	.738	.764	.768	.778	.785

Table 4.14

The Performance of 4-ary, 2-cube Network Composed of Block Switches. Shown is the Message Latency Obtained from Simulations Where 4 Virtual Channels Per Physical Channel Used When Uniform Traffic is Applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.15	*.20*	*.25*	*.30*	*.40*	*.50*	*.60*
SAMQ	*16*	107	130	150	162	176	183	188
DAMQ $_{all}$	*16*	103	132	153	167	183	190	196
DAMQ$_{shared}$	*10*	110	137	159	170	182	190	193
	12	108	138	163	176	191	197	201
	14	108	140	165	182	199	206	210
	16	109	139	166	184	203	210	213

Table 4.15

The Performance of 4-ary, 2-cube Network Composed of Block Switches. Shown is the Buffer Usage Obtained from Simulations Where 4 Virtual Channels Per Physical Channel Used When Uniform Traffic is Applied.

Buffer Type	**BS per PC**	**Applied Traffic Load Rate**						
		.15	*.20*	*.25*	*.30*	*.40*	*.50*	*.60*
SAMQ	*16*	98	165	227	262	298	312	322
DAMQ $_{all}$	*16*	128	239	329	385	434	449	462
DAMQ$_{shared}$	*10*	107	170	220	241	262	271	276
	12	135	232	304	339	369	381	384
	14	155	280	375	425	468	482	490
	16	189	320	446	506	561	579	591

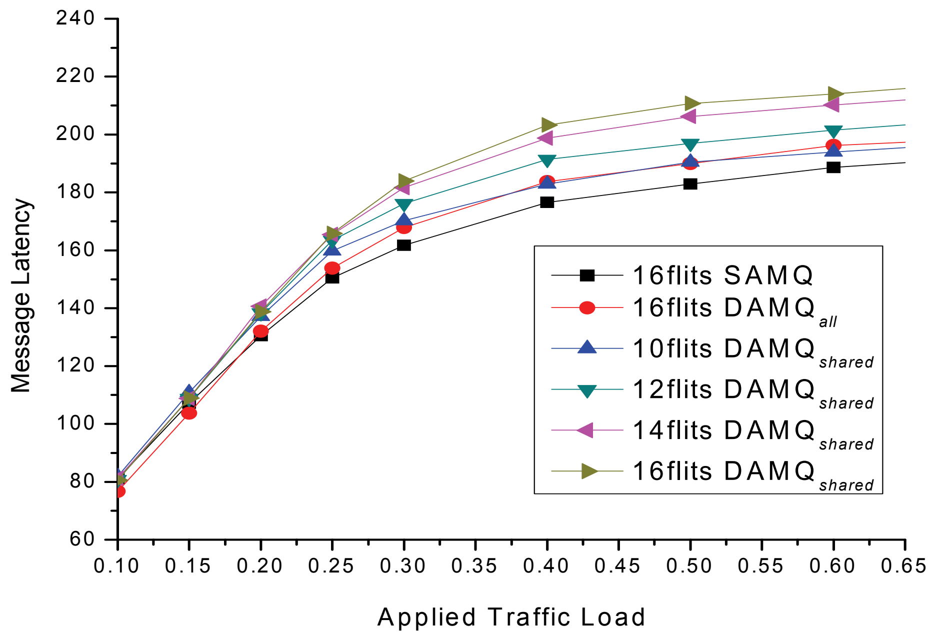


Figure 4.26: Comparison on latency between 10-16 flit-buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 16 flit-buffer SAMQ under uniform traffic.

