SFL: A PARALLEL SIMULATION PROGRAM FOR BANYAN NETWORKS

Dong W. Kim, G. Jack Lipovski, and R. Jenevein

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-88-43 December 1988

Abstract

We evaluate the performance of double- and single-ended regular CC- and SW-banyan networks by simulation. First, we give concrete system level switch models for double- and single-ended banyan networks. Next, we introduce a powerful simulation program called SFL. SFL (Spread, Fanout, and Level) is a parallel time-driven simulation program for packet-switching synchronous banyan networks. SFL provides users with a three dimensional design space. The first dimension provides a choice of either a CC- or an SW-banyan network. The second dimension provides a choice of tuples (spread, fanout, level). The last dimension provides a choice of three communication protocols: DES (Double-Ended Simplex), SEFD (Single-Ended Full Duplex), or SEHD (Single-Ended Half Duplex). Finally, our simulation results show that single-ended CC-banyan networks can be excellent interconnection networks for multiprocessor systems.

SFL: A Parallel Simulation Program for Banyan Networks

Dong W. Kim, G. Jack Lipovski, & R. Jenevein

The University of Texas at Austin
Austin, TX 78712

1 Introduction

Two approaches to evaluating the performance of packet-switching interconnection networks are analytic performance modeling and simulation performance modeling [21]. For analytic performance modeling, there are three widely accepted assumptions to make the analysis tractable: 1) output buffers for incoming packets are infinite, 2) networks are under a uniform traffic pattern, and 3) k packets on input ports of a k by k switch can enter into the same output buffer each network cycle. Many analytic performance models with similar assumptions have been reported during the past decade [3,5,7,11,16,26,39]. However, in this work, we developed simulation performance models which relax these assumptions and significantly extend the logical design space.

SFL (Spread, Fanout, and Level) is a parallel time-driven (TD) simulation program for packet-switching synchronous banyan networks. It is written in PPL[1] (Parallel Programming Language based on C) [47] and runs on the Sequent Balance 8000. The input parameters to SFL are as follows:

1. Choice of target network: CC- or SW-banyan.
2. Spread, Fanout, and Level.

[1] ©Microelectronics and Computer Technology Corporation, 1985

3. Choice of network communication protocol: DES (Double-Ended Simplex), SEFD (Single-Ended Full Duplex), or SEHD (Single-Ended Half Duplex).
4. Buffer size: the amount of storage allocated in each switch for incoming packets (the unit is the number of packets).
5. Birth rate: the probability that a packet is generated per source (or processor) per network cycle.
6. Reference matrix whose $(i, j)^{\text{th}}$ element is the probability that the i^{th} source will send a packet to the j^{th} sink, and is a function of time.

The three network communication protocols will be further explained in later sections.

The output parameters from SFL are as follows:

1. Maximum and average response time for packets to travel from their sources to their sinks.
2. Maximum and average buffer length for each level.

The primary goal of SFL is to ease the decision-making process of designing interconnection networks by providing performance estimates of the following design alternatives:

1. CC-banyan networks vs. corresponding SW-banyan networks
2. rectangular networks vs. nonrectangular networks
3. double-ended networks vs. single-ended networks
4. full duplex networks vs. half duplex networks

Since the reference matrix is arbitrary, we can also investigate the performance of the networks under uniform, hot spot, local, and any other realistic traffic patterns.

Section 2 introduces switch models for DES, SEFD, and SEHD protocols. Section 3 presents our simulation method in depth. Section 4 gives simulation results and draws conclusions on which networks perform better under what conditions. Finally, Section 5 gives conclusions.

2 Switch Models

We begin with a switch model for DES networks, then extend it for SEFD and SEHD networks. The switch model for DES networks is based on the TRACs packet switch [44]. Many researchers have proposed similar switches for their analysis and simulation [7,11,16,26,32,39,41,49]. Recently, switches with a message combining unit have been proposed for the NYU Ultracomputer project [19] and the IBM RP3 project [42].

2.1 DES Networks

In DES networks, there are input ports for processors on one side, and output ports for memories on the opposite side. Packets are unidirectionally delivered from the input ports to the output ports. The processors have their own local momories or caches, and interfaces to the network. Similarly, the memories have interfaces to the network, the global memories, and possibly mass storage. The processors send packets to the memories to read or update global variables through one DES network. The memories send packets to the processors to return contents of the requested memory locations for read cycles or acknowledge signals for write cycles through another DES network. SFL explicitly represents and simulates only the input and output interface to the network and switches, because our only concern is the performance of the network.

2.1.1 Input and Output Interface

One active hardware component in a processor board is a communication channel which has an input interface to the network. Figure 1 (a) shows the interface which consists of an output queue and a control unit. The reason why the output queue exists is that 1) bursty arrival of packets may temporarily overload the channel, 2) a few processors may share the channel, and 3) there may be a mismatch betweeen a processor cycle and a network cycle. The control unit exchanges handshaking signals with the corresponding switch to ask for the next available space. The two handshaking signals are a request signal and a grant signal. The request signal is sent from the interface to the switch to tell that there is a packet

to be sent to the switch. The grant signal is sent from the switch to the interface to tell whether the packet can be relayed or not. The interface is synchronized with the network cycle. We assume that a new packet can join the queue at the end of each cycle.

In a memory unit, there is an output interface from the nework. Figure 1 (b) shows the interface which consists of an input queue and a control unit. The queue exists here for similar reasons. The control unit accepts a request signal from the corresponding switch and sends a grant signal to it according to the state of its queue. The interface is synchronized with the network cycle.

2.1.2 Switch Model for DES Networks

A k by k switch for DES networks consists of a control unit, a crossbar switch, and k output queues. Figure 2 shows a 2 by 2 DES switch.

The control unit exchanges an input request signal (REQI) and an output grant signal (GRANTO) with each of its predecessors (P's). It also exchanges an output request signal (REQO) and receives an input grant signal (GRANTI) with each of its successors (S's). The control unit is a state machine which handles the signals in the following sequence during a network cycle:

1. Extracts destination tag fields from packets on the front of its queues.
2. To each S, sends REQO with the destination tag field.
3. Receives REQI from each P and GRANTI from each S.
4. Determines which packets can be sent or received according to the REQIs, the GRANTIs, vacancies of its queues, and a given conflict resolution scheme [2].
5. Sends proper GRANTO to each P.
6. Sets the crossbar properly.

[2] Three conflict resolution schemes are random, fixed priority, and round robin.

