Evaluating the Performance of Photonic Interconnection Networks

Roger Chamberlain, Ch'ng Shi Baw, Mark Franklin, Christopher Hackmann, Praveen Krishnamurthy, Abhijit Mahajan, and Michael Wrighton

Computer and Communications Research Center
Washington University, St. Louis, Missouri

ABSTRACT

This paper describes the design and use of the Interconnection Network Simulator (ICNS) *framework. ICNS is a modular, object-oriented simulation system that has been developed to investigate performance issues in multiprocessor interconnection networks that exploit photonic technology in their design. We describe the ICNS infrastructure, present two distinct photonic interconnection networks that have been modeled using ICNS, and give performance results for each of these networks.*

1 Introduction

With the advent of optical fiber, photonic technology has become an indispensable component in the design and deployment of the world's long-distance communications infrastructure. The bandwidth capacity of long-distance fiber links is enormous, and the technical and economic advantages of photonic technology in this arena are indisputable. What we haven't yet seen is the exploitation of photonic technology for short-distance communications (e.g., chip-to-chip and board-to-board links within a single system). The reasons for this are multifold. Generally, however, although photonic interconnection networks have significantly increased bandwidth, the complexity and cost of such systems, coupled with the inability of processor interfaces to cope with high photonic data rates, usually negates any expected bandwidth advantages. It is a misconception that merely replacing an existing electronic interconnect with an optical fiber equivalent will result in a viable architectural design. To truly take advantage of photonic technology, the total system design must be rethought with an understanding of the strengths and weaknesses of photonics.

[1]Research reported herein is supported in part by NSF grant MIP-9706918 and DARPA contract DAAL01-98-C-0074.

The Interconnection Network Simulator (ICNS) framework was developed to help evaluate candidate architectural alternatives that exploit photonic technology in their design. Our specific interest is the use of photonics in the processor-to-processor interconnection network that is an integral part of any multicomputer system. The design of ICNS was not limited to this application, however, and we have modeled both multicomputer systems and switching fabrics for internet routers.

The use of photonic technologies as building blocks for multicomputer interconnects is not, as yet, a well-studied subject. Photonics posses strengths and weaknesses different from electronics from an interconnection network standpoint. Thus, fundamental design space parameters such as slotted-time versus asynchronous transmission, buffered versus unbuffered switching, packet-based versus message-based transport, etc. need to be reconsidered when designing a photonic interconnect.

ICNS was developed using the MODSIM III language in a modular, object-oriented manner. It is designed to be extended as new architectures are proposed and new performance questions are raised. For this reason, the simulation framework must be flexible enough to allow various network components with varied characteristics to be modeled faithfully.

As applications demand higher performance from the interconnection network, the design of the network will be more closely guided by the specific targeted applications. Hence it is important to consider both the application and the interconnection network together in the design process. ICNS takes special care to ensure that application-level simulation modules can be easily integrated with the interconnect modules. This allows application issues to be fully explored in tandem with interconnect issues in the design process.

At a very high level, an interconnection network

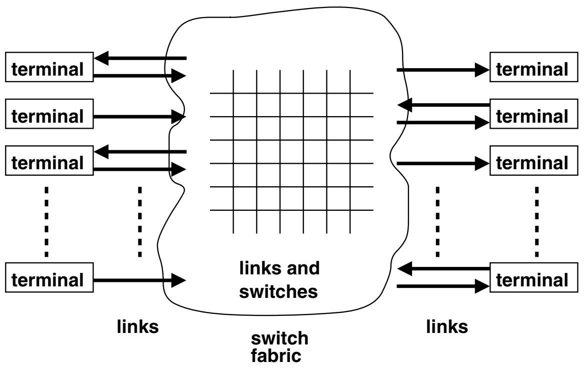


Figure 1: A Generic Interconnection Network.

can be abstracted as a system composed of terminals that generate and consume messages, and links and switches that facilitate the transportation of messages from one terminal to another. Figure 1 shows a generic interconnection network.

To achieve the desired flexibility and extensibility, a modular, object-oriented approach is adopted as the principle design and development methodology for the simulator. For example, a component that models a simple bufferless switching element can be enhanced to model a switching element with a simple FIFO buffer. The FIFO buffer component can be easily extended to model a prioritized multiqueue buffer. A switching element with prioritized multiqueue buffer can further be enhanced to model a switching element with scheduling capability. The scheduler module can be easily modified to perform scheduling using different policies.

We have used ICNS to model a pair of systems. The first is the *Gemini* interconnect, a parallel photonic and electronic network that utilizes lithium niobate optical switches to construct a circuit-switched high-bandwidth data path in the switching fabric. The second is a photonic multiring interconnect, in which 2-D arrays of Vertical Cavity Surface Emitting Lasers (VCSELs) and photodetectors are used to provide high-bandwidth I/O to/from CMOS chips. The variety in photonic technologies used, as well as the distinct architectures that result, point to the flexibility of the ICNS framework.

Section 2 describes the ICNS implementation environment, as well as the base classes and basic object types that form the core of ICNS. Section 3 describes the model used to simulate such objects as links, switches, processing nodes, etc. Section 4 describes the usage of the simulator. Section 5 provides a description of the architectures simulated to date using ICNS, including a description of the photonic components that are enabling technology for these architectures. It also gives some performance results that have been derived using ICNS. Section 6 concludes and describes the future plans for ICNS.