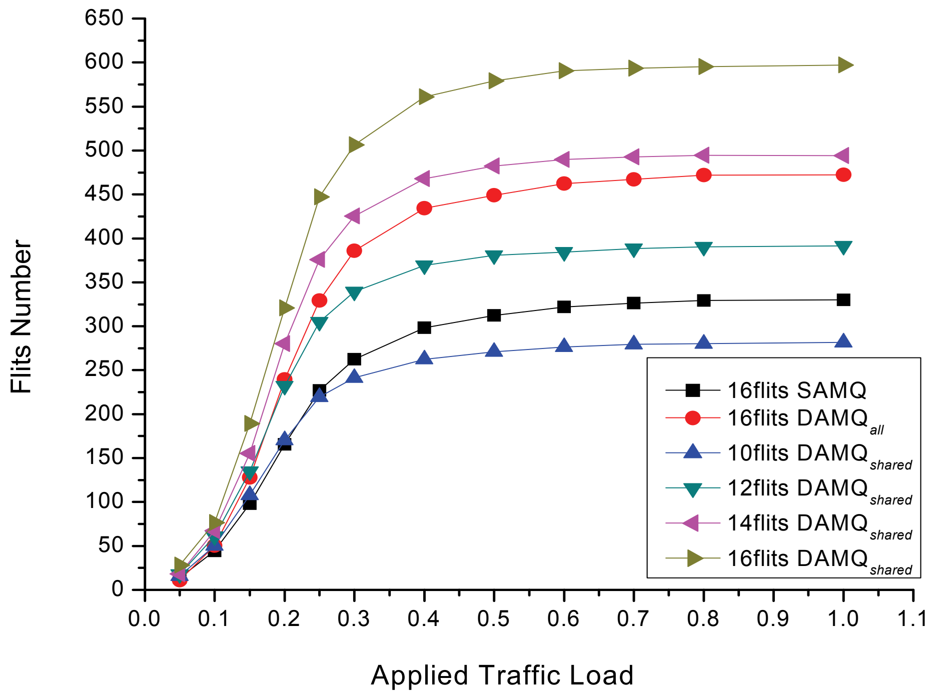


Figure 4.27: Comparison on buffer usages between 10-16 flit-buffer DAMQ$_{shared}$, 16 flit-buffer DAMQ$_{all}$ and 16 flit-buffer SAMQ under uniform traffic

4.5 Conclusion

In this chapter we have presented three novel buffer schemes based on a DAMQ self-compacting buffer. These schemes outperform existing approaches and DAMQ$_{shared}$ is the most efficient scheme among them.

DAMQ$_{all}$ and DAMQ$_{min}$ have similar performance using only half of the buffer size used in SAMQ when deterministic routing method is used. And they both provides higher throughput than SAMQ when same size buffers are used.

Based on DAMQ$_{all}$, we proposed DAMQ$_{shared}$ by letting virtual channels from two physical channel share free buffer space. The extensive simulations results show that DAMQ$_{shared}$ provides an excellent scheme to optimize buffer management. The proposed DAMQ$_{shared}$ scheme has the following features:

- *Less buffer space at similar performance*. At a similar throughput, DAMQ$_{shared}$ needs 37% and 12.5% less buffer space than SAMQ and DAMQ$_{all}$, respectively.
- *Better space utilization*. DAMQ$_{shared}$ utilizes more than 26% and 13% more buffer space than SAMQ and DAMQ$_{all}$ respectively in uniform traffic simulations.
- *Higher throughput*. It outperforms existing approaches with 2% - 3% higher than SAMQ on 16 nodes and 64 nodes network, and 1% higher than DAMQ$_{all}$ in uniform traffic simulations when same size buffer is used.

In summary, when an adaptive routing protocol such as Duato's algorithm is used for the NoC, DAMQ$_{shared}$ is an excellent scheme to optimize buffer management providing a good throughput when the network has a larger load. It can utilize significantly less buffer space without sacrificing the network performance.

Implementing the proposed schemes in hardware requires minor modifications to early implementations of the self-compacting buffer [40].

Chapter 5

Conclusion and Future Study

In this dissertation the results of a study on performance modeling and efficient buffer schemes for network on chip have been reported. The contributions of this work include proposing a network performance modeling method and evaluation of novel DAMQ self compacting buffer schemes. This chapter summarizes the study presented in this dissertation as well as lists the contribution of this study.

Chapter 2 provided background on issues related to this research work. The chapter included network topology, switching techniques, flow control, virtual channel, buffer schemes, routing algorithms and analytical models.

Chapter 3 presented a performance model for k-ary n-cubes network using fully adaptive routing algorithm. We use a 2D torus network as the analysis target and presented how to obtain average message latency through a series of queuing theory and probability calculations. By examining the predicted network performance against the results obtained in simulation experiments, we showed that this model yields satisfactory accuracy. Also as explained in chapter 3, this model can be applied to other topology network with minor modifications.

In Chapter 4, we presented three novel DAMQ self compacting buffer schemes. The schemes organization methods are described in detail. The schemes can be implemented in hardware with minor modifications based on the method proposed in [40]. Extensive simulations were performed on Flexsim1.2 [18] to evaluate the performance of these buffer schemes. The simulations results on different network configurations, such as different traffic mode, network size, virtual channel number, buffer size per virtual/physical channel and routing protocols, were presented to show that these novel buffer schemes especially the DAMQ_{shared} scheme are efficient buffer organization methods to be used in the network on chip. As shown in our simulation results, these buffer schemes can provide marginally higher throughput than traditional SAMQ when same amount of resource is used, this is due to the fact that buffer cannot play a major role in determining the network performance in terms of throughput or latency. However, the results show that these schemes can use significantly less hardware to provide a same performance as traditional SAMQ buffer.