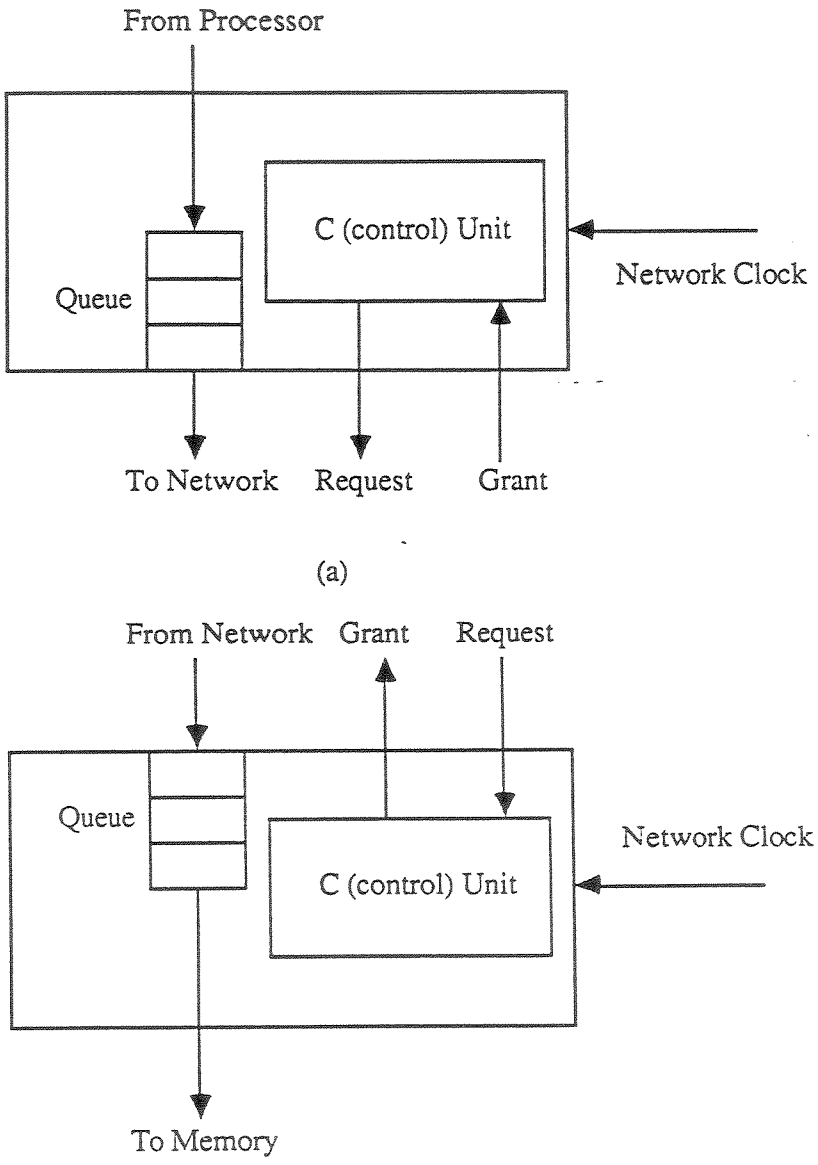


Figure 1: (a) Input, and (b) output interface to the network

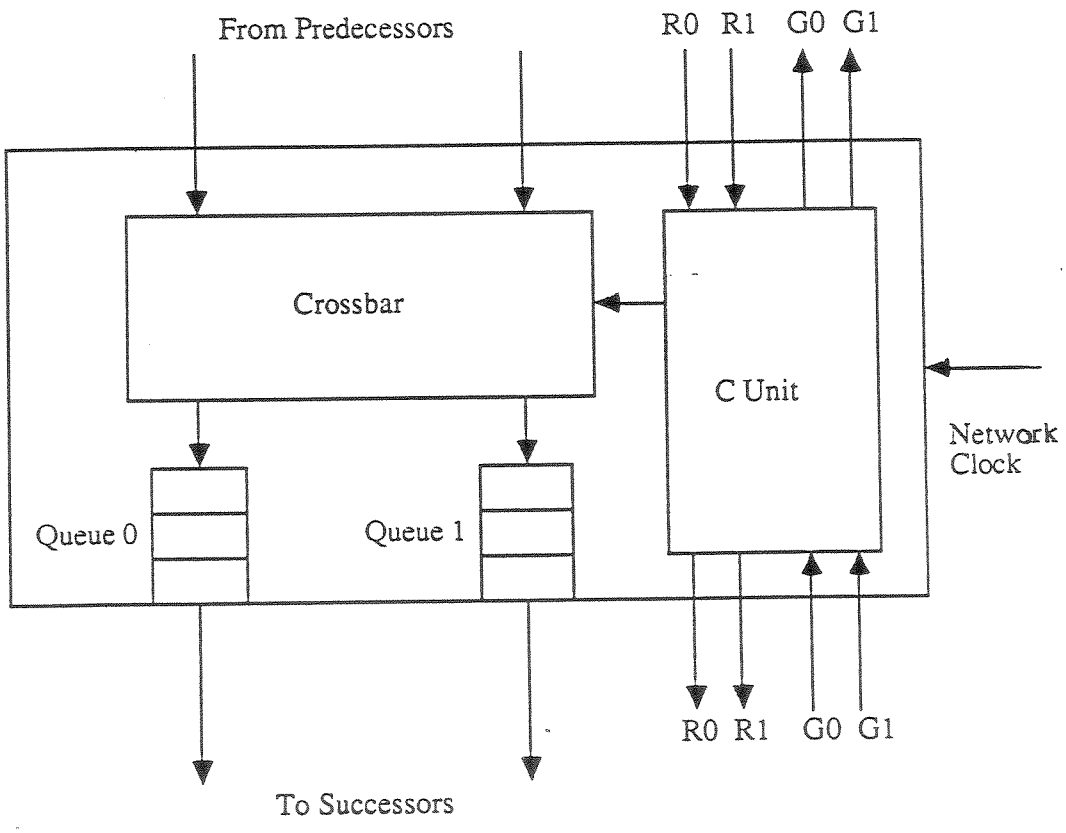


Figure 2: A 2 by 2 DES switch

A packet consists of fixed length bytes or words and is called a packet train [34]. The first byte or word of the packet train contains control information and is called the header. The rest of the packet train is data. Once input and output signals of the control units are settled, each packet train, granted by the control unit, can be sent to the next stage during a network cycle. In this way, the current packet's data can be sent while the next packet's headers are being processed, and this increases the degree of pipelining. However, this prevents more than one packet train from entering the same queue each network cycle. A queue is supposed to be able to accept an incoming packet [3] and send an outgoing packet each network cycle. Note that the grant signals are propagated from the memory units through the network to the processor units like a ripple carry. Therefore, this propagation delay is proportional to the number of stages in the network and affects the length of the network cycle (it is also affected by the length of the packet train).

States of the crossbar switch are set by the control unit. Since only one packet can enter into a queue each network cycle, there are only seven possible states in a 2 by 2 switch as shown in Figure 3. Figure 4 shows a (2,2,2) DES CC-banyan network.

2.2 Switch Model for SEFD Networks

In single-ended networks, the input and output ports of the network are on the same side (say on the base level). Each processor has its own local memory which can be globally accessed by other processors and an input and an output interface to the network. Packets are generated by the processors, sent up to the network, reflected in the minimum level, and eventually absorbed by other processors. Therefore, switches in the single-ended network are designed to transmit up, reflect, and transmit down packets.

Figure 5 shows a 2 by 2 SEFD switch. Each port of the SEFD switch consists of two logical simplex lines; one for upgoing packets and the other for downgoing packets. There is a queue on each output port. A queue can accept an incoming packet and send an outgoing packet each network

[3] We use a term "packet" to represent a packet train.

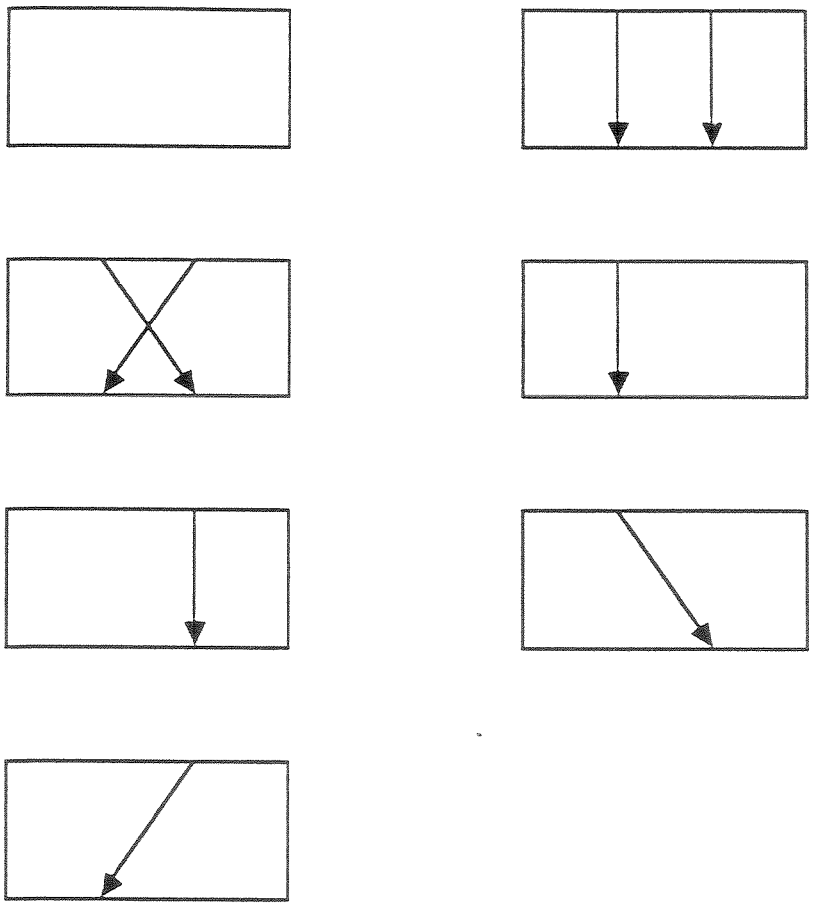


Figure 3: States of DES crossbar switch

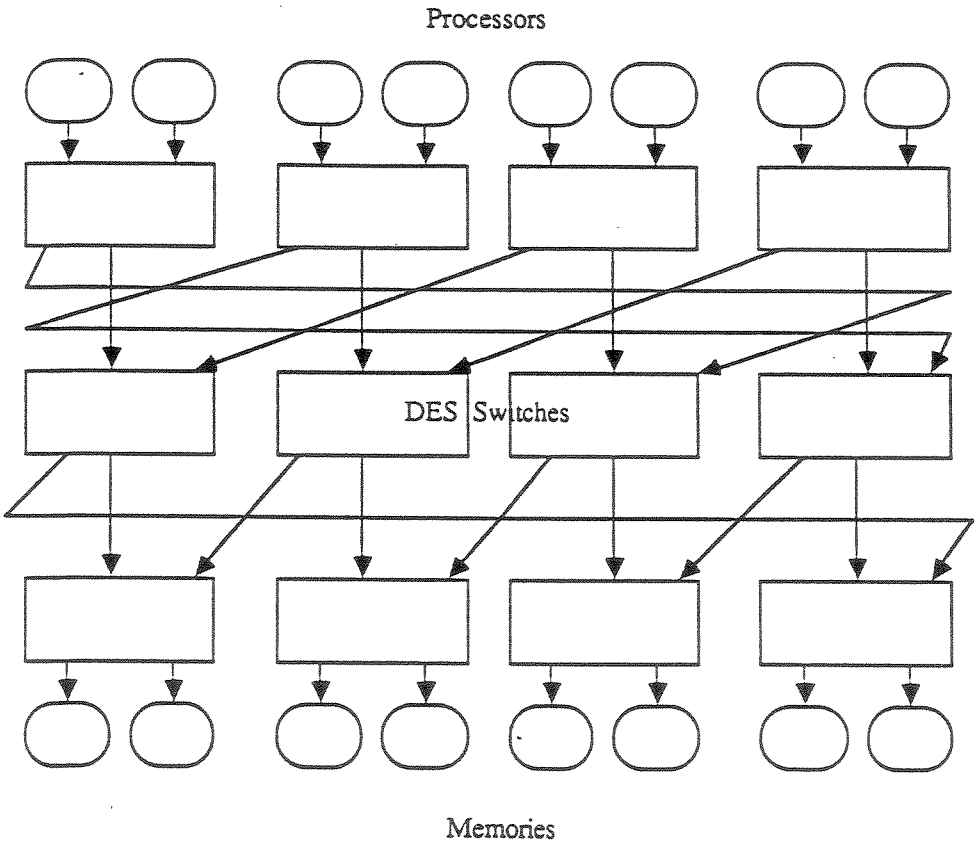


Figure 4: A (2,2,2) DES CC-banyan network

cycle. Queues on the top of a switch are called T-queues and those on the bottom B-queues. A 4 by 4 crossbar switch is in the middle. Input ports are connected to rows of the crossbar switch and output ports to columns. The check pattern in the $(i,j)^{\text{th}}$ element of the crossbar switch indicates that the i^{th} row is disconnected from the j^{th} column. Figure 6 shows states of the crossbar switch.

