

**Design of an Interconnection Network  
Using VLSI Photonics and Free-Space  
Optical Technologies**

**Ch'ng Shi Baw  
Roger D. Chamberlain  
Mark A. Franklin**

Ch'ng Shi Baw, Roger D. Chamberlain, and Mark A. Franklin, "Design of an Interconnection Network Using VLSI Photonics and Free-Space Optical Technologies," in *Proc. of the 6th Int'l Conf. on Parallel Interconnects*, October 1999, pp. 52-61.

Computer and Communications Research Center  
Washington University  
Campus Box 1115  
One Brookings Dr.  
St. Louis, MO 63130-4899

# Design of an Interconnection Network Using VLSI Photonics and Free-Space Optical Technologies\*

Ch'ng Shi Baw                      Roger D. Chamberlain                      Mark A. Franklin  
*baw@ccrc.wustl.edu*                      *roger@ccrc.wustl.edu*                      *jbf@ccrc.wustl.edu*  
Computer and Communications Research Center  
Washington University, St. Louis, Missouri

## Abstract

*This paper presents the design and initial analysis of an optically interconnected multiprocessor based on the use of VCSELs (Vertical Cavity Surface Emitting Laser) and free-space optical interconnects. The design is oriented to applications where the performance is bandwidth limited in conventional multiprocessors. The processor interconnection network is based on a physical ring topology which is logically configured as a multiring. Two alternative communications protocols are presented and the performance properties associated with a packet-based Go-Back-N protocol are discussed. Relationships between bit error rate and performance are provided.*

## 1 Introduction

Recent advances in VLSI photonic technologies enable us to design and implement optical interconnects with terabits per second bandwidth capacity. In this paper, we present a simple, feasible architecture that exploits the high bandwidth provided by these new technologies. The design is targeted at multiprocessor systems and applications that require frequent, massive data transfer among processors as well as input/output devices. The design proposed can be implemented in a relatively straightforward manner with available base component technologies and demonstrates the benefits that accrue from the high bandwidth provided by VLSI photonics in the interconnection fabric.

At the heart of the architecture is a VLSI photonic device based on the use of an  $M \times M$  matrix of *Vertical Cavity Surface Emitting Laser (VCSEL)* and detector pairs. Each VCSEL-detector pair is capable of operating at the rates of 1 to a few gigabits per second (Gbps). With  $M = 32$ , individual data rates of 1 Gbps, and a quarter of the VCSEL-detector pairs active, one can expect the device to deliver 64 Gbps of

raw bandwidth. With  $M = 128$ , the raw bandwidth deliverable will be in excess of four terabits per second (Tbps). Though not described here, the VCSELs will communicate with each other using optoelectrical components which permit free space interaction. Early work in this area can be found in [2, 7].

Section 2 presents the overall architectural organization of the system. Two basic transmission protocols for efficient data transfer are presented in Section 3 and a preliminary performance analysis is provided in Section 4. In Section 5 the protocols described earlier are improved upon by adding more refined control and scheduling techniques to deal with fairness issues. Section 6 summarizes our results and concludes with suggestions for future research.

## 2 Network Architecture: Topology and Channel Assignment

The approach selected for the multiprocessor interconnect utilizes a ring topology. Consider a four node ring example where each processing node is connected to the ring as shown in Figure 1. Given the numerous VCSEL-detector pairs, we can assign disjoint subsets of VCSEL-detector pairs to each processing node. If these subsets are allocated according to receiver designation, then each subset can be thought of as a channel associated with messages being received by a given node. This arrangement implements a multiring topology [5, 6].<sup>1</sup>

Figure 1 illustrates the logical topology of a multiring. In [5] and [6], the physical links are constructed using optical fiber and exploit tunable lasers to implement WDM multiplexing. Here, we do not require tunable lasers, since there are sufficient channels using space division alone.

---

<sup>1</sup>Given the number of VCSEL-detector pairs available, a complete connection scheme is also possible here if one associates a source and designation with a smaller subset. However, this limits channel sharing and will lead to lower overall performance.

---

\*This work was supported in part by the DARPA/ARL VLSI Photonics Program under contract L01-98-C-0074.

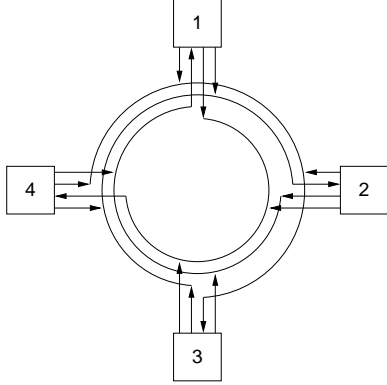


Figure 1: Logical multiring topology [5].

## 2.1 The Multiring

Figure 2 shows one possible allocation of a  $16 \times 16$  VCSEL-detector matrix on a four node system. Using the multiring, a four node system requires four channels since each node represents a receiver that has its own channel. The top left part of Figure 2 is the  $16 \times 16$  transmitter grid which is divided into four vertical stripes, each containing  $4 \times 16 = 64$  VCSELs. The top right shows the receiver grid similarly divided into four vertical stripes each also containing  $4 \times 16 = 64$  detectors. Transmitter stripes are assigned channel numbers from left to right (e.g., 1, 2, 3, and 4) while receiver stripes are assigned in reverse order. The multiring has the following advantages:

- **No Need for Explicit Destination Address Specification:** An incoming message landing on the detectors assigned to channel  $i$  on node  $i$ 's receiver automatically indicates that the message destination is node  $i$ .
- **No Need for Explicit Routing:** If the incoming message lands on any other detectors (e.g., the detectors assigned to channel  $j$ ) on node  $i$ 's receiver, then node  $i$  knows the message destination is node  $j$  and needs only to repeat the message on its VCSELs assigned to channel  $j$ . No routing table lookup or similar operation is necessary.