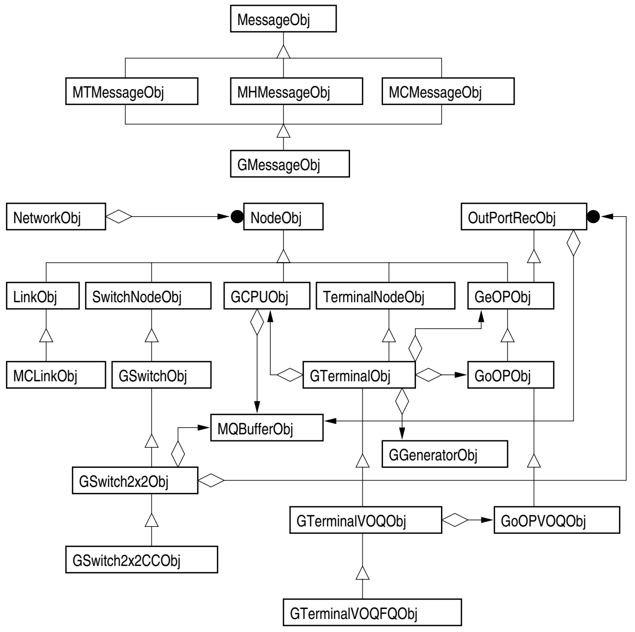


Figure 2: ICNS partial class diagram.

2 ICNS Implementation Environment and Base Classes

2.1 The MODSIM III Implementation Environment

ICNS is implemented using the MODSIM III language developed by the CACI Products Company. The MODSIM III runtime environment is provided in the form of a shared runtime library. The MODSIM III compiler takes in MODSIM III source code, compiles it into C++ code, and then uses the system C/C++ compiler and linker to compile the C++ code and link the resulting object code with the runtime library to make an executable program.

MODSIM III is a modular, block structured language that supports object oriented programming. Its runtime environment provides an implicit event queue and the binding of events to event handlers. Simulation time progression and event scheduling are implicitly handled by the runtime environment. Using MODSIM III, ICNS is built in an object-oriented fashion. Subclassing and object composition are used extensively. This will be shown in subsequent sections.

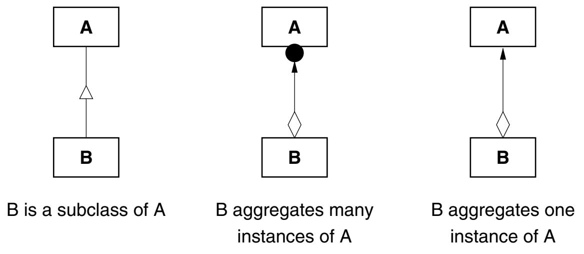

A
B
B is a subclass of A
A
B
B aggregates many instances of A
A
B
B aggregates one instance of A

Figure 3: Brief nomenclature for the symbols used above in Figure 2.

link source address	link destination address
network source address	network destination address
sequence number	length
message type	channel
time stamp	
data	

Figure 4: The GMessageObj.

2.2 ICNS Base Classes

ICNS is primarily composed of the following base classes:

- MessageObj
- NodeObj
- NetworkObj

Of the three classes listed above, interactions between the MessageObj and the NodeObj generate the basic events that move the simulation forward. Figure 2 shows a class diagram using OMT-notation. The symbols used in Figure 2 are explained briefly in Figure 3. Only the classes that more directly model particular network or multicomputer components are shown.

2.2.1 MessageObj

MessageObj models messages in the system. The base MessageObj class allows source and destination addresses, time of creation of the message, and the length of the message to be recorded. There are also a basic set of interfaces that facilitate interactions between messages and NodeObj derivatives. The interaction mechanism will be described later in Section 2.3.

MessageObj has been extended to allow specification of message types, communication channels being used, and link level source/destination addresses via the MTMessageObj, MCMessageObj, and MHMessageObj subclasses respectively. For example, the GMessageObj subclassed from all the MessageObj classes mentioned can model the message shown in Figure 4.

MessageObj can also model control signals that help control the network. This is done by assigning reserved type values to messages that the nodes (e.g., switches and terminals) recognize as control signals.

To allow the ICNS to be specialized to the application, MessageObj also has a reference to a general object type that may be of any class specified by the user (shown as 'data' in Figure 4). This allows the user to attach application specific information to a MessageObj. Also, in the event that a complex multi-layered transmission protocol is chosen, the 'data' reference can be used to let one MessageObj encapsulate another MessageObj. This enables simulation of layered protocols where data units are encapsulated at different protocol layers.

2.2.2 NodeObj

NodeObj models anything that accepts a MessageObj and later passes the MessageObj to some other entity (such as another NodeObj). Each instance of a NodeObj has an ID and a set of interfaces that interacts with MessageObj. NodeObj by itself provides no useful function except to mandate the minimal set of interfaces that allows any MessageObj subclass to interact with any NodeObj subclass in a consistent manner.