Figure 5 also shows a control unit. The control unit exchanges an input request signal (REQI), an output grant signal (GRANTO), an output request signal (REQO), and an input grant signal (GRANTI) with each corresponding switch. Given a switch S in level k, let SL be a switch or a processor in level $k-1$ which corresponds with switch S, and let SH be a switch in level $k+1$ which corresponds with switch S. The control unit in switch S handles the signals in the following sequence during a network cycle:

1. Extracts destination tag fields from packets on the front of its queues.
2. To each SL and SH, sends REQO with the destination tag field.
3. Receives REQI from each SH for SH's downgoing packets, REQI from each SL for SL's reflecting packets, and GRANTI from each SL for S's downgoing packets.
4. Determines which downgoing or reflecting packets can be sent or received according to the REQIs, the GRANTIs, vacancies of S's B-queues, and a given conflict resolution scheme.
5. Sends proper GRANTO to each SH and SL.
6. Receives REQI from each SL for SL's upgoing packets and GRANTI from each SH for S's upgoing packets.
7. Determines which upgoing packets can be sent or received according to the REQIs, the GRANTIs, vacancies of S's T-queues, and a given conflict resolution scheme.
8. Sends GRANTO to each SL.
9. Sets the crossbar properly.

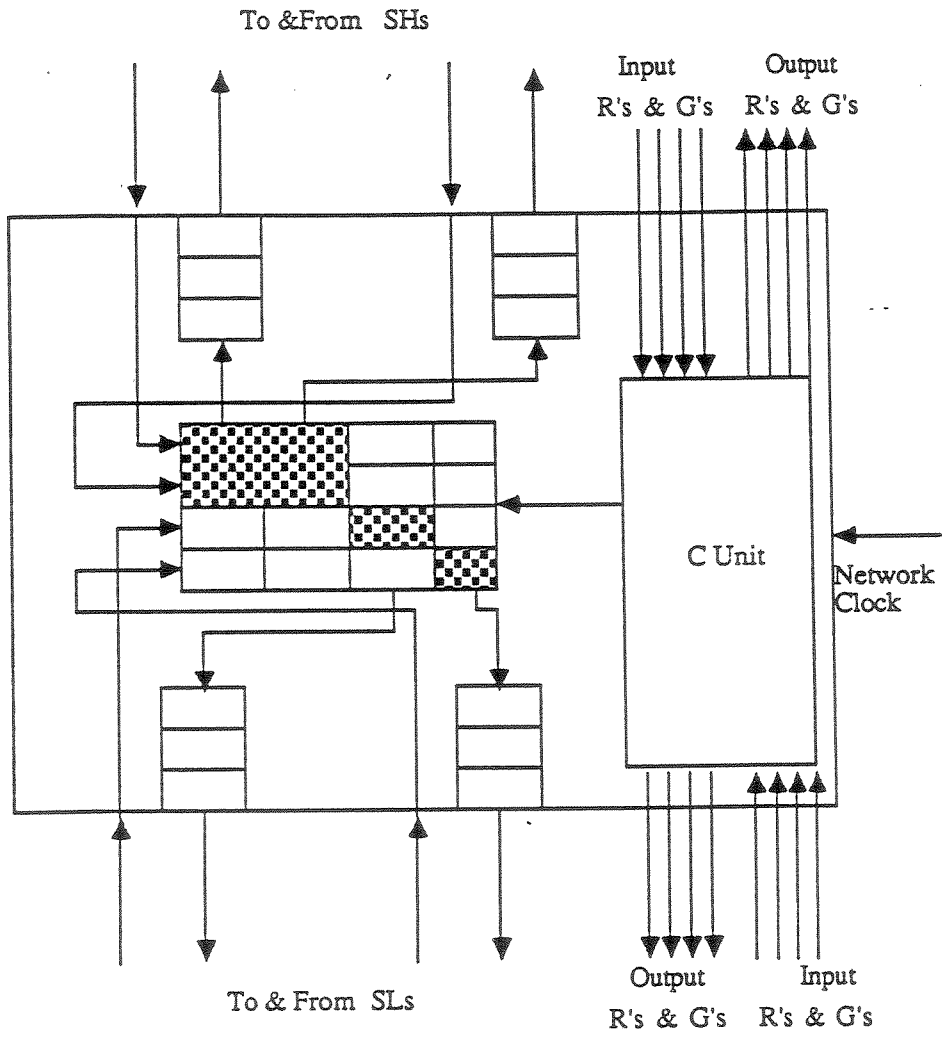
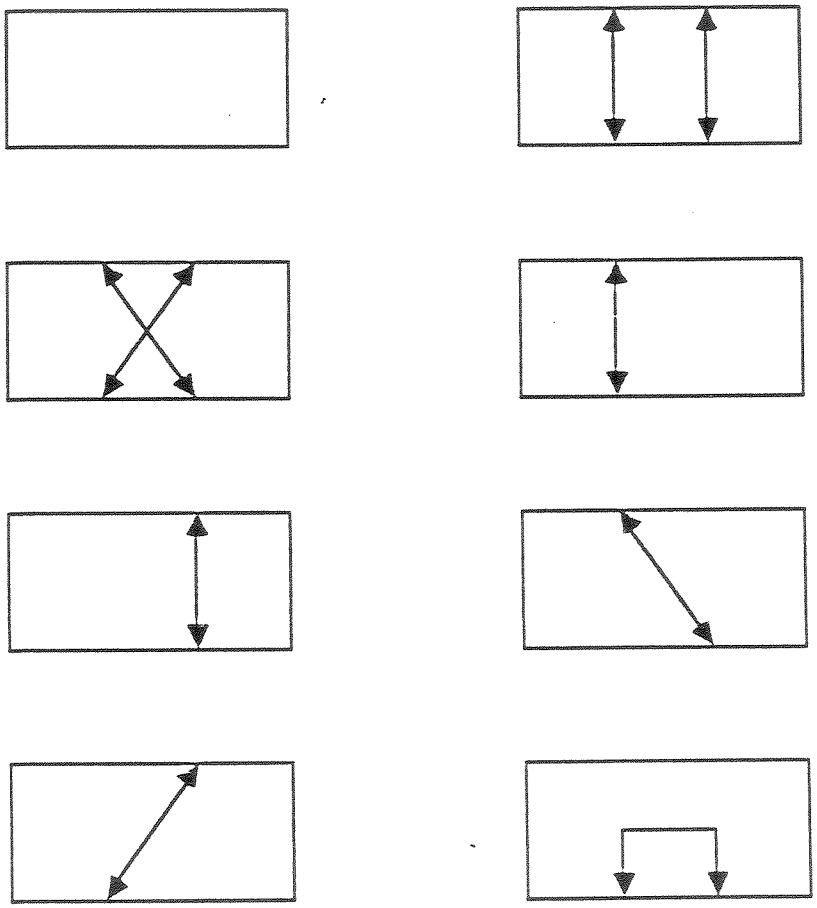


Figure 5: A 2 by 2 SEFD switch



Note: ↔ means either → or ←, but not both.

Figure 6: States of SEFD crossbar switch

Grant signals are initially generated from the output interfaces of the processor units. Then, switches in the base level determine the future vacancies of their B-queues and can generate proper grant signals for switches in level 1 or processors which have packets heading for them. Similarly, once the grant signals arrive at switches in level 1, the switches determine the future vacancies of their B-queues and send proper grant signals to switches in level 2 or 1 which have packets heading for them. In this way, grant signals for B-queues are propagated up to switches in the apex level. Then, switches in the apex level determine the future vacancies of their B-queues and can generate grant signals for switches one level below whose T-queues have packets heading for them. Then, grant signals for T-queues are similarly propagated down to the input interfaces of the processor units. Therefore, overall propagation delay is proportional to two times the number of stages in the network. Figure 7 shows a (2,2,2) SEFD CC-banyan network.

2.3 Switch Model for SEHD Networks

There are two logical simplex lines for each port of a SEFD switch. Supposing that each simplex line provides 8 bits of data path, at least 64 pins for data paths are required in a 2 by 2 SEFD switch. To reduce the number of pins (the most valuable resource), the two logical simplex lines can be replaced by a single logical half duplex line.