With the multiring topology, each channel can be thought of as a daisy chain terminating at the receiver. Figure 3 shows a four node (P1, P2, P3, and P4), four channel system with each channel viewed as a logical daisy chain terminating at its receiver.

In the example, if node 4 wants to send a message to node 2, it will send the message on channel 2. The message will first land on channel 2 of node 1's detector matrix. Node 1 will then repeat that message on channel 2 of its VCSEL matrix. The message is then

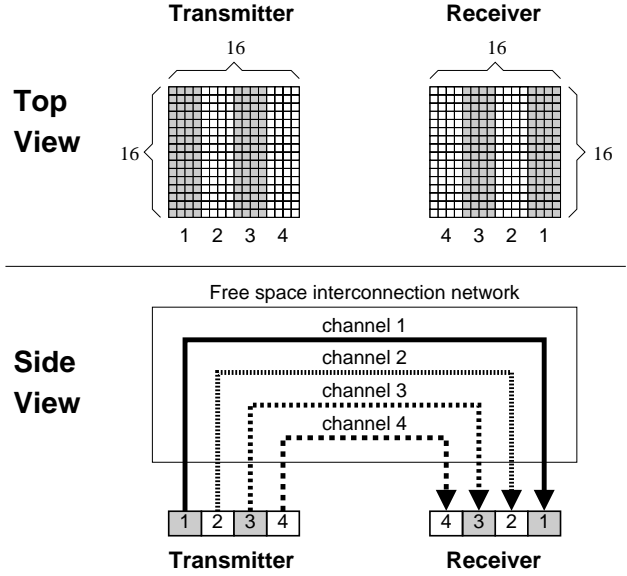


Figure 2: Allocation of VCSEL-detector pairs to a four channel, four node system.  $16 \times 16$  VCSEL-detector matrices are used.

reflected to channel 2 of node 2's detector matrix and is thus received by node 2.

## 2.2 Control Channels

Referring to Figure 3, note that since a node will never send messages to itself, for a four node ring, each node needs to monitor only three receiver channels and sends on at most 3 transmitter channels. In a multiring, node  $i$  will never send on channel  $i$ . Thus, if we divide each VCSEL/detector matrix into  $n$  channels on an  $n$  node system, there will always be an *extra* channel (marked *extra* in Figure 3).

While there are a number of ways to utilize the extra channel associated with each node, we will focus on its use in implementing various control functions needed in an efficient and error-tolerant transmission protocol. These subchannels will be referred to as *control* channels (as opposed to the *data* channels). While control messages could be incorporated within data messages, with the extra channels reallocated for use as general data channels, there are advantages to explicitly separating control functions from the data stream. The primary advantage is design simplicity. By providing for separate control channels, no extra logic is necessary to separate out control from data messages. Furthermore, since we expect that data messages (for the target applications) will be very long, and control messages relatively short, this separation permits further optimization of the design to account for disparate mes-

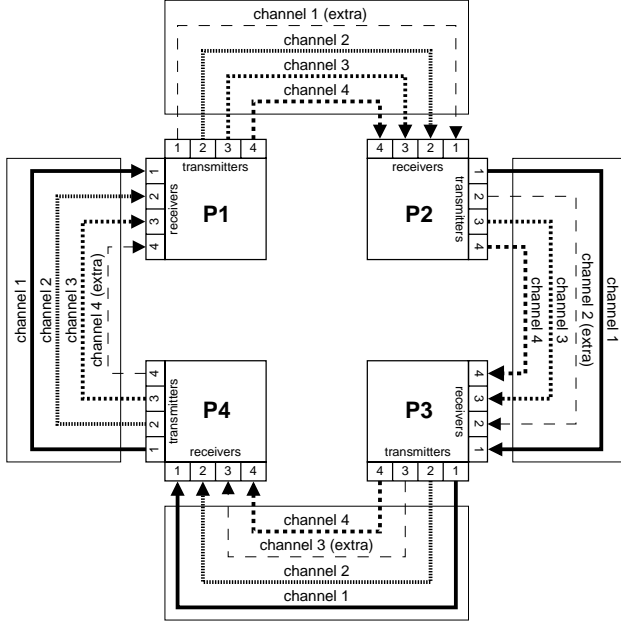


Figure 3: Conceptual diagram of a 4 node multiring.

sage lengths. Finally, having separate control channels eliminates potentially large delays which could occur if short control messages had to contend with very long data messages for use of a channel. Such control delays could significantly degrade performance of the overall interconnection network.

### 3 Network Architecture: Message Transmission Issues

The sections above considered the underlying topology, channel allocation, and interconnection design. We now consider methods for utilizing this design in a manner that will provide for efficient and error-tolerant message transmission. Two approaches are distinguished: *store-and-forward* (i.e., packet-by-packet) or *per connection* (i.e., circuit switched) with each having its advantages and disadvantages. We propose a feasible protocol for each approach but focus on the packet-based scheme. Protocols requiring no arbitration are considered first. Later, in Section 5, arbitration issues are discussed.

Two factors influence the protocol design:

- **High Data Rate.** It is expected that each channel operates at a very high bit rate (e.g., multiple gigabits per second). As such, it is important to avoid buffering data since the cost of buffering a data burst can be high.
- **Unreliable Links.** Free space optical-mechanical components may become misaligned due to vibration, temperature change, etc. This combination

of factors can lead to bit error rates which must be dealt with effectively by the data transmission protocol.

The following definitions will be used in describing the protocols:

- **Message:** A message refers to a logical unit of data being sent by the application or middleware (e.g., MPI [4]).
- **Packet:** In a store-and-forward protocol, a message may be broken down into smaller, fixed-sized units referred to as *packets*. A *packet* is a protocol-level abstraction.
- **Frame:** At the hardware level, data are sent in *frames* where the hardware has the ability to recognize frame boundaries. A packet may be broken down into frames or each packet may be a frame by itself. Each frame or packet carries a sequence/identification number.
- **Signals:** Signals are control messages that are sent through the network to regulate and coordinate the interconnection fabric. In our design, signals use the control channels, are always buffered, and are sent in a store-and-forward manner.