For example, MessageObj needs to interact with switches, links, and terminals. Since the classes that model switches, links, and terminals are subclasses of NodeObj, a MessageObj can interact with any one of them using the same interfaces. Regardless of whether a MessageObj is interacting with a LinkObj, SwitchNodeObj, or GCPUObj, the MessageObj uses the same interfaces described in Section 2.3. Since these classes are all subclasses of NodeObj as shown in Figure 2, they provide MesageObj with a consistent set of interfaces.

2.2.3 NetworkObj

NetworkObj is a container object that holds the NodeObj's. NetworkObj provides a single point-of-entry for user programs to access a particular NodeObj as well as keeping track of the number of terminals and switches in the system.

2.3 Interactions between MessageObj and NodeObj

The interconnection network is used to send messages. This is modeled as messages being passed from one node to another in ICNS. The following describes how MessageObj and NoteObj interact.

When a MessageObj is passed to a NodeObj, it first asks if the NodeObj can process it immediately. If the reply is negative, the MessageObj asks the NodeObj to place it in the NodeObj's buffer. Otherwise, the MessageObj asks the NodeObj how long it will take to process the MessageObj. The MessageObj will wait that amount of time and then ask the NodeObj to finish up the processing.

When asked by a MessageObj whether the NodeObj can process the MessageObj immediately, the NodeObj gives a simple *yes* or *no* answer. When asked to buffer a MessageObj, the NodeObj can buffer the MessageObj as requested, or discard the MessageObj should the NodeObj not have a buffer or have a full buffer. When asked how long it will take to process a MessageObj, the NodeObj returns a real number representing the amount of time needed to process the NodeObj. For example, if the NodeObj concerned is a link, then processing a MessageObj requires sending the MessageObj through the link. The processing time is simply the link delay. When asked to finish processing a MessageObj, the NodeObj usually either discards the MessageObj or passes the MessageObj to another NodeObj.

3 Description of Selected Objects in a Simulated Interconnection Network

This section describes the formulation of specific objects in an interconnection network simulated using ICNS. The selected objects are as follows:

- Links
- Terminals
  - Message Generator
  - Buffer
  - Central Processing Unit (CPU)
- Switches

3.1 Links

Links simply accept messages from a node and pass the messages to another node. A simple link has a fixed bandwidth, a destination, and a propagation delay parameter. The destination of a link is usually a switch node or a terminal. A simple link is modeled by the LinkObj and is depicted in Figure 5. The LinkObj, shown in the middle left of Figure 2, is a subclass of the NodeObj.

The MCLinkObj, a subclass of LinkObj, extends the LinkObj to model multichannel links. MCLinkObj

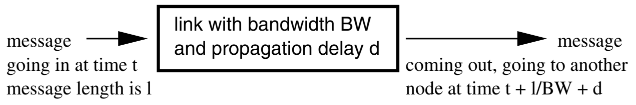


Figure 5: A simple link.

is suitable for modeling such objects as frequency division or spatial division multiplexed links. The multichannel link model is depicted in Figure 6.

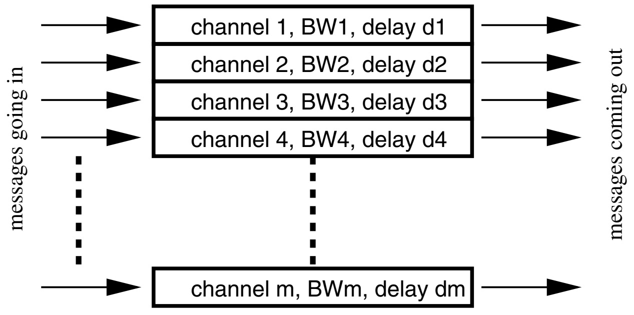


Figure 6: A multichannel link.

Channel error can also be modeled by specifying an error probability parameter. Every time a message enters a channel, it is dropped or marked corrupted with the error probability specified.

3.2 Terminals

Terminals generate and consume messages. A processing node, for example, can be modeled as a terminal. When modeling signal processing applications the sensor banks can also be modeled as terminals (i.e., terminals that generate but do not consume messages). For example, a terminal modeling a processing node would have a construction similar to that shown in Figure 7 while a terminal modeling a sensor bank would look like that shown in Figure 8.

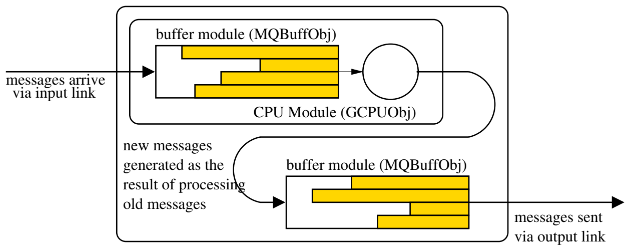

buffer module (MQBuffObj)
messages arrive via input link
CPU Module (GCPUObj)
new messages generated as the result of processing old messages
buffer module (MQBuffObj)
messages sent via output link

Figure 7: A processing node model.

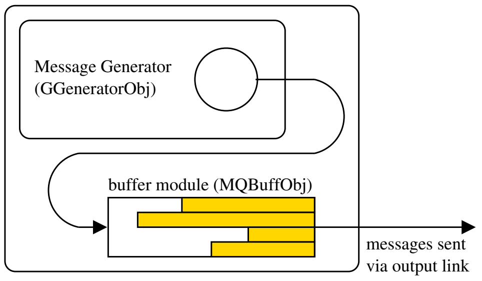


Figure 8: A sensor bank model.