Figure 8 shows a 2 by 2 SEHD switch. Each port of the SEHD switch has a single logical half duplex line; each line can transmit an upgoing or a downgoing packet at a time, but not both at the same time. There are two T-queues and no B-queues in the switch. Each queue can store both upgoing and downgoing packets. During each network cycle, it can either accept an upgoing packet and send an upgoing (or reflecting) packet, or it can accept a downgoing packet and send a downgoing packet. However, it cannot accept an upgoing and a downgoing packet or send an upgoing and a downgoing packet in a single network cycle. The same 4 by 4 crossbar switch as that shown for a 2 by 2 SEFD switch is in the middle. Reflecting packets are not buffered by the switch which reflects them.

A queue for a SEHD switch can store both upgoing and downgoing packets. Let Q_0 and Q_1 be two corresponding queues in the network as

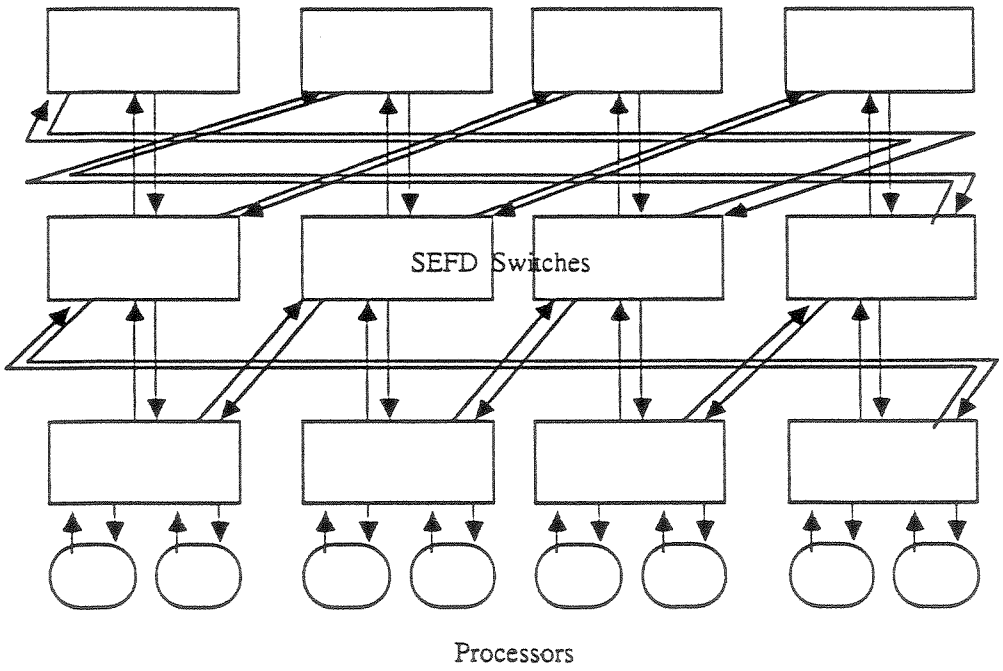


Figure 7: A (2,2,2) SEFD CC-banyan network

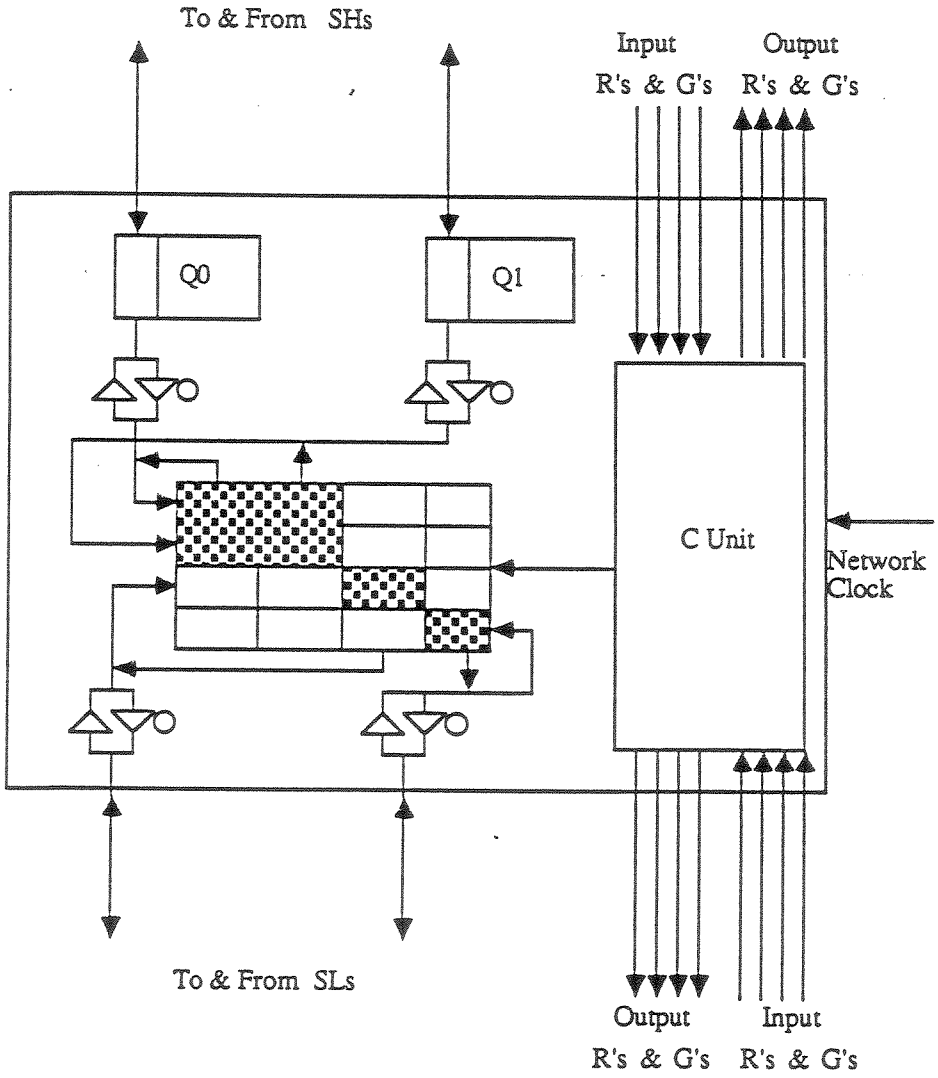


Figure 8: A 2 by 2 SEHD switch

shown in Figure 9 (a). Since a single logical half duplex line is shared by the queues, a deadlock state occurs under the following conditions:

1. Both queues are full of upgoing and downgoing packets.
2. A packet on the front of Q_0 heads for Q_1 and a packet on the front of Q_1 heads for Q_0.

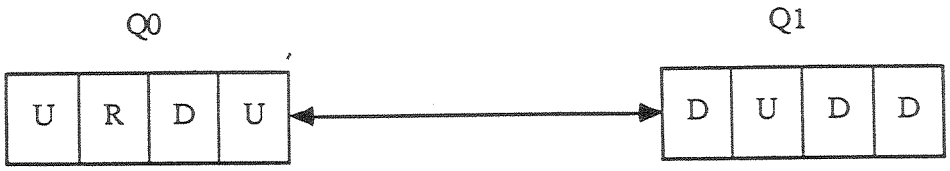
This condition can be modeled by a directed graph in which each node represents a subset of a queue and each directed arc represents permitted packet flow between nodes [12,17,36,37]. See Figure 9 (b). The deadlock state can be avoided by designing a special queue unit which prevents the packet flow from forming a cycle in the directed graph.

As shown in Figure 10, the queue unit has a storage pool. The pool is divided into three lists; one for upgoing packets (ULIST), another for downgoing packets (DLIST), and the other for free space (FREE). FREE has space for at least two packets. A packet can enter into a proper list if:

1. FREE has space for more than one packet.
2. FREE has space for only one packet and ULIST and DLIST are not empty.
3. FREE has space for only one packet, ULIST is empty, and the packet is upgoing.
4. FREE has space for only one packet, DLIST is empty, and the packet is downgoing.

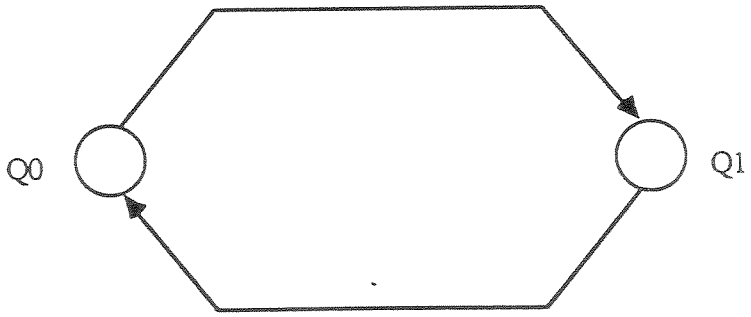
The queue unit is now represented by two nodes (UNODE and DNODE) in a directed graph; UNODE represents ULIST and DNODE represents DLIST. In the directed graph, there are arcs from UNODEs to UNODEs (representing upgoing packet flow), from UNODEs to DNODEs (representing reflecting packet flow), and from DNODEs to DNODEs (representing downgoing packet flow). Since there is no arc from a DNODE to an UNODE, the directed graph is acyclic. Therefore, the queue unit prevents deadlock at the cost of space for at most one packet. See Figure 11.