5.1 Major Contributions

The major contribution of this study can be summarized as follows:

- This research sought the ways of predicting the performance of directed connected network on a chip which is characterized by limited resource
- An analytical model for directed connected network, *k*-ary *n*-cubes has been proposed and evaluated.
- Two novel DAMQ buffer schemes, DAMQ_{all}, DAMQ_{min}, which let virtual channels of one physical channel share free buffer space, have been proposed, simulated and evaluated.
- A novel DAMQ buffer scheme, DAMQ_{shared}, which let virtual channels from two physical channel of same communication node share free buffer space, has been proposed, simulated and evaluated.

5.2 Future Research Directions

This section suggests areas for future work to complement this study in both performance modeling and buffer schemes for network on chip.

For the network performance modeling, as we can see from the calculation process presented in chapter 3, blocking probability at a channel determines the waiting time for a message given a specific service rate. A higher blocking probability will results in an exponentially larger waiting time. Thus if we can reduce the blocking probability for a message travelling across the network, the network will have better performance. At given message generation rate and link service rate, one can alter the message blocking probability by different routing algorithm, network topology, or increasing the buffer size at every communicating node. The first two are major factors in determining network performance while the buffer size has smaller impact [2]. Because buffer is usually not deep enough to store the whole blocked message, so that it may not be able to free the upstream channels that occupied by the blocked message. This is especially the case for network on chip with wormhole switching, where buffer needs to be carefully designed to be both compact and efficient to reduce hardware cost while maintaining network performance. Therefore, to predict the subtle impact of buffer size on the network performance, we need more sophisticate queuing system analysis.

Hence our future work on NoC performance modeling can be to extend the model to incorporate buffer size impact, i.e. how to model a network with virtual channels that have buffers that are more than 1 flit deep. By mathematically figuring out the contribution that buffer offers in alleviating blocking probability, we can better predict the performance of directly connected network with more complicate configurations.

We can also extend the work on DAMQ buffer schemes for the systems with NoC. For the short term goal, we can continue studying hardware requirements and effect of proposed buffer in other network topologies. For the long term goal, we can extend these buffer schemes to incorporate fault tolerance ability.

The proposed buffer schemes are based on self compacting buffer hardware design. As these schemes have additional features to the original SCB, the new buffers may require a larger control circuit. This however can be compensated by the need of far less buffer space.

We can also study the performance of proposed schemes in other network topologies, for instance, SPIN (Scalable, Programmable, Integrated Network) [27], Butterfly Fat-Tree [50] and Octagon [30] etc.

Fault tolerant mechanisms for interconnection networks are becoming a critical design issue for large massively parallel computers. [45] It is also important to high performance SoCs as the system complexity keeps increasing rapidly. Researchers have reported many routing protocols to provide fault tolerant mechanism [46-49]. On the message switching layer, we can make improvement to boost system performance when there are faults involved in the components communication. The basic proposal is when a node or a physical channel is deemed as faulty, the previous hop node will terminate the buffer occupancy of messages destined to the failed link. The buffer usage decisions are made at switching layer without interactions with higher abstract layer, thus buffer space will be released to messages destined to other healthy nodes quickly. Therefore, the buffer space will be efficiently used in case fault occurs at some nodes. For example, if node A is connecting to node B and C and there are message flows from A to B and A to C. When B fails, there are probably still a number of message flits left in A's buffer. It will improve the system performance if the buffer space occupied by B’s message can be allocated to C’s message quickly. It would be interesting to conduct this research in the future.

References

[1] D. Bertsekas and R. Gallager, Data Networks, second edition. Prentice-Hall, 1992.

[2] J.Duato, S. Yalmanchili and L. Ni, Interconnection Networks, IEEE Computer Society, 1997

[3] Agarwal, "Limits on Interconnection Network Performance", IEEE Trans. Parallel and Distributed Systems, vol. 2, pp. 398-412, 1991.

[4] Y. Boura, C.R. Das, and T.M. Jacob, "A Performance Model for Adaptive Routing in Hypercubes," Proc. Int'l Workshop Parallel Processing, pp. 11-16, Dec. 1994.

[5] M. Ould-Khaoua, "A Performance Model for Duato's Fully Adaptive Routing Algorithm in k-Ary n-Cubes", IEEE Transactions on Computers, v.48 n.12, p.1297-1304, Dec 1999.

[6] B. Ciciani, M. Colajanni, and C. Paolucci, "An Accurate Model for the Performance Analysis of Deterministic Wormhole Routing, " Proc. 11th Int'l Parallel Processing Symp., pp. 353-359, 1997.