3.2.1 Message Generation

The GGeneratorObj is implemented to model a message generator. Messages can be generated according to a Poisson process with user specified rates. Message lengths can be exponentially distributed or fixed. Support for generating messages according to other statistical models (such as Gaussian, Gamma, Erlang, Weibull, etc.) can be easily added. An important class of applications, *Space Time Adaptive Processing (STAP)*, generate messages that are either very large (on the order of hundreds of kilobytes (KB) or larger) or very small (on the order of tens or hundreds of bytes) [1]. This type of traffic can also be simulated using GGeneratorObj.

GGeneratorObj is used within a terminal since only terminals can generate messages. After a message has been generated, it may not be possible to send the message immediately because of network contention. In this case, a message needs to be buffered. Buffering is simulated using the MQBuffObj described in the following section.

3.2.2 Buffer

Buffering is an important aspect of interconnection network design. MQBuffObj implements a flexible multi-queued buffer module so that various buffering techniques and policies (such as prioritized queue, virtual output queues, fair queueing, etc.) can be simulated. MQBuffObj does not implement queueing policies but merely provides basic enqueue/dequeue, state reporting, and queue management functions to other classes that make use of it. For example, GTerminalObj's output ports (modeled using the GeOPObj and GoOPObj) use MQBuffObj as their buffering modules. Buffering policy is implemented in GeOPObj and GoOPObj.

When a message arrives at a switch node, that message may not be processed immediately because some other messages are being processed. In this case, the incoming message can be buffered in MQBuffObj.

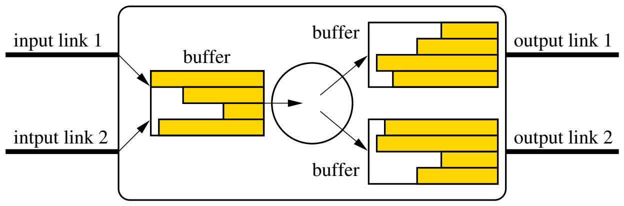


Figure 9: A 2×2 switch.

3.2.3 Central Processing Unit (CPU)

To facilitate application-level simulation, the GCPUObj class is provided. GCPUObj accepts and processes messages and keeps such statistics as CPU utilization. MQBuffObj is used by GCPUObj to buffer messages that arrive faster than it can process. GCPUObj may also generate new messages in response to messages received. GCPUObj can be subclassed to simulate specific applications in conjunction with attaching application specific data to the MessageObj.

We note that GCPUObj performs application-level simulation only and does not simulate data transmission protocols. Simulation of transmission protocols is performed by the terminal node itself.

3.3 Switches

At the heart of the switching fabric of a network are the switches. Generally, an $n \times m$ switch accepts an incoming message from any of the n inputs, determines which of the m outputs the message should be sent to using the message's header information, and sends the message to the output. The GSwitchObj provides a framework upon which a general $n \times m$ switch capable of performing packet- as well as circuit-switching can be built.

Routing a message to its output destination is a process that takes time. In the event that message arrival outpaces the routing process, the incoming messages are buffered. In addition, the rate at which messages are routed to the output may also outpace the rate at which messages can actually be sent out on the output link. Messages are also buffered in this case. Again, MQBuffObj can be used to maintain the buffers.

Figure 9 shows a 2×2 switch with a centralized input buffer and separate output buffers. The GSwitch2x2CCObj (a subclass of SwitchNodeObj) implements a switch similar to that depicted in Figure 9.

4 ICNS Usage

Previous sections have described various objects provided by ICNS that can be used to simulate various components in an interconnection network. These components need to be connected together to form a network. ICNS provides a procedure that can be used to build a network using the available components. The procedure, called `BuildGNetwork`, reads in a plain text file that describes the topology of the desired network. The procedure instantiates component objects as needed, and connects them according to the topology specified.[1]

Each component has various parameters that need to be specified. For example, the `BuildGNetwork` procedure needs to be told what types of terminals, switches, and links it should instantiate as it processes a topology descriptor file. Each terminal needs to be told at what rate it should generate messages, what the messages' length distribution should be; what are the link bandwidths, etc. Thus ICNS provides the ParamObj class that reads in a list of parameter values from a text file. The parameter values are then used by the `BuildGNetwork` procedure to configure the relevant components as they are instantiated. The parameter descriptor file is simply a list of parameter name-value pairs.

Using the ParamObj and `BuildGNetwork` facilities, a simple top level program can be built to accept two file names (one parameter file and one topology file), use ParamObj to register all parameter values (in the parameter file), then use the `BuildGNetwork` procedure to build a network according to the topology file, start simulation, and finally, collect and display statistics from various components at the end of simulation.

To support both the verification of the simulation models as well as improved understanding of the operation of modeled systems, a set of visualization tools has been developed. The visualization tools are driven from a static topology description file and trace data derived from the simulation execution.