The control unit as shown in Figure 8 handles the signals in the following sequence during a network cycle:



Note: U is a upgoing packet, D is a downgoing packet, and R is a reflecting packet

(a)



(b)

Figure 9: (a) A deadlock state, and (b) its directed graph of two corresponding SEHD queues

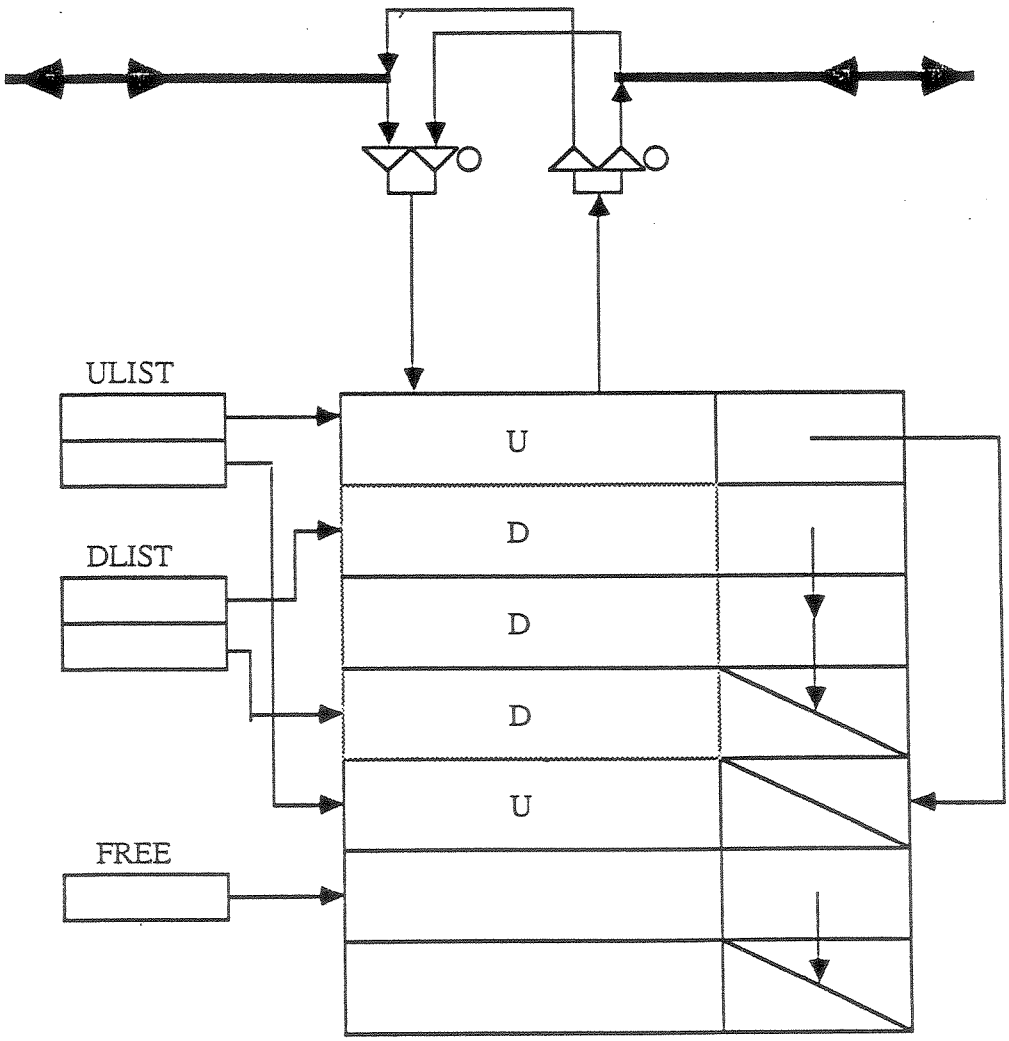


Figure 10: A special SEHD queue unit for preventing deadlock

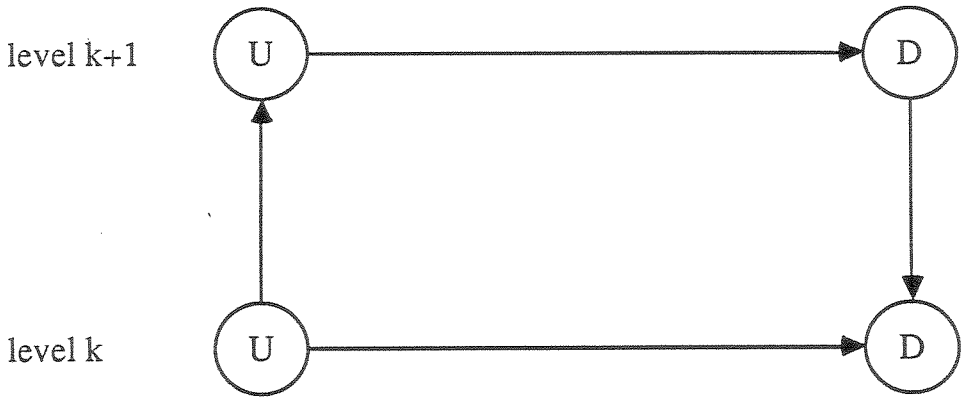


Figure 11: An acyclic directed graph which guarantees deadlock free routing

1. Extracts a destination tag field from a packet on the front of each list in its queues.
2. To each SL and SH, sends REQO with the destination tag field.
3. Receives REQI from each SL for SL's reflecting packets and GRAN-TI from each SL for S's downgoing packets or SL's reflecting packets.
4. Determines which downgoing or reflecting packets can be sent or received according to the REQIs, the GRANTIs, availability of S's half duplex lines, vacancies of SL's queues, and a given conflict resolution scheme.
5. Sends GRANTO to each SL for its reflecting packets.
6. Receives REQI from each SH for SH's downgoing packets or S's siblings reflecting packets, determines which downgoing or reflecting packets can be received, and sends GRANTO to each SH.
7. Receives REQI from each SL for SL's upgoing packets and GRANTI from each SH for S's reflecting or upgoing packets.
8. Determines which upgoing or reflecting packets can be received and sends GRANTO to each SL.
9. Sets the crossbar properly.

Like the SEFD network, grant signals are generated from processor units for DLISTs, and propagated up to switches in the apex level. Then, grant signals for ULISTs are propagated from the switches in the apex level to the processor units. Overall propagation delay is proportional to two times the number of stages in the network. Figure 12 shows a (2,2,2) SEHD CC-banyan network.

3 Simulation Method

The variable in a simulation program which gives the current value of simulated time is called the simulation clock. Two principal approaches

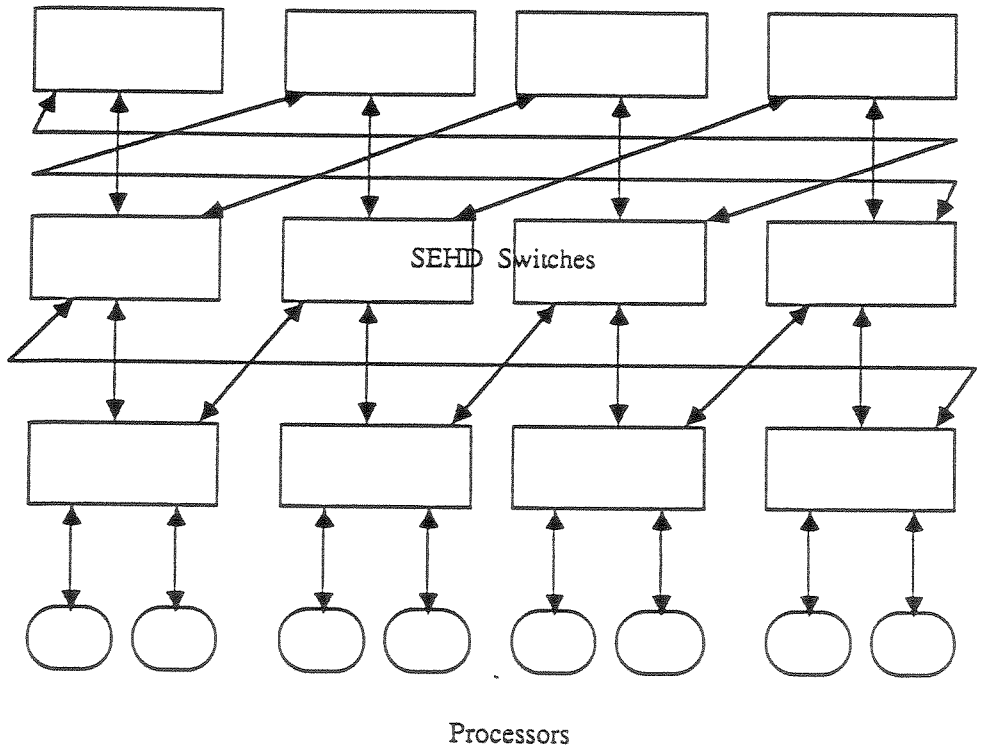


Figure 12: A (2,2,2) SEHD CC-banyan network

for advancing the simulation clock are *next-event time advance* and *fixed-increment time advance* [29]. With the next-event time advance approach, the simulation clock is advanced to the time of the most imminent event. Simulation with this approach is called event-driven (ED) simulation. With the fixed-increment time advance approach, the simulation clock is advanced in increments of exactly δt time units for some appropriate choice of δt. δt is usually chosen to be the smallest time interval between two events in which the earlier event affects the later event. However, this approach may become inefficient if events occur irregularly, and therfore, many useless computational cycles are involved. Simulation with this approach is called time-driven (TD) simulation.