Since link reliability issues must be addressed, error detection and feedback are necessary for efficient operation. Basic error detection is assumed to take place (mainly in hardware) at the frame and packet levels via standard check code mechanisms. The application or middleware will be responsible for message-level error detection. It is also assumed that when data is corrupted it is still possible for a node to detect the presence of a transmission. Other errors such as a link being severed or a node crashing are not considered.

#### 3.1 Packet-Based Transmission (PBT)

##### 3.1.1 Basic Protocol for PBT

Using a packet-based transmission protocol, messages are decomposed into small, fixed-sized packets before transmission. The simplest possible PBT protocol is *Stop-and-Wait*.<sup>2</sup> The basic idea in the *Stop-and-Wait* protocol is that packets are sent by the source and then stored and forwarded by successive nodes until the destination node is reached. During this store-and-forward process the source node stops-and-waits for an *acknowledgement signal (ACK)* for the packet. The destination node, upon receiving the packet, responds by sending an *ACK* (using the control channels) to the source node. If the *ACK* is received by the source node within a time-out period, the source node sends its

<sup>2</sup>Stop-and-Wait is also referred to as the *Alternating Bit Protocol (ABT)* in the literature [1].

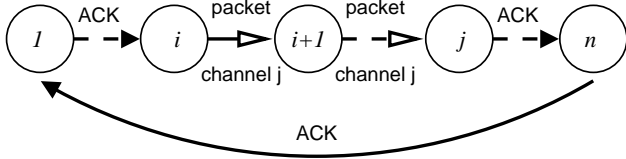


Figure 4: Node  $i$  sending a message to node  $j$ .

next packet. Otherwise, the source node resends the packet. The process continues until all source packets have been delivered. After sending a packet, the source needs to wait at least one roundtrip time (for the packet to be delivered and the ACK to get back) before it can send the next packet. The roundtrip time can be significant for large rings. The result is that the source spends a lot of time waiting, message delivery times are large, and channel utilization is low.

To improve this situation we turn to the Go-Back-N protocol [1] which is a generalization of Stop-and-Wait. In this situation the source need not wait for an ACKnowledgement before sending the next packet. After establishing a parameter  $N$ , referred to as the *window size*, the sender can send up to  $N - 1$  additional packets while waiting for receipt of the ACK for the initial packet. In this way multiple packets can be in transit from a given node simultaneously. This significantly improves overall channel utilization and reduces message delivery times.

Assume that node  $i$  wants to send a message to node  $j$  (Figure 4) with the message being divided into  $x$  packets. In Go-Back-N, node  $i$  is allowed to send packets  $1, 2, \dots, N$  before an ACK for packet 1 is received. The sequence  $\{1, 2, \dots, N\}$  is referred to as the *window*. When an ACK for packet 1 is received, the window advances to  $\{2, 3, \dots, N + 1\}$ , and node  $i$  sends packet  $(N + 1)$ . This continues and the window *slides* forward as more and more ACKs are received.

Note that a packet from  $i$  is forwarded  $(j - i)$  times before it reaches  $j$ . If the ACK signal sent by  $j$  also propagates in a store-and-forward manner, then it is forwarded  $(n - j + i)$  times before it reaches  $i$ . Let  $t_{pkt}$  be the time needed to forward one packet,<sup>3</sup> and  $t_{ACK}$  the time needed to store and forward an ACK. If the ACK generation time is negligible, then the time elapsed between node  $i$  sending a packet and node  $i$  receiving a corresponding ACK is:

$$t_{roundtrip} = ((j - i) \bmod n) \cdot t_{pkt} + ((i - j) \bmod n) \cdot t_{ACK} \quad (1)$$

If we have an  $n$  node unidirectional ring, assuming

<sup>3</sup> $t_{pkt}$  will be more rigorously redefined in Section 4.

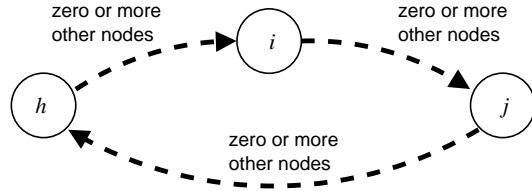


Figure 5: Nodes  $h$ ,  $i$ , and  $j$  on a ring.

a uniform distribution of node message activity, the roundtrip time would be  $t_{roundtrip} = n/2 \times (t_{pkt} + t_{ACK})$  on average.

In an error free environment,  $N$  can be chosen to be arbitrarily large so that the sender can keep transmitting new packets without waiting for an ACKnowledgement. We would thus have a pipeline effect and this would result in full channel utilization. When errors occur, however, the sender will need to retransmit packets that are not properly ACKnowledged and thus the efficiency will be reduced. Note that a time out parameter,  $t_{out}$ , needs to be selected so that if a packet is not acknowledged within time  $t_{out}$  after transmission, the sender will resend the packet. With the Go-Back-N protocol, all packets within the current window are retransmitted if an ACK is not received within  $t_{out}$ , however, with more complex protocols (e.g., *Selective Repeat Request (SRQ)*) only those packets within the window for which no ACK has been received need to be retransmitted. The SRQ approach requires a more involved retransmission scheme since the window may now contain non-contiguous packets due to errors. We restrict ourselves here to the simple Go-Back-N protocol. Choice of various protocol parameters is considered later.

### 3.1.2 Contending for a Channel

Situations will naturally arise where multiple senders will want to use the same channel at the same time. This section considers an approach to deal with such situations without buffering blocked messages.