The topology description defines the structure of the network: terminal objects, switching node objects, queues within terminals and switching nodes, and links between objects. Links (both electrical and optical) form connections between the terminals and the switching nodes. Within each of the terminals and switching nodes, the message queues are represented graphically. Different message types (e.g., *setup*, *teardown*, *data*) are represented by distinct colors. When in use, links take on the color of the message type in transit.

[1]Users can build a network by directly manipulating the objects if the `BuildGNetwork` procedure provided proves too restrictive to the particular application.

Simulator derived trace data encapsulates the dynamic activity present in the network. This reflects the state of the queues, links, and switches, and other components. The visualization tools are implemented in Java (primarily for portability reasons).

5 Photonic Interconnection Networks

There are two primary system designs that the ICNS framework has been used to model to date. Each of these two systems relies on a distinct photonic technology, and as a result, the architectures are appreciably different from one another. The ability for ICNS to be applied to both of these systems attests to its flexibility.

5.1 The *Gemini* Interconnect

The *Gemini* interconnect is an experimental implementation of a novel processor-to-processor interconnection network for tightly-coupled multicomputers [1, 3, 7, 8]. It includes an end-to-end optical data path (including switching of the optical signals) for high-bandwidth, large data volume message delivery. The optical switching is accomplished using $LiNbO_3$ electrooptical 2×2 switches [11, 15]. In addition, *Gemini* includes an electrical path (in parallel with the optical path) that both controls the optical path (i.e., setup of the electrooptical switches) and delivers low-latency, small data volume messages.

The *Gemini* interconnect uses a Banyan topology. Although this is a blocking network, it provides the minimum number of switching stages through the network, and has the additional advantage that each signal goes through the same number of switches. An 8×8 *Gemini* network is illustrated in Figure 10. As can be seen in the figure, each optical switch has an associated electrical counterpart.

Due to the absence of buffering in the optical domain, the optical network is circuit switched. This implies that it will perform well for large data volume messages, for which the latency associated with circuit setup and teardown can be amortized over a large message insertion time. By contrast, the electrical network is packet switched. Here, the design can be optimized for low-latency delivery of small messages (either data messages or control messages) that do not have significant bandwidth requirements.

Electrooptical 2×2 switching elements are the key devices in the fabrication of the *Gemini* $N \times N$ optical data path. [11, 15]. These switching elements rely on the electrooptic effect (i.e., the application of an

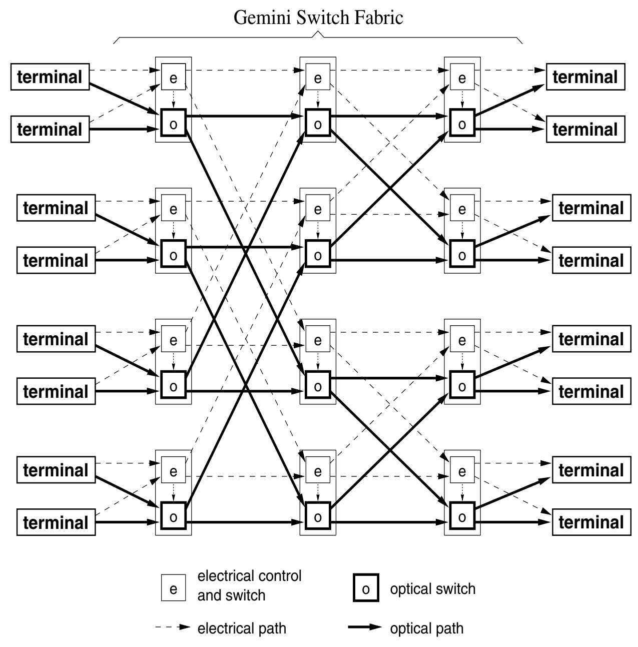
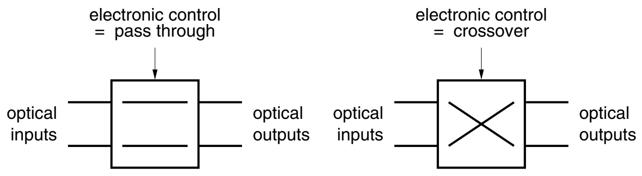


Figure 10: An 8 × 8 *Gemini* network.

electric field to an electrooptical material changes the refractive index of the material). The result is a 2 × 2 optical switching element whose state is determined by an electrical control signal. This is illustrated in Figure 11, which shows a switching element in the pass-through state as well as in the crossover state.

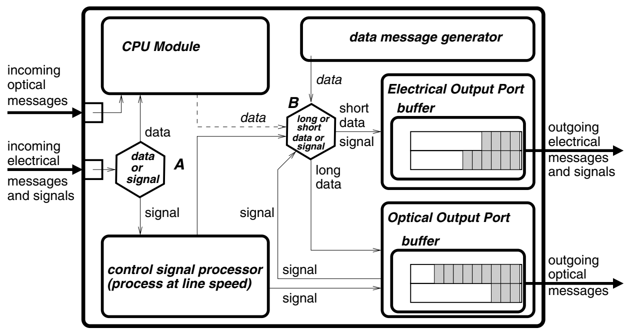


electronic control = pass through
electronic control = crossover
optical inputs
optical outputs
optical inputs
optical outputs

Figure 11: Electrooptical switching elements.