SFL is a parallel time-driven (TD) simulation program. We choose a TD simulation for the following reasons:

1. One may write a parallel ED simulation program with a process-oriented approach [29]. In the program, one usually defines a process which describes the entire experience of a packet as it flows through a target network. A process is created whenever a packet arrives at the network. This approach is taken in many simulation programs for queueing networks. The first problem of this approach is that there is no efficient way to detect and resolve conflicts among packets because some process needs to know the destination tags which are local variables to processes representing packets. This problem can be alleviated on the assumption that k packets on input ports of a k by k switch can enter into the same output buffer in a single network cycle. The second problem is that the average number of concurrent processes can easily exceed tens of thousands for a moderately large network. Therefore, more simulation time will be spent for creating and terminating processes and swapping pages. Besides, it may exceed the process limit provided by the available multiprocessor system. The last problem is that not only is a parallel algorithm for handling the shared event list efficiently hard to devise, but also it is possible that the algorithm can perform poorly for synchronous network simulation because most events are scheduled in the time slot in the event list which stores events

occuring at the next network cycle. See [35] for discussions about the difficulties of parallel ED simulation.

2. For our TD simulation program for synchronous banyan networks, we define processes which describe how active components such as processors and switches operate during a network cycle. Packets are represented by a data structure instead of a process. They are updated by the switch processes as they flow through the network. By doing this, we can alleviate the aforementioned problems. For the first problem, to each process, we assign a pointer to each queue having incoming packets. Then, the switch process can easily access the destination tag of each incoming packet and resolve any conflicts. For the second problem, since the processes simulate switches or processors, the number of processes cannot exceed the sum of the number of switches and the number of processors. Moreover, the processes are static. Once they are created, they do not terminate until the simulation is over. However, even this number may be excessive for a large network, and also the simulation may become inefficient due to the small granularity of the process and its context switch overhead. Instead, since the network is synchronous, a process can simulate a set of switches in the same level or a set of processors because components in the same set do the same operations [4] during a network cycle. Therefore, the number of processes and hence the process granularity can be varied by changing the number of components in the set. In TD simulation, during every network cycle, every possible event is checked and system states are updated according to the events that actually occur. Therefore, no event list is maintained by the simulation. This solves the last problem.

3. Many concurrent events that occur in a heavily loaded network make the parallel TD simulation efficient. If a target network is lightly loaded, TD simulation is less efficient than ED simulation because many blank cycles may be involved. In this case, how-

[4] This is the property of the synchronous network that makes a parallel execution of simulation possible.

ever, we can save total simulation time by reducing the batch size [29] because the network will reach steady state faster than with a heavily loaded network.

If we limit the number of outstanding requests from each processor, there will be a fixed number of packets circulated in the network. We refer to this network model as a *closed* model [30]. Meanwhile, we refer to an *open* model as a network model with an infinite stream of arriving packets from each processor. While real networks are more likely to be modeled as closed networks, the open networks are better for comparing the maximum throughput (or capacity) of the proposed networks. Therefore, we model the networks as open networks. We will present simulation methods for all three protocols in the following subsections. We introduce internal representations of the components with PASCAL-like pseudo code, and component processes.

3.1 Simulation Method for DES networks

PPL [47] allows users to define concurrent processes. These processes communicate via shared access to common memory and via PPL communication facilities like *events*, *locks*, *mailboxes*, and *synchronization points*. We assume that the facilities are also available in our pseudo language [20].

3.1.1 Representations

Packets are represented by the following data structure:

```
TYPE ptr_des_packet = ^ des_packet;
TYPE des_packet = RECORD
   destination : integer;
   time_of_birth : integer;
   routing_tag : array[0..MAXSTAGE] of integer;
   next : ptr_des_packet;
end;
SHARED VAR pool_of_packets : array[0..MAXPOOLSIZE] of des_packet;
```

The first attribute stores the location of the target memory. It is used for checking whether the packet correctly arrives at its destination memory. The second attribute is set to the simulation clock when the packet is generated. It is used for collecting statistics about packet delays. The attribute *routing_tag* is filled by the routing algorithm and accelerates the simulation run by providing each switch with its destination queue number. A pool of packets is statically allocated in shared memory.

Each DES switch has output FIFO queues which are represented as follows:

```
TYPE ptr_des_queue : ^ des_queue;
TYPE des_queue = RECORD
   head, tail : ptr_des_packet;
   numpackets : integer;
   received : boolean;
end;
```

The queue is implemented as a singly linked list. The attribute *received* is a dirty bit that prevents more than one packet from entering into the same queue during a network cycle.

Each DES switch is represented as follows:

```
TYPE des_switch = RECORD
   output_queues : array[0..MAXFANOUT] of des_queue;
   ptr_predecessor_queues : array[0..MAXSPREAD] of ptr_des_queue;
end;
SHARED VAR des_switches : array[0..MAXSWITCHES] of des_switch;
```

The first attribute represents the output queues of each DES switch. They can accept packets up to *max_buffer_size* which is an input parameter. The second attribute is a set of pointers to the predecessor output queues and plays the following roles:

1. It reflects topological differences among networks with different parameters. For example, its values for an SW-banyan are different from those for a CC-banyan. These values are initialized by the construction algorithms.

2. It separates the simulation code for processes from the construction algorithms. Specifically, the code stays the same regardless of the network choice.
3. A packet hop can be easily simulated by deleting the packet from a predecessor output queue and inserting it into its destination output queue.

Each processor is represented as follows:

```
TYPE des_processor = RECORD
   output_queue : des_queue;
end;
SHARED VAR des_processors : array[0..MAXPROCESSORS] of des_processor;
```

We assume that the size of the processor queue is infinite. Otherwise, we cannot measure the maximum throughput of the network. A newly generated packet is entered into the queue at the end of each cycle.

Each memory is represented as follows:

```
TYPE des_memory = RECORD
   ptr_predecessor_queue : ptr_des_queue;
end;
SHARED VAR des_memories : array[0..MAXMEMORY] of des_memory;
```

We assume that a packet can be absorbed by a memory in a single network cycle. A packet which arrives at a memory is returned to the pool.

3.1.2 Processes

At the beginning of a simulation run, a main process holds control. It accepts the input parameters and initialize system variables like the system clock, statistical counters, and process communication facilities. Then, it builds the user specified network by calling the construction algorithm. Next, it creates *clock_manager*, *memory*, *des_switch*, and *processor* processes and passes control to the clock_manager process. The main process also sends a proper index to each component process so that it can access the

data structure allocated for it. For example, the des_switch process with index i may access only the attributes in des_switches[i].

The clock_manager increases the simulation clock at the end of each simulation cycle and checks the termination condition. If it should terminate the current simulation run, it passes control to the main process. Otherwise, it passes control to the memory processes. Each component process simulates a set of components in the same level. Also, component processes in the same level can run in parallel. For simplicity, we assume that a component process simulates a single component. Each memory process pulls and frees a packet from its corresponding switch in the base level, and collects statistics about packet delays. Then, it passes the control to the des_switch processes in the base level. Each des_switch process examines the routing_tag of the packet on the front of each predecessor queue in some order determined by the user-specified conflict resolution scheme. It moves the packets from its predecessors to its internal queues if there is space, and passes control to its predecessors. In this way, the control propagates from the des_switch processes in the base level to those in the apex level. Then, the processor processes gain control. Each processor process generates a packet according to the birth rate and puts it into its queue. The destination tag of the packet is determined by the reference matrix and the routing algorithm. Finally, control is passed back to the clock_manager and the current simulation cycle ends.

3.2 Simulation Method for SEFD networks

3.2.1 Representations

Packets are represented by the following data structure:

```
TYPE ptr_sefd_packet = ^ sefd_packet;
TYPE sefd_packet = RECORD
    destination : integer;
    time_of_birth : integer;
    gdist : integer;
    upgoing_tag : array[0..MAXSTAGE] of integer;
    downgoing_tag : array[0..MAXSTAGE] of integer;
    next : ptr_sefd_packet;
```

```
end;
SHARED VAR pool_of_packets : array[0..MAXPOOLSIZE] of sefd_packet;
```

The attribute *gdist* stores the minimum level number in which the reflection switch exists. The attribute *upgoing_tag* is the routing tag for upgoing packets and the attribute *downgoing_tag* is for downgoing packets. Packets with *gdist* greater than 0 are transmitted up. Otherwise, they are either reflected or transmitted down. Switches decrease *gdist* by one as they transmit packets up. A pool of packets is statically allocated in shared memory.