Assume that nodes  $h$ ,  $i$ , and  $j$  are connected in an  $n$  node ring (see Figure 5) and that both nodes  $h$  and  $i$  have messages they want to send to node  $j$ . Before sending any packet to  $j$ ,  $i$  first monitors channel  $j$  for a short period of time (e.g., one packet time) to determine if channel  $j$  is in use. If any node between  $i$  and  $j$  is sending packets to  $j$ , node  $i$  will not know that channel  $j$  is in use by monitoring its segment of channel  $j$  since the on going traffic does not flow through  $i$ . Hence node  $i$  will assume that channel  $j$  is not in use. If any other node (e.g., node  $h$ ) before  $i$  is sending packets on channel  $j$ , node  $i$  will know that channel  $j$

is in use.

If node  $i$  determines that channel  $j$  is in use, it can either continue monitoring channel  $j$  until it appears to node  $i$  that channel  $j$  is no longer in use, or node  $i$  can abandon monitoring and wait for some time before reinitiating monitoring. If node  $i$  determines that channel  $j$  is not in use, it initiates transmission (using Go-Back-N).

Once a node starts transmission on a channel, it will not forward other nodes' packets on that channel. For example, if, after node  $i$  initiated transmission on channel  $j$ , node  $h$  also initiates transmission on channel  $j$ , node  $h$ 's packets will be discarded by node  $i$  (assuming that no other node before  $i$  is accessing channel  $j$ ). Similarly, if node  $i$  initiates transmission after some other node between  $i$  and  $j$  started transmission on channel  $j$ , then that other node will *not* forward node  $i$ 's packets on channel  $j$ .

Hence, if node  $i$ 's packets successfully reach node  $j$ , node  $i$  has successfully gained access to channel  $j$ . If there were other nodes contending for channel  $j$ , node  $i$  has won the contention. As soon as node  $j$  receives a packet from node  $i$ , node  $j$  issues a *channel-captured* signal to  $i$  to notify  $i$  that it has gained access to the channel. If we assume that a separate control channel design has been adopted, then the *channel-captured* signal will utilize a control channel and be passed between nodes in a store-and-forward manner.

Near the end of a Go-Back-N transmission, there may not be enough packets to fill a window. For example, the window size  $N$  may be 5 but there are only 4 or less packets remaining to be sent. It is important to insure that the channel remains captured until the last packet has been sent and its associated *ACK* has been received. If this is not done then the channel may be lost and, if an error has occurred, a recapture process would be required. The sender can maintain channel capture even after the last packet has been sent by inserting filler packets to keep the window full. Note that filler packets need not be *ACK*nnowledged and may also have other uses as discussed in Section 3.1.4.

### 3.1.3 Timing Out a Transmission

A node may not realize that the channel is in use before it initiates transmission. For example, suppose node  $i$  has successfully captured channel  $j$ . Node  $h$  does not know that channel  $j$  is in use during its monitoring process. Thus node  $h$  may begin transmission only to have all its packets discarded by node  $i$ . In this case, node  $h$  has two options:

1. Node  $h$  can keep transmitting until node  $i$  finished transmission to  $j$ . At which point, should other nodes between  $i$  and  $j$  *not* begin transmission to

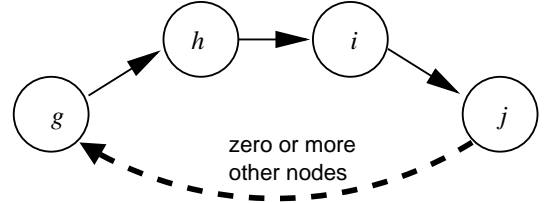


Figure 6: Nodes  $g$ ,  $h$ ,  $i$ , and  $j$  on a ring.

$j$ , node  $h$ 's packet will reach  $j$  and node  $h$  captures channel  $j$ .

2. Node  $h$  can keep transmitting until some preset time out period,  $t_{cout}$ . If a *channel-captured* signal is received before time  $t_{cout}$ , node  $h$  continues transmission. Otherwise, node  $h$  stops transmission and tries again at a later time.

In Option 1 channel  $j$  between  $h$  and  $i$  is held by node  $h$  thus preventing and other nodes between  $h$  and  $i$  from capturing the channel. It is not clear, however, that this is a significant advantage.

### 3.1.4 Using Filler Packets to Sustain Transmission

A node determines that a channel is not in use when it detects no packet flowing through it on that channel for a short period of time. Consider however, nodes  $g$ ,  $h$ ,  $i$ , and  $j$  connected as shown in Figure 6. Say that node  $g$  has captured channel  $j$  but after a few packets, several contiguous packets are corrupted when they are being sent from  $g$  to  $h$ .

If the general local policy is not to retransmit corrupted data, then  $h$ , detecting an error will not retransmit the packet from  $g$ . However, node  $i$  may now mistakenly think that channel  $j$  is no longer in use and initiate transmission to node  $j$ . To avoid this problem,  $h$  inserts filler packets for  $g$  whenever it receives corrupted packets on channel  $j$ . That is, if a packet is corrupted, a filler packet is forwarded in its place so that the channel remains captured.

### 3.1.5 Choosing Protocol Parameters

Assuming Go-Back-N is used, the important protocol parameters are the window size  $N$ , the time-out value  $t_{out}$ , and the *channel-captured* signal time-out  $t_{cout}$ .

Let  $RTT$  be the round trip time and assume  $RTT = t_{roundtrip}$  as defined earlier. To increase channel utilization as much as possible,  $N$  should be chosen as follows:

$$N = \lceil RTT/t_{pkt} \rceil \quad (2)$$

The Go-Back-N time-out parameter  $t_{out}$  should be set such that  $t_{out} \geq RTT$ , but not too large since if  $t_{out}$  is too large, packets that are corrupted will not be quickly retransmitted and this could adversely affect performance. If  $t_{out}$  is set too small (but still  $\geq RTT$ ), then occasional congestion on the return (control) channel may cause ACKs to be delayed, triggering the sender to retransmit packets that otherwise need not be retransmitted. Thus the optimal  $t_{out}$  highly depends on traffic patterns (traffic source models) and channel error rates. However, assuming that channels used by ACK signals are lightly loaded, and channels are error prone, we can choose  $t_{out} = RTT + \delta$  where  $\delta$  is a small fraction of  $RTT$ .