[7] W.J. Dally, "Performance Analysis of k-ary n-cubes Interconnection Networks," IEEE Trans. Computers, vol. 39, no. 6, pp. 775-785, June 1990.

[8] J. Duato and P. Lopez, "Performance Evaluation of Adaptive Routing Algorithms for k-ary n-cubes," Proc. First Workshop Parallel Computer Routing and Comm., K. Bolding and L. Snyder, eds., pp. 45-59, May 1994.

[9] J.T. Draper and J. Ghosh, “A Comprehensive Analytical Model for Wormhole Routing in Multicomputer Systems,” J. Parallel and Distributed Computing, vol. 32, pp. 202-214, 1994.

[10] R. Greenberg and L. Guan, “Modeling and Comparison of Wormhole Routed Mesh and Torus Networks,” Proc. Ninth IASTED Int'l Conf. Parallel and Distributed Computing and Systems, 1997.

[11] W.J. Guan, W.K. Tsai, and D. Blough, “An Analytical Model for Wormhole Routing in Multicomputer Interconnection Networks”, Proc. Int'l Conf. Parallel Processing, pp. 650-654, 1993.

[12] J. Hu, U.Y. Ogras, R. Marculescu, “System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design” IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 12, pp. 2919-2933, Dec. 2006.

[13] P. Hu and L. Kleinrock, "An Analytical Model for Wormhole Routing with Finite Size Input Buffers", Proc. 15th Int’l Teletraffic Congress, 1997.

[14] V. S. Adve , M. K. Vernon, “Performance Analysis of Mesh Interconnection Networks with Deterministic Routing”, IEEE Trans Parallel and Distributed Systems, v.5 n.3, p.225-246, March 1994

[15] C. Chen, W. Wu, Z. Li, “Multipath routing modeling in ad hoc networks” Proc. ICC 2005. 2005 IEEE Int’l Conference on Vol5, 16-20 May 2005 P,2974 – 2978

[16] J. Duato, “A New Theory of Deadlock-Free Adaptive Routing in Wormhole Routing Networks,” IEEE trans. Parallel and Distributed Systems, vol. 4, no. 12, pp. 1,320-1,331, Dec. 1993.

[17] W.J. Dally and C.L. Seitz, “Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,” IEEE Trans. Computers,vol. 36, no. 5, pp. 547-553, May 1987.

[18] S. Warnakulasuriya and T.M. Pinkston, “Characterization of Deadlocks in k-ary n-cube Networks,” IEEE Trans. Parallel and Distributed Systems, vol. 10, no 9, Sept. 1999, 904-932.

[19] J. Duato, “A new theory of deadlock-free adaptive routing in wormhole networks,” IEEE Trans. Parallel Distributed Syst., vol. 4, no. 12, Dec. 1993, 1320–1331.

[20] SMART Interconnects Grp, USC, FlexSim1.2, 2002

[21] Patrick T. Gaughan and Sudhakar Yalamanchili, “Adaptive Routing Protocols for HvDercub Interconnection Networks,” IEEE Trans. Computer, vol. 26, no 5, pp. 12-23, May 1993.

[22] W.J. Dally and H. Aoki, “Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channnels,” IEEE Trans. Parallel and Distributed Systems, Vol. 4, No. 4, pp. 466-475, April 1993

[23] E. Baydal, P. L´opez and J. Duato, "Increasing the Adaptivity of Routing Algorithms for k-ary n-cubes," in Proc. 10th Euromicro Workshop on Distributed and Network-based Processing, pp. 455-462, Jan. 2002

[24] L. Benini and G. De Micheli, “Networks on chips: a new SoC paradigm,” IEEE Computer, Vol. 35 , No. 1, Jan. 2002, 70-78.

[25] P. Magarshack and P.G. Paulin, “System-on-Chip beyond the Nanometer Wall,” Proc. Design Automation Conf. (DAC), pp. 419-424, June 2003.

[26] M. Horowitz and B. Dally, “How Scaling Will Change ProcessorArchitecture,” Proc. Int’l Solid State Circuits Conf. (ISSCC), pp. 132-133, Feb. 2004.

[27] P. Guerrier and A. Greiner, “A Generic Architecture for On-Chip Packet-Switched Interconnections,” Proc. Design and Test in Europe (DATE), pp. 250-256, Mar. 2000.

[28] S. Kumar et al., “A Network on Chip Architecture and Design Methodology,” Proc. Int’l Symp. VLSI (ISVLSI), pp. 117-124, 2002.