Each SEFD switch has output FIFO queues which are represented as follows:

```
TYPE ptr_sefd_queue : ^ sefd_queue;
TYPE sefd_queue = RECORD
   head, tail : ptr_sefd_packet;
   numpackets : integer;
   received : boolean;
   sent : boolean;
end;
```

The queue is implemented as a singly linked list. The attribute *received* is a dirty bit that prevents more than one packet from entering into the same queue in a single network cycle. Similarly the attribute *sent* is a dirty bit that prevents more than one packet from leaving the same queue in a single network cycle.

Each SEFD switch is represented as follows:

```
TYPE sefd_switch = RECORD
   T_queues : array[0..MAXSPREAD] of sefd_queue;
   B_queues : array[0..MAXFANOUT] of sefd_queue;
   ptr_SH_queues : array[0..MAXSPREAD] of ptr_sefd_queue;
   ptr_SL_queues : array[0..MAXFANOUT] of ptr_sefd_queue;
end;
SHARED VAR sefd_switches : array[0..MAXSWITCHES] of sefd_switch;
```

The first two attributes represent the output queues of each SEFD switch. The attribute *T_queues* stores upgoing packets and *B_queues* stores downgoing packets. Both can accept packets up to *max_buffer_size* which is an input parameter. The attribute *ptr_SH_queues* is a set of pointers to *B_queues* in switch SHs and is used for receiving downgoing packets. The attribute *ptr_SL_queues* is a set of pointers to *T_queues* in switch SLs and is used for receiving upgoing and reflecting packets.

Each processor is represented as follows:

```
TYPE sefd_processor = RECORD
   output_queue : sefd_queue;
   ptr_B_queue : ptr_sefd_queue;
end;
SHARED VAR sefd_processors : array[0..MAXPROCESSORS] of sefd_processor;
```

Each processor has its internal queue whose size is infinite and a pointer to a *B_queue* in the corresponding switch. We assume that a packet can be absorbed by a processor in a single network cycle.

3.2.2 Processes

A main process creates *clock_manager*, *processor*, and *sefd_switch* processes with proper indices and passes control to the clock_manager process. The clock_manager checks the termination condition and passes control to the processor processes. Each processor process frees a packet from its corresponding switch in the base level, and collects statistics about packet delays. While control propagates from the sefd_switch processes in the base level to those in the apex level, downgoing and reflecting packets are transmitted. Then, control propagates from the sefd_switch processes in the apex level to those in the base level. During this period, each sefd_switch process transmits upgoing packets. Then, the processor processes gain control. Finally, each processor process generates a packet and passes control to the clock_manager and the current simulation cycle ends.

3.3 Simulation Method for SEHD networks

3.3.1 Representations

Packets are represented by the same data structure for SEFD networks.

```
TYPE ptr_sehd_packet = ^ sefd_packet;
TYPE sehd_packet : sefd_packet;
SHARED VAR pool_of_packets : array[0..MAXPOOLSIZE] of sehd_packet;
```

Each SEHD switch has special queues which are represented as follows:

```
TYPE ptr_sehd_queue : ^ sehd_queue;
TYPE sehd_queue = RECORD
   ULIST_head, ULIST_tail : ptr_sehd_packet;
   ULIST_numpackets : integer;
   DLIST_head, DLIST_tail : ptr_sehd_packet;
   DLIST_numpackets : integer;
   received : integer;
   sent : integer;
end;
```

There are two FIFO queue structures; one for ULIST and the other for DLIST. The total number of packets in both lists cannot exceed *max_buffer_size* which is an input parameter. The attribute *received* and *sent* are dirty bits. Together they prevent the special queue from accepting or sending two packets in a single network cycle.

Each SEHD switch is represented as follows:

```
TYPE sehd_switch = RECORD
   T_queues : array[0..MAXSPREAD] of sehd_queue;
   ptr_SL_queues : array[0..MAXFANOUT] of ptr_sehd_queue;
end;
SHARED VAR sehd_switches : array[0..MAXSWITCHES] of sehd_switch;
```

The attribute *ptr_SL_queues* is a set of pointers to *T_queues* in switch SLs and is used for receiving upgoing and reflecting packets.

Each processor is represented as follows:

```
TYPE sehd_processor = RECORD
   output_queue : sehd_queue;
end;
SHARED VAR sefd_processors : array[0..MAXPROCESSORS] of sefd_processor;
```

Each processor has its internal queue whose ULIST has infinite capacity. We assume that a packet can be absorbed by a processor in a single network cycle. Thus, its DLIST is always empty.

3.3.2 Processes

There are main, *clock_manager*, *processor*, and *sehd_switch* processes. They do the similar operations to those for the SEFD protocol.

4 Simulation Results

Two widely used techniques for analyzing the performance of packet-switching interconnection networks are queueing network models [3,30,19] and discrete Markov chains [3,7,24,26]. Baskett, Chandy, Muntz, and Palacios-Gomez [1] greatly extended the class of queueing networks that can be solved by computationally tractable numerical algorithms. However, their stochastic assumptions such as exponential arrival and service rates and no conflicts among customers heading for the same service center at the same time make it hard to apply their algorithms to our networks directly. Meanwhile, Markov chains often possess too many states to be solved efficiently; the number of states grows exponentially with respect to the number of queues. Therefore, some assumptions are usually made to isolate a queue from its correspondents. In this work, we give the simulation results with intuitive explanation because we did not make such assumptions.

Figure 13 shows performance of the three protocols on a (2,2,7) rectangular CC-banyan network with infinite buffer size. Unless otherwise stated, a uniform traffic pattern is used in our simulations. The performance of the

DES protocol is slightly better than that of the SEFD protocol for the case of low birth rate. That is because the average path length that a packet must travel in the DES network is shorter than that in the SEFD network. The SEHD network shows poor performance because packets compete not only for queue space on their paths, but also for the links between two corresponding queues. In the worst case, there may be three packets competing for a single link. However, the SEFD network costs twice as much as the SEHD network. In Figure 14, we normalized the performance of the SEHD network by a factor of two. The graph shows that the SEFD network still performs better than the SEHD network.

Figures 15, 16, and 17 show how the performance of the three protocols varies as a function of buffer size. Networks with buffer size 8 or 16 show acceptable performance compared to the case of infinite buffer size. Practically speaking, there is little additional overhead in having more memory space, so we choose 16 as a proper buffer size. From this point, we set the default buffer size to 16.

Figure 18 compares the performance of a (2,2,7) DES CC-banyan network with that of a (2,2,7) DES SW-banyan network under a uniform traffic pattern. There is little difference between their performance because in both networks there is a unique fixed-length (8 in this case) path between a processor and a memory. But as shown in Figures 19 and 20, a (2,2,7) single-ended CC-banyan network shows better performance than its counterpart SW-banyan network. This is because, for a given level, there are about two times more reachable base nodes from a given base node in the CC-banyan network than in the comparable SW-banyan network.

Figure 21 and 22 show the performance of a (2,2,7) single-ended CC-banyan network under a local traffic pattern. The local traffic pattern is caused by localizing program modules into a set of processors which are closely located to each other. Here, we assume that the local traffic pattern is represented by the reference matrix whose $(i,j)^{\text{th}}$ element is the probability that a source i accesses a sink j, which is inversely proportional to the minimum level number in which at least one reflection node for i and j exists. The figures show that the localized traffic improves performance of the networks. That is because under local traffic, the average path length that a packet must travel becomes shorter than that under uniform traffic. However, as shown in Figure 23, the same traffic pattern can degrade per-