Proper setting for the *channel-captured* signal time-out value,  $t_{cout}$ , is less obvious. It is highly dependent on channel characteristics (especially error rates). Assuming that separate control channels are used for signals and the *channel-captured* signal is embedded in every ACK, suppose the data channel is such that its probability to corrupt  $k$  packets in a row is negligible, and the control channel is such that its probability to corrupt  $l$  control packets in a row is negligible, then  $t_{cout}$  can be set as follows:

$$t_{cout} = k \times l \times t_{pkt} + RTT \quad (3)$$

### 3.2 Connection-Based Transmission

An alternative to a packet-based protocol is connection-based transmission (CBT). Assume that associated with each VCSEL channel there is logic which has the ability to pass the incoming data stream directly from an input detector to an output VCSEL. If the detector is connected to the VCSEL, the node is said to be in *repeater mode*, otherwise (if data to the VCSELS are coming from the processor) it is said to be in *insertion mode*. Note that even though data are transmitted in a connection-oriented manner, control signals are still transmitted as packets via the control channels, are buffered by intermediate nodes, and transmitted in a store-and-forward manner.

Consider Figure 6 and suppose node  $h$  wants to transmit a message to node  $j$ . Then using CBT, the VCSEL-detector pairs of node  $i$  that are assigned to channel  $j$  must be set to operate in the repeater mode. These VCSEL-detector pairs must remain in the repeater mode until  $h$  finishes transmitting the message. Afterwards, node  $i$  needs to know that  $h$  is no longer transmitting, thus allowing node  $i$  to revert to insertion mode should it need to send a message to  $j$ . Configuring intermediate nodes between the sender and the receiver so that they remain in the repeater mode is called the *setup* process. This corresponds to setting

up a circuit in a circuit-switched communications system. Releasing the intermediate nodes from this commitment is called the *teardown* process (e.g., releasing the circuit) and a protocol similar to that described in [3] can be utilized.

Note that the *setup* process in CBT is analogous to the *capture* process in PBT. Similarly the *teardown* process in CBT is analogous to the process of nodes releasing a channel when there are no packets flowing through it in PBT. Indeed, with small packet size and large enough message size, the two approaches have similar performance characteristics.

## 4 Performance Analysis

In the analysis presented below we model each channel as an independent queueing system. Except where indicated, we assume error-free communication and the use of separate control channels as described earlier. We define the following symbols and quantities to aid our analysis:

- $BW$ : *Channel bandwidth*. For example, if a channel consists of eight parallel, active VCSEL-detector pairs each operating at 1 Gbps, then  $BW = 8$  Gbps. We assume all data channels have the same  $BW$ .
- $\lambda$ : *Mean node message generation rate*. The analysis assumes that all nodes have the same  $\lambda$ .
- $\bar{l}$ : *Mean message length* measured in bits.
- $l_{pkt}$ : *Packet length* measured in bits (defined only when packet-based transmission is used).
- $l_{sig}$ : *Control signal length* measured in bits.
- $t_{pkt}$ : *Packet time*, the time to send a packet from a node to its immediate neighbor;  $t_{pkt} = l_{pkt}/BW$ .
- $t_{sig}$ : *Signal packet time*, the time needed to send a control signal packet via the control channel from a node to its immediate neighbor. Assume that  $t_{sig} = t_{pkt}$  for simplicity.
- $t_{ins}^m$ : *Message insertion time* for message  $m$  which is  $l$  bits in length,  $t_{ins}^m = l/BW$ .
- $RTT_{pkt}$ : *Packet-based transmission round trip time*, the time between the sending of the first bit of a packet and the complete receipt of the acknowledgement for the packet using PBT.

### 4.1 Message Generation and Message Size Assumptions

The following assumptions are made concerning the source traffic model used in subsequent analyses:

- **Message Size.** Messages are long such that the message insertion time is large compared to the round trip times (i.e.,  $t_{ins}^m \gg RTT_{pkt}$  for all  $m$ ).

- **Message Generation.** Assume messages are generated in a Poisson process with a mean rate of  $\lambda$ . Each node's message generation process is independent. Message destinations are uniformly distributed across all nodes except the source node.

#### 4.1.1 The Validity of the Message Size Assumption when Using PBT

Consider an  $n$  node ring. Using the packet-based transmission (PBT) approach, a data packet sent from node  $i$  to node  $j$  takes  $(j - i) \bmod n$  hops to reach  $j$  with each hop taking  $t_{pkt}$ . The acknowledgement from  $j$  to  $i$  takes  $(i - j) \bmod n$  hops to reach  $i$  with each hop taking  $t_{sig}$ . Since we assume  $t_{sig} = t_{pkt}$ ,  $RTT_{pkt} = n \times t_{pkt}$  in this case.

Assuming a 32 node ring (i.e.,  $n = 32$ ) and  $BW = 8$  Gbps using a 64 byte packet size ( $l_{pkt} = 64 \times 8$ ), we see that  $t_{pkt} = 64 \times 8 / 8 \text{ Gbps} = 64 \text{ ns}$ .  $RTT_{pkt} = 32 \times 64 \text{ ns} = 2.048 \mu\text{s}$ . At 8 Gbps, 2.048  $\mu\text{s}$  corresponds to the time needed to insert 256 bytes into the network. A modestly sized 64 kilobyte (KB) message  $m$  requiring  $t_{ins}^m = 65536 \times 8 / 8 \text{ Gbps} = 65.5 \mu\text{s} = 32 \times RTT_{pkt}$  satisfies the message size assumption.

The significance of the message size assumption is that it permits performance analysis to proceed without accounting for the effects of round trip times and control signal transmission times. We have shown that even with messages as small as 64 KB, the round trip times and control signal transmission times accounts for less than 1/10 the message insertion time. With a message size of 0.5 MB, the round trip times and control signal transmission times will account for less than 1/100 the message insertion time. Although not considered here, a similar analysis can be done for the CBT protocol which verifies the message size assumption in that case.