[29] W.J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks,” Proc. Design Automation Conf. (DAC), pp. 683-689, 2001.

[30] F. Karim et al., “An Interconnect Architecture for Networking Systems on Chips,” IEEE Micro, vol. 22, no. 5, pp. 36-45, Sept./Oct.2002.

[31] P.P. Pande. et al. "Performance Evaluation and Design Trade-offs for MP-SoC Interconnect Architectures”, IEEE Trans. Computers, vol. 54, no. 8, Aug 2005, 1025-1040.

[32] S.F. Nugent, "The iPSC/2 Direct-Connect Communication Technology," Proc. Conf. Hypercube Concurrent Computers and Applications, vol. 1, pp. 51-60, 1988.

[33] M. Noakes and W.J. Dally, "System Design of the J-Machine," Proc. Advanced Research in VLSI, pp. 179-192, 1990.

[34] R.E. Kessler and J.L. Schwarzmeier, "CRAY T3D: A New Dimension for Cray Research, Proc. CompCon, pp. 176-182, Spring 1993.

[35] J. Kim and C.R. Das, “Hypercube Communication Delay with Wormhole Routing,” IEEE Trans. Computers, vol. 43, no. 7, pp. 806-814, July 1994.

[36] C. L. Wu and T. Y. Feng, “On a class of multistage interconnection networks,” IEEE Transactions on Computers, vol. C-29, No.8, pp. 694-702, Aug. 1980

[37] R. Sivaram, C. B. Stunkel, and D. K. Panda, “HIPIQS: A High Performance switch architecture using input queueing,” IPPS/SPDP ’98, pp. 134–143, Orlando, FL, March 1998.

[38] C. Grecu, et al “Structured Interconnect Architecture: A Solution for the Non-Scalability of Bus-Based SoCs,” Proc. Great Lakes Symp. VLSI, pp. 192-195, Apr. 2004

[39] Y. Tamir and G. L. Frazier, “Dynamically-allocated multiqueue buffers for VLSI communication switches,” IEEE Transactions on Computers, vol. 41, no. 2, pp. 725–737, June 1992.

[40] J. G. Delgado-Frias and R. Diaz, “A VLSI Self-Compacting Buffer for DAMQ Communication Switches,” IEEE Eighth Great Lakes Symposium on VLSI, pp. 128-133, Lafayette, Louisiana, February 1998.

[41] J. Park, B. O’Krafka, S. Vassiliadis, and J. Delgado-Frias, “Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers,” IEEE Supercomputing ’94, The Conference on High Performance Computing and Communications, pp. 713–722, Nov.14-18, 1994

[42] Santi, S. et al. “On the Impact of Traffic Statistics on Quality of Service for Networks on Chip”, ISCAS´05, 2349-2352.

[43] J. Liu, J. G. Delgado-Frias, “DAMQ Self-Compacting Buffer Schemes for Systems with Network-On-Chip,” In Proc, 2005 Int. Conf. Comput Design, pp. 97-103, Las Vegas, June 2005.

[44] W. Dally. “Virtual-channel flow control,” IEEE Transactions on Parallel and Distributed Systems, vol. 3, no. 2, pp. 194–205, 1992

[45] M. E. Gomez, et al, “A Routing Methodology for Achieving Fault Tolerance in Direct Networks”, IEEE Trans, Computers, vol 55, no. 4, pp 400-415, Apr. 2006.

[46] C.T. Ho and L. Stockmeyer, “A New Approach to Fault-Tolerant Wormhole Routing for Mesh-Connected Parallel Computers,” IEEE Trans. Computers, vol. 53, no. 4, pp. 427-439, Apr. 2004.

[47] Z. Jiang, J. Wu, and D. Wang, “A New Fault Information Model for Fault-Tolerant Adaptive and Minimal Routing in 3-D Meshes,” Proc. Int’l Conf. Parallel Processing, pp. 500-507, June 2005.

[48] M. E. Gomez, et al, “A Routing Methodology for Achieving Fault Tolerance in Direct Networks”, IEEE Trans, Computers, vol 55, no. 4, pp 400-415, Apr. 2006

[49] P.T. Gaughana and S. Yalamanchili, “A Family of Fault-Tolerant Routing Protocols for Direct Multiprocessor Networks,” IEEE Trans. Parallel and Distributed Systems, vol. 6, no. 5, pp. 482-497,May 1995.

[50] P.P. Pande, C. Grecu, A. Ivanov, and R. Saleh, “Design of a Switch for Network on Chip Applications,” Proc. Int’l Symp. Circuits and Systems (ISCAS), vol. 5, pp. 217-220, May 2003.