The ICNS model of the *Gemini* interconnect is described next. The terminals connected to the *Gemini* network are modeled as general purpose processors with electrical and optical interfaces. Figure 12 depicts the model of a terminal. Network control signals are assumed to be processed at line rate. Hence there is no input buffer for the terminal. The terminal has separate output buffers for messages intended for different networks. The controller marked 'A' dispatches incoming packets according to packet type. The controller marked 'B' dispatches outgoing traffic according to message type and length.

Figure 13 shows the model of a *Gemini* 2 × 2 electrical switch. The electrical switch has a shared input buffer and separate output buffer at each output. A



CPU Module
data message generator
incoming optical messages
incoming electrical messages and signals
data
data or signal
A
signal
data
B
long or short data or signal
short data
signal
long data
Electrical Output Port
buffer
outgoing electrical messages and signals
Optical Output Port
buffer
outgoing optical messages
control signal processor (process at line speed)
signal
signal
signal

Figure 12: Model for a terminal attached to the network.

routing function module informs the controller where to forward a packet as well as how to control its companion optical switch when a path setup request is being processed.

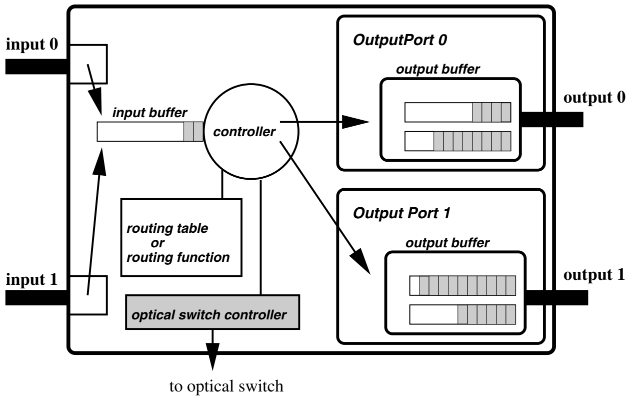


Figure 13: Model for the *Gemini* electrical switch.

In the simulation, the optical and electrical switches are modeled as one object. The links that connect the switches are modeled as one link entity with two channels, one channel carries the electrically switched traffic, the other the optically switched traffic.

The following performance results come from discrete-event simulations using ICNS. Figure 14 shows performance results (mean message delay versus offered load) for four *Gemini* networks using both a basic *setup-teardown* protocol and a virtual output queueing protocol. The *setup-teardown* protocol uses the electrical control network to establish a path in the optical network, sends the data via the optical network, and then tears down the path. The virtual output queueing protocol maintains separate queues for each destination at the source, uses the electrical control network to attempt to establish a path to all destinations that have messages to be delivered, and selects one of the successfully reserved paths to deliver data.

The parameters chosen for the simulated networks are such that the ratio of average message length to control signal length is 16K and the ratio of optical link bandwidth to electrical link bandwidth is 12.

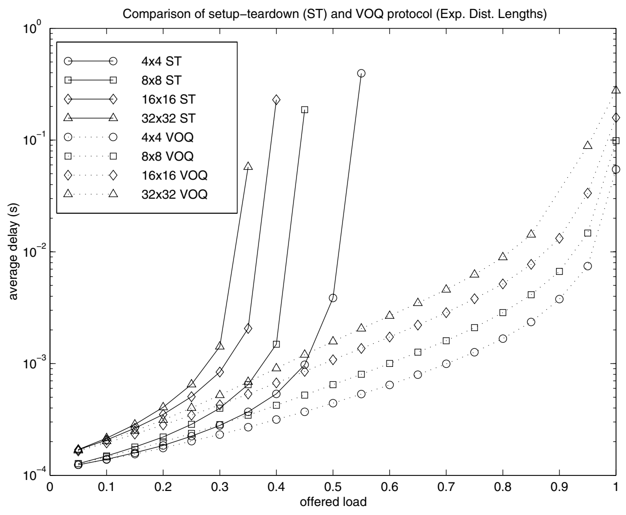


Figure 14: Average optical message delay using the basic *setup-teardown* protocol and the virtual output queueing protocol [3].

The four networks were simulated using the same set of parameters. Messages are generated at each terminal according to an independent and identically distributed Poisson process. Their destinations are uniformly distributed to all outputs and message lengths are exponentially distributed. In the figure, the load axis is normalized to the theoretical maximum throughput.

While throughput is clearly limited using the basic setup-teardown protocol, we see that the optical network can provide close to 100% throughput using the VOQ protocol[2]. At the same time the figure also shows that without VOQ saturation occurs at an offered load of between 30% and 50% capacity.

Table 1 shows the load experienced by the control signals on the electrical network. We see that for the parameters chosen, sending multiple setup requests (as required by the VOQ protocol) does not lead to significant congestion in the electrical network. This supports the use of the electrical network for sending short, latency-sensitive messages (both control and data) without incurring significant queueing delays.

[2]McKeown et al. have proven in [14] that 100% throughput is achievable in a non-blocking, input-queued switch using a non-FIFO queueing scheme such as VOQ assuming random, homogeneous traffic. It remains to be seen whether such performance is achievable in a blocking network such as the Banyan network used in *Gemini.*

Table 1: Electrical network load using the VOQ protocol [3].