## 4.2 Performance Analysis Using Markovian Queueing Model

Since message destinations are assumed to be uniformly distributed to all nodes except the source, a given node will generate messages for another node at a rate of  $\lambda / (n - 1)$ . Since  $n - 1$  nodes are each generating messages at this rate for each node, the aggregate generation rate of messages for a channel is  $\lambda$ .

We further define the following:

- $t_{msg}$ : *Average message service time*, defined as  $t_{msg} = \bar{l} / BW$  (i.e., the average time it takes to send a message).
- $\mu$ : *Channel service rate or capacity*,  $\mu = 1 / t_{msg}$ .

The  $n$  node,  $n$  channel unidirectional ring can be modeled by viewing each channel as a resource or server being accessed by  $n - 1$  nodes with each channel being treated as a single server queueing system (i.e., M/G/1 system). Consider first how often an arriving message would find the channel it requires is occupied (busy). If the channel is in use, the newly arrived message can queue up and wait for its turn to use the channel. Note that  $\lambda < \mu$  for a stable system. For notational convenience, we define  $\rho = \lambda / \mu$  as is commonly done in the queueing theory literature.

We focus on the case where a message queues up at its source when its intended channel is in use and assume that each node maintains different queues for messages with different destinations.<sup>4</sup> Message lengths conform to an exponential distribution or are constant.

Let  $L$  be the steady state aggregate queue size in terms of number of messages for a channel, and let  $W$  be the average time between a message being generated and being delivered (i.e., message system time or waiting time). With exponential message lengths (i.e., an M/M/1 system) standard queueing theory results give  $L$  and  $W$  as:

$$L = \frac{\lambda}{\mu - \lambda} \quad (4)$$

$$W = \frac{1}{(\mu - \lambda)} \quad (5)$$

If message length is constant, then the system becomes an M/D/1 system with:

$$L = \frac{\rho^2}{2(1 - \rho)} \quad (6)$$

$$W = \frac{2 - \rho}{2\mu(1 - \rho)} \quad (7)$$

Note that while none of the above equations appear to depend on the number of nodes  $n$ , the channel capacity  $\mu$  is a function of  $n$ . With a given system, the larger  $n$  is, the fewer VCSEL-detector pairs get assigned to individual channels, thus effectively reducing  $\mu$ . Furthermore, as the ring gets larger, the roundtrip time becomes more significant and the message size assumption becomes less valid. A more complex approach will be needed to analyze large rings.

## 4.3 Performance Analysis With Channel Error

With error free channels,  $t_{msg} = \bar{l} / BW$ , however, if errors are taken into account this analysis has to be modified. Define the error probability  $p_{pkt}$  as the

<sup>4</sup>This is referred to as *Virtual Output Queueing* [8].

probability that *either* a packet (or frame) *or* its corresponding acknowledgement signal is corrupted.

Consider the Go-Back-N protocol discussed earlier. Let  $N$  be the window size and  $t_{out}$  be the time-out parameter. As shown earlier, the Go-Back-N protocol is efficient when  $N$  and  $t_{out}$  are chosen such that:

$$N t_{pkt} = RTT_{pkt} = t_{out} \text{ for PBT} \quad (8)$$

Define *channel efficiency*,  $\eta_{chl}$ , to be the fraction of time a channel spends sending *new* packets. If all the channels are error free, then no packet will ever need to be resent and  $\eta_{chl} = 1$ . If the channel spends half its time resending packets, then  $\eta_{chl} = 0.5$ . If Equation 8 is satisfied, channel efficiency for PBT can be calculated as:

$$\eta_{chl} = \frac{t_{pkt}}{t_{pkt} + \left(\frac{p_{pkt}}{1-p_{pkt}}\right)t_{out}} \quad (9)$$

We see that as the error probability  $p_{pkt}$  approaches zero,  $\eta_{chl}$  approaches one; and as  $p_{pkt}$  approaches one,  $\eta_{chl}$  approaches zero. The derivation of Equation 9 can be found in [1].

The error probability  $p_{pkt}$  is related to the distance the data and acknowledgements need to travel. For simplicity, we assume that each point-to-point link on an  $n$  node ring has a bit error rate of  $p_{bit}$  and the errors are random. Assume also that each node is equally likely to send to any other node except itself, then the error probability  $p_{pkt}$  can be calculated as:

$$p_{pkt} = \frac{1}{n-1} \sum_{i=1}^{n-1} 1 - (1 - p_{bit})^{i \cdot l_{pkt} + (n-i) \cdot l_{sig}} \quad (10)$$

Using the above results, Figure 7 plots the packet-level error probability  $p_{pkt}$  as a function of the bit error rate  $p_{bit}$  for various ring sizes. For this plot, the packet length,  $l_{pkt}$ , is set equal to 64 bytes, and the control signal length,  $l_{sig}$ , is set equal to 4 bytes. As expected, the packet error probability increases with the bit error probability and with the number of nodes. Though not shown here, similar results can be obtained by plotting  $p_{bit}$  as a function of  $l_{pkt}$  and  $l_{sig}$ . Doubling the packet and control signal sizes has the same effect on packet error probability as doubling the ring size.