Network size	Electrical network load
4×4	$< 0.6\%$
8×8	$< 1.2\%$
16×16	$< 2.4\%$
32×32	$< 4.6\%$

The above set of example performance simulations illustrate the utility of ICNS for evaluating alternative queuing protocols. In [7], the ICNS framework is used to demonstrate the importance of a fairness protocol in the circuit-switched optical data path as well as present performance results with the fairness protocol in place.

5.2 A Photonic Multiring

The photonic multiring is a system that exploits new developments in Vertical Cavity Surface Emitting Laser (VCSEL) technology [2, 4, 6, 12]. The enabling technology for this system is the availability of 2-dimensional arrays of VCSELs and detectors bonded to silicon circuitry [10]. The union of silicon processing with GaAs-based optoelectronics provides a powerful combination, significantly increasing the communications bandwidth available off-chip.

Prototype interconnects have been constructed with 16×16 arrays of VCSELs and photodetectors on a single chip [16]. In this system, the VCSELs arrays and photodiode arrays were flip-chip bonded to a CMOS chip using heterogeneous integration techniques. Although the demonstration of [16] used bulk optics to deliver light between ICs, optical paths have been designed using both rigid optical links [5] (useful for chip-to-chip links on a board), and flexible fiber imaging guides [9] (useful for board-to-board links).

The availability of a large number of VCSEL-detector pairs in the optical interconnect suggests the partitioning of the optical links into sets with each set being associated with an individual channel (i.e., space-division multiplexing). Figure 15 illustrates the allocation of VCSELs and detectors for a four channel system utilizing 16×16 arrays of optical elements. As shown in the top of the figure, one quarter of the elements are used for each channel. Each square in the top view of Figure 15 contains a single VCSEL or detector. If the individual element communicates at

1 Gb/s, this yields $16^2/4 = 64$ Gb/s per channel. The side view of the figure illustrates (conceptually) how two adjacent chips might communicate.

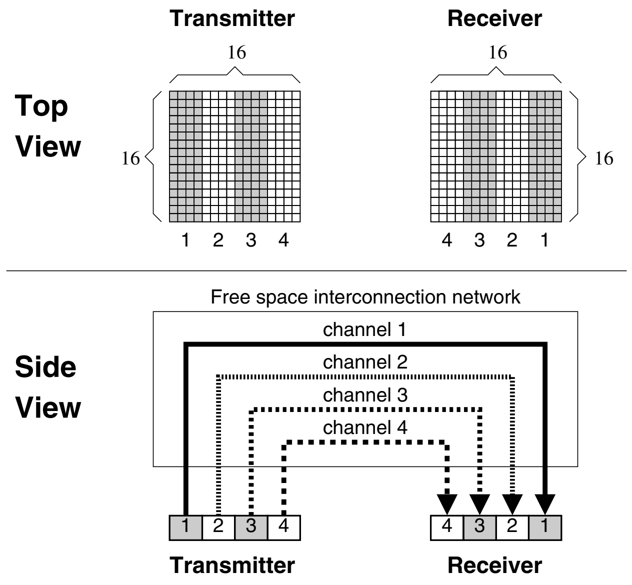


Figure 15: Allocation of VCSEL-detector pairs to a four channel system. 16×16 VCSEL-detector arrays are used, with a 4×16 array allocated to each channel.

The photonic technologies used in this design are most cost-effective when used with a fan-in and fan-out of one and a topology meeting this fan-in/fan-out goal is a ring. While there are many approaches to developing a ring based interconnect, given the very high bandwidths available, the multiring [13] design of Figure 16 has been chosen.

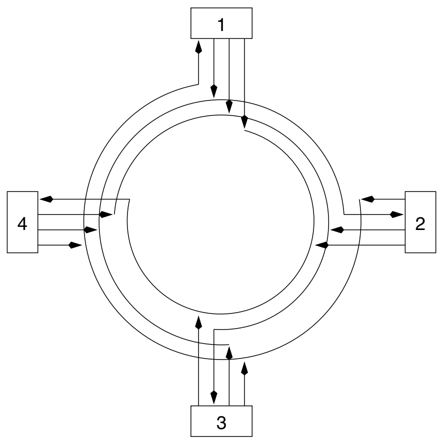


Figure 16: Multiring.

In the 4-node example of Figure 16, each of the four rings is associated with a given destination node. The outside ring, for example, is associated with node 1 and the next-to-outside ring is associated with node 2. The inside ring is associated with node 4. With the multiring topology, each ring can be thought of as a daisy chain terminating at the destination node. Thus, communication between node i and node j requires that node i send its message on the ring which has node j as the destination.

The performance evaluation for this system explores the ability to reconfigure the bandwidth associated with each ring in the multiring by changing the number of VCSEL-detector pairs associated with each channel (and hence the input bandwidth available to each destination).

The applications of interest are ones in which computation and communication alternate with one another (i.e., proceed in phases). The performance results presented here are derived from 2 real applications (synthetic aperture radar (SAR) image formation and a beamforming application) and 5 synthetic applications. The synthetic applications have from 3 to 6 communications phases. Their communications patterns are randomly chosen from the following set: broadcast, reduce, all-to-all, and point-to-point. The flows and message sizes are also randomly generated.