The queueing expressions presented earlier can be applied by derating the service rate  $\mu$  by a factor of  $\eta_{chl}$  to take into account the lower available bandwidth

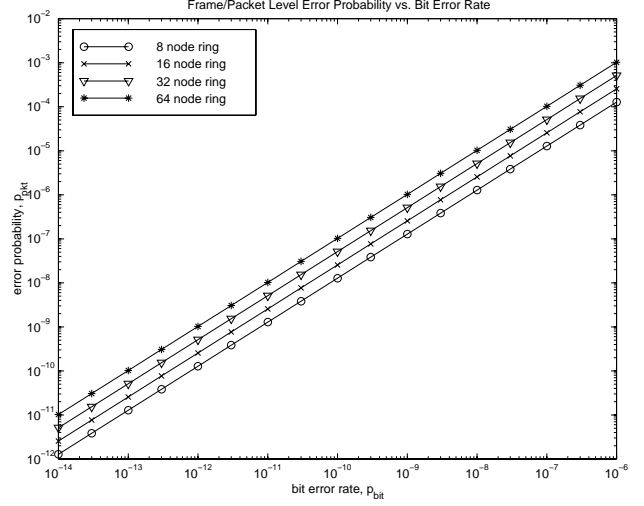


Figure 7: Error probability plot.

which results from errors. For the M/D/1 case we have:

$$\hat{\mu} = \mu \cdot \eta_{chl} \quad (11)$$

$$\hat{\rho} = \lambda / \hat{\mu} = \lambda / (\mu \cdot \eta_{chl}) \quad (12)$$

$$\hat{L} = \frac{\hat{\rho}^2}{2(1-\hat{\rho})} \text{ for M/D/1} \quad (13)$$

$$\hat{W} = \frac{2-\hat{\rho}}{2\hat{\mu}(1-\hat{\rho})} \text{ for M/D/1} \quad (14)$$

By substituting Equation 10 for  $p_{pkt}$  into the expression for channel efficiency, Equation 9, one can obtain channel efficiency as a function of bit error probability. Figure 8 plots channel efficiency  $\eta_{chl}$  vs. bit error rates  $p_{bit}$  for various ring sizes. Figures 7 and 8 assume a packet insertion time of  $t_{pkt} = 64$  ns (equal to the time to send 64 bytes over an 8 Gbps channel) and  $t_{out} = n \times 64$  ns. As expected, the channel efficiency decreases as the number of nodes increases. This is due to the larger number of hops a packet and its ACKnowledgement must go through and the resultant higher packet error probabilities. Additionally, the plots indicate that for large bit error rates, channel efficiency declines exponentially. The results indicate that bit error rates on the order of  $10^{-11}$  or greater will result in a large decrease in performance.

The mean message system time  $\hat{W}$  may also be obtained as a function of the bit error rate by in turn substituting appropriately from Equations 9 to 12 into Equation 14. Note that to evaluate  $\hat{W}$ , a value of  $\lambda$ , the message arrival rate, must be determined. The arrival rate  $\lambda$  is, of course, a function of the application being executed on the system.

Consider, for example, a (fictitious) imaging application where the nodes process data sent in the form of

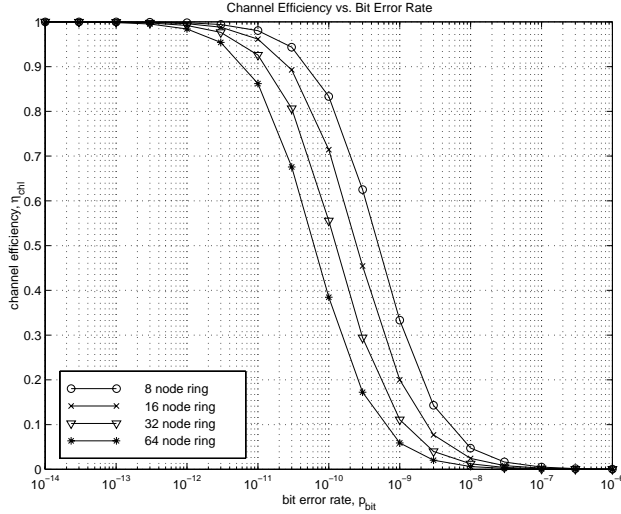


Figure 8: Channel efficiency plot.

large two dimensional arrays. Let each 2-D array contain  $256 \times 256$  single precision floating point numbers that represent an image. Assume that each node (processor) processes the image it has received and generates as the result another same-sized image. This new image is in turn sent to another processor. Since each array has 65,536 elements in it, and each element is 4 bytes in length, successive image arrays are sent as 256 KB messages.

Suppose each element requires on average 5 clock cycles of processing. Assuming that the processor's local memory poses no bottleneck, then it takes roughly 327,680 clock cycles for a processor to generate a new array. This new array is then sent as a new message. If the processor is clocked at 500 MHz, 327,680 clock cycles translate to about  $655 \mu\text{s}$ . This then roughly is the time between two successive messages being generated by a processor. Thus, the message generation rate is  $\lambda = 1/655 \mu\text{s} = 1526$  messages per second.

Given  $\lambda$  we can now plot  $\bar{W}$  versus  $p_{bit}$  as shown in Figure 9. As indicated from the curves, the message system time remains relatively constant until the bit error rate goes above about  $10^{-11}$ . For certain applications, message delay may impact performance. For example, suppose we wish to bound the average message system time to 0.4 ms, then a higher effective channel capacity is required which in turn leads to a stricter bit error rate requirement. For example, were we to use an ring with 32 nodes, a bit error rate less than  $10^{-11}$  would be necessary to satisfy the 0.4 ms mean message system time requirement. If 64 processors were needed, then a bit error rate on the order of  $10^{-12}$  or less would be required. In this manner, tradeoffs between applica-

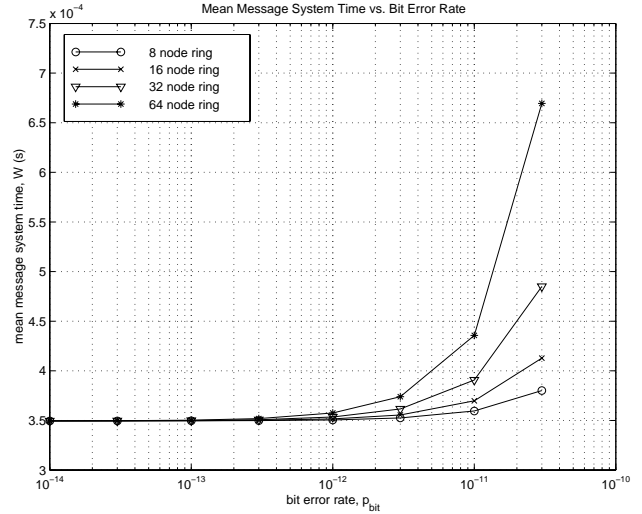


Figure 9: Mean message system time plot.

tion requirements, node processing speed, the bit error rates, channel capacity, and total number of nodes can be examined.