An ICNS simulation model was used to evaluate the performance of the reconfigurable photonic multiring. An 8 node system was simulated, with communication traffic generated according to the 2 real applications and 5 synthetic applications described above. For each application, the system was simulated twice. The initial simulation utilized a uniform bandwidth allocation to each potential flow (all source-destination pairs are allocated an equal fraction of the total bandwidth). In the second simulation the interconnection network was reconfigured at the start of each communication phase to provide an optical bandwidth allocation best suited to the communication pattern required by the algorithm in that phase.

Figure 17 shows the mean, minimum, and maximum speedup obtained for each type of communication pattern, independent of the application in which it is found. The speedup is defined as the ratio of the communication completion time with a uniform bandwidth allocation to the completion time with a reconfigured allocation. The performance improvement is significant across the board, indicating a clear benefit to reconfigurability in the interconnection network.

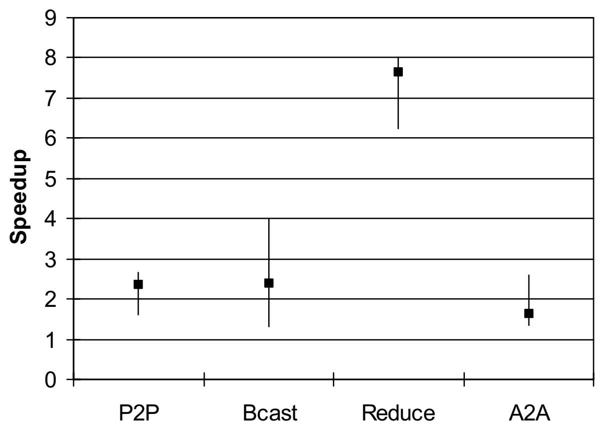


Figure 17: Communications speedup with a reconfigurable multiring. [2]

6 Conclusions and Future Work

In this document we have presented the design of the interconnection network simulator, ICNS. We have described the classes that form the core of ICNS. We have also shown how various components in a multicomputer interconnection network can be modeled using the classes provided by ICNS. The classes can be subclassed and extended to model components with richer functions.

ICNS has been used to study the *Gemini* network and a photonic multiring network. We are currently extending the multiring network simulation to include dynamic reconfigurability and exploring its use as a switching fabric for an internet router.

References

[1] 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.

[2] R. Chamberlain, M. Franklin, and P. Krishnamurthy. Performance evaluation of a reconfigurable, embedded photonic multiring interconnection network. In *Proc. of 5th High Performance Embedded Computing Workshop*, November 2001.

[3] R.D. Chamberlain, M.A. Franklin, and Ch'ng Shi Baw. *Gemini*: An optical interconnection network for parallel processing. *IEEE Trans. on Parallel and Distributed Systems*, (in press).

[4] R.D. Chamberlain, M.A. Franklin, and A. Mahajan. VLSI photonic ring interconnect for embedded multicomputers: Architecture and performance. In *Proc. of 14th Conf. on Parallel and Distributed Computing Systems*, August 2001.

[5] M. Chateauneuf et al. Design, implementation and characterization of a 2-D bi-directional free-space optical link. In *Proc. of Optics in Computing*, pages 530–538, June 2000.

[6] Ch'ng Shi Baw, R.D. Chamberlain, and M.A. Franklin. Design of an interconnection network using VLSI photonics and free-space optical technologies. In *Proc. of 6th Int'l Conf. on Parallel Interconnects*, pages 52–61, October 1999.

[7] Ch'ng Shi Baw, R.D. Chamberlain, and M.A. Franklin. Fair scheduling in an optical interconnection network. In *Proc. of 7th Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 56–65, October 1999.

[8] Ch'ng Shi Baw, R.D. Chamberlain, M.A. Franklin, and M.G. Wrighton. The *Gemini* interconnect: Data path measurements and performance analysis. In *Proc. of 6th Int'l Conf. on Parallel Interconnects*, pages 21–30, October 1999.

[9] H. Kosaka et al. A two-dimensional optical parallel transmission using a vertical-cavity surface emitting laser array module and an image fiber. *IEEE Photon. Tech. Lett.*, 9:253–255, 1997.

[10] Y. Li, E. Towe, and M. Haney, eds. *Proc. on Short Distance Optical Interconnections in Digital Systems.* IEEE, 2000.

[11] Lucent Technologies. Guided wave optical switch products. Preliminary data sheet, 1997.

[12] A. Mahajan, M.A. Franklin, and R.D. Chamberlain. Fairness issues in an embedded photonic ring interconnect. In *Proc. of 4th High Performance Embedded Computing Workshop*, September 2000.

[13] M. Marsan et al. All-optical WDM multi-rings with differentiated QoS. *IEEE Communications Magazine*, pages 58–66, February 1999.

[14] N.W. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proc. of Infocom*, March 1996.

[15] E.J. Murphy, T.O. Murphy, R.W. Irvin, R. Grencavich, G.W. Davis, and G.W. Richards. Enhanced performance switch arrays for optical switching networks. In *Proc. of ECIO*, April 1997.

[16] D. Plant et al. A 256 channel bi-directional optical interconnect using VCSELs and photodiodes on CMOS. In *Proc. of Optics in Computing*, pages 1046–1054, June 2000.