## 5 Design Refinements

The protocols described above do not provide for arbitration. It is possible that a node with a message to send to a channel, say  $j$ , may never manage to capture  $j$  when multiple other nodes also have messages for  $j$ . To avoid this starvation situation, we propose an arbitration mechanism for use in the multiring.

While it is often preferable that an arbiter allow fair access to all the nodes that contend for a channel, this may be application dependent. For example, it may be desirable to give a subset of nodes more access to a channel than other nodes (e.g., one of the nodes may be attached to a sensor bank that transmits massive data to other compute nodes). One approach that suits the multiring is to use a *Deficit Round Robin (DRR)* scheduler [9] as the arbiter. The DRR scheduler has the following attractive characteristics, details of which can be found in [9]:

- Flexibility. Nodes can be given different amounts of access to a channel by assigning different *quantum* values to the nodes for that channel.
- Fast Decision Making. DRR algorithm is fast since it needs to only examine the node in question to decide whether it should be given access.
- Fairness. DRR has been proven to be fair to the following extent: at any time, assuming that all parties have continuously contended for a channel and all were designated to have equal access, the

difference in the amount of access granted to the most advantaged contender and the most disadvantaged contender is no more than three times the maximum message size.

The following describes how DRR can be adapted to arbitrate traffic in a multiring. Since every channel is attached to a particular receiver, we assign the DRR scheduler for that channel to its associated receiver. That is, for a given channel  $j$ , the DRR scheduler for channel  $j$  will run on node  $j$ .

In an  $n$  node ring, every DRR scheduler will keep  $n - 1$  deficit counters. Each deficit counter is uniquely associated with a potential channel user. In the general case, the DRR at node  $j$  will keep deficit counters for all nodes except  $j$ . A deficit counter keeps track of how much access a potential channel user can have at the moment. Each counter is initialized to zero.

Prior to sending data messages on channel  $j$ , each node wanting channel  $j$  sends a short control signal, *request*( $i, j, x$ ), indicating the size of the message it would like to send. After sending a control signal the node waits. Node  $j$  queues up all these requests in a *request queue*. When the request queue becomes non-empty, the receiver node  $j$  will dequeue a request, determine which node sent it, and add to the node's deficit counter its associated quantum. Node  $j$  then compares the message size to the node's deficit counter. If the message size is less than the deficit counter,  $j$  will grant channel access to the node and subtract the message size from the contender's deficit counter. Node  $j$  then sends a *grant* signal to the successful node, say node  $i$ . Node  $i$ , upon receipt of the *grant* signal, transmits its message to  $j$ . At the end of the transmission,  $i$  will send its next *request* signal to  $j$ .

Node  $j$ , upon receipt of the new message request from  $i$ , determines its size and then compares the new message size with  $i$ 's deficit counter. If  $i$ 's request now exceeds its deficit counter,  $j$  inserts this new request at the end of the request queue. Node  $j$  then proceeds to process the next request in the request queue. If  $i$ 's request doesn't exceed its deficit counter, then  $j$  proceeds as described in the preceding paragraph.

If after sending a message,  $i$  no longer has any message to send to  $j$ , then  $i$  simply indicates so by sending a zero message size request. Node  $j$  will then reset  $i$ 's deficit counter to zero and discard  $i$ 's request. If, at a later time,  $i$  wants to transmit to  $j$  again,  $i$  simply sends a new request to  $j$ .

## 6 Conclusions and Future Research

Recent advancements in VLSI photonics and related optics technologies have made possible the de-

sign of terabit interconnection networks. Based on these technologies, we have proposed a feasible design of such a high performance interconnection network. We have shown that our design can accommodate both the packet-based and connection-oriented transmission paradigms. We have also shown how DRR can be adapted to arbitrate for both transmission paradigms in our system. Additionally, we have provided an initial performance evaluation of such a system and have indicated the design parameters which must be considered when determining whether an application will be suitable for the system in question.

Even though our system can achieve multiple terabits per second performance, we have not fully exploited the vast amount of bandwidth made available. Using the multiring, we divided the bandwidth into channels. This approach does not allow us to statistically multiplex data traffic. In the future, other architectural alternatives that allow statistical multiplexing will be considered.

## References

- [1] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, second edition, 1992.
- [2] G. C. Boisset et al. Optomechanics for a four-stage hybrid-self-electro-optic-device-based free-space optical backplane. *Applied Optics*, 36:7341-7358, 1997.
- [3] R. Chamberlain, M. Franklin, R. Krchnavek, and B. Baysal. Design of an optically-interconnected multicomputer. In *Proc. of 5th Int'l Conf. on Massively Parallel Processing Using Optical Interconnections*, pages 114-122, June 1998.
- [4] Message Passing Interface Forum. MPI: A message passing interface. In *Proc. of Supercomputing*, pages 878-883, 1993.
- [5] M. Marsan et al. All-optical WDM multi-rings with differentiated QoS. *IEEE Communications Magazine*, pages 58-66, February 1999.
- [6] M. Marsan et al. Daisy: A scalable all-optical packet network with multi-fiber ring topology. *Comp. Networks and ISDN Sys.*, 30:1065-1082, 1999.
- [7] D. V. Plant et al. A 4x4 VCSEL/MSM optical backplane demonstrator system. *Applied Optics*, 35:6365-6368, 1996.
- [8] Balaji Prabhakar and Nick McKeown. On the speedup required for combined input and output queued switching. Technical Report CSL-TR-97-738, Stanford University, November 1997.
- [9] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM '95*, pages 231-243, August 1995.