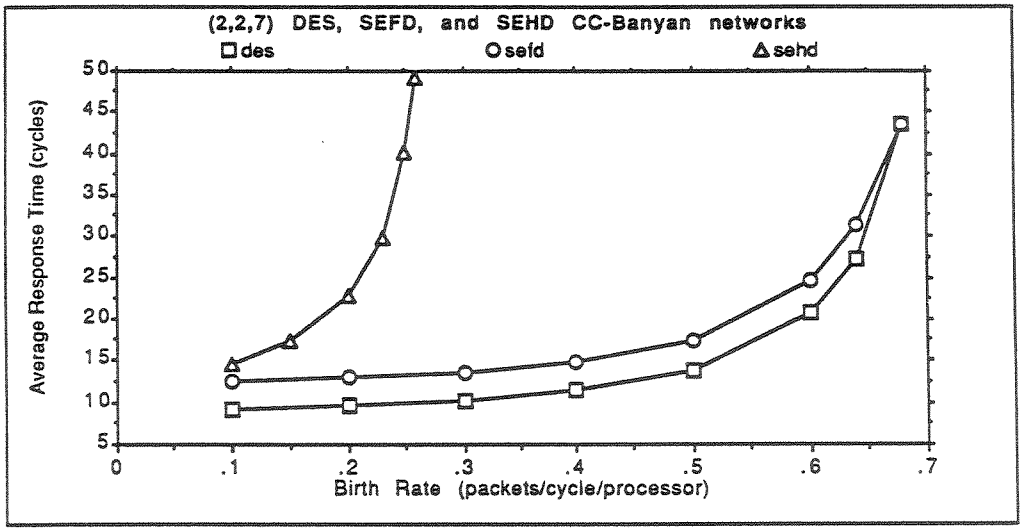


Figure 13: Performance of the three protocols

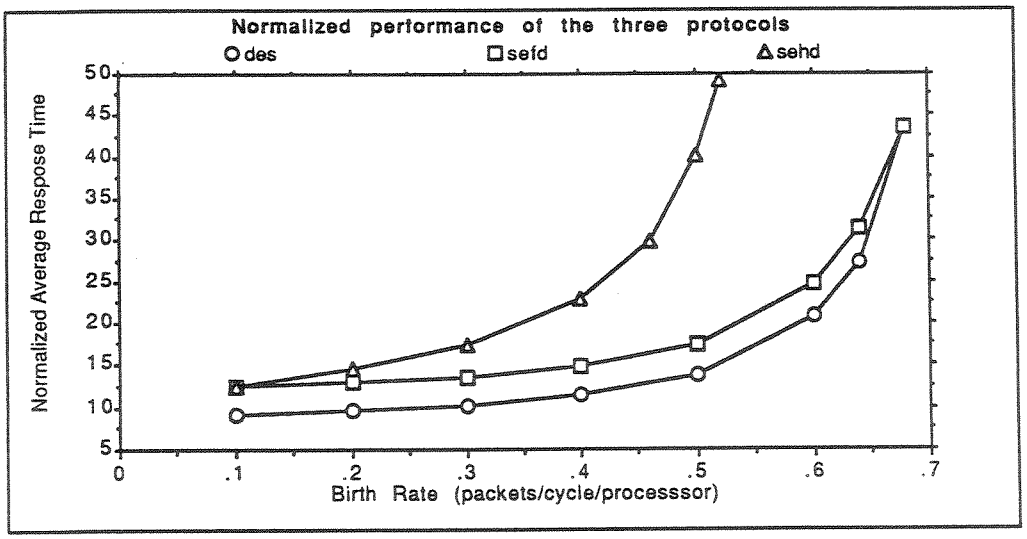


Figure 14: Normalized performance of the SEHD protocols

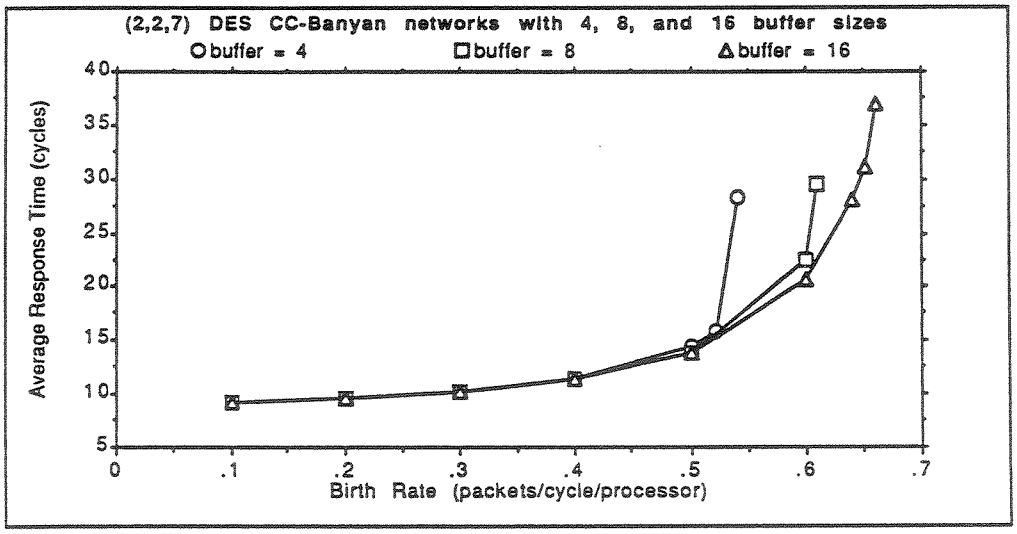


Figure 15: Performance of a DES network as a function of buffer size

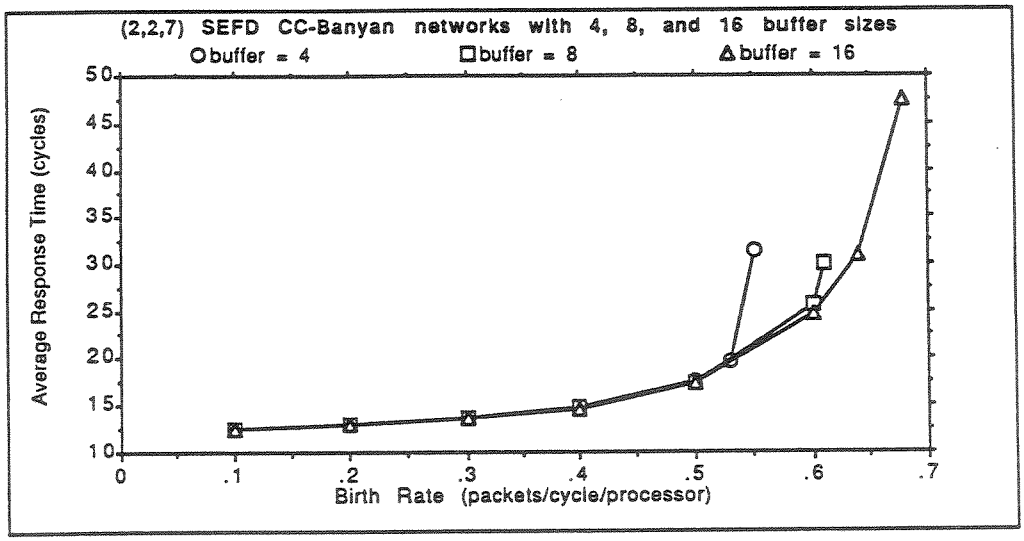


Figure 16: Performance of a SEFD network as a function of buffer size

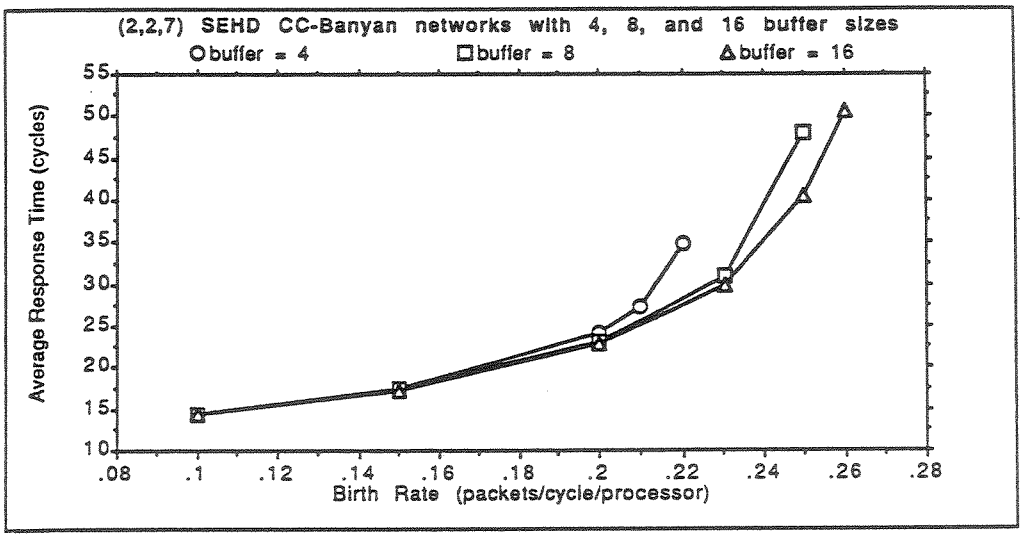


Figure 17: Performance of a SEHD network as a function of buffer size

formance of a (2,2,7) DES CC-banyan network. Under this traffic pattern, the i^{th} processor is most busy sending packets to the i^{th} memory for each i, where $0 \leq i < 256$. This permutation is not allowed in CC-banyan networks, therefore, programmers must be careful to place data into memory so that the memory is uniformly accessed by the processors at run time.

In this work, we consider three conflict resolution schemes: random (RAND), fixed priority (FP), and round robin (RR). Figure 24 shows that the performance of RAND is as good as that of RR, and that both are better than that of FP. While switches with RR must maintain a state variable to keep information about the order of processing grant and request signals, switches with RAND can process grant and request signals on FCFS basis. Therefore, RAND is the best strategy in terms of performance and hardware cost among the three.

Figure 25 shows the performance of SEFD CC-banyan networks as a function of the number of stages. As expected, the performance of the networks degrades as the number of stages increases because packets experience more conflicts while they are transmitted through the larger networks.

Figure 26 shows how SEFD CC-banyan networks perform as a function of switch size. For the case of birth rate being less than 0.5 packets/cycle/processor, networks with larger switch sizes show better response time because they have fewer levels for a fixed number of processors. However, as the birth rate increases over 0.5 packets/cycle/processor, the larger the switch size is, the more contentions are expected. For example, for a f by f switch, there may be $2 * f - 1$ packets heading for the same queue at the same time in the worst case (this is not true for an n by n SEFD crossbar switch, where n is the number of processors). This can degrade the performance of networks with larger switch sizes.

Figures 27 and 28 show the performance of nonrectangular SEFD CC-banyan networks. It degrades as the ratio (R) of the number of apex nodes to the number of base nodes decreases. This is because the number of reflection nodes decreases as R decreases. In Figures 29 and 30, we normalized the performance of nonrectangular networks by comparing their costs with their rectangular counterparts. The cost of each network is calculated by using the following cost function:

$cost = the\ number\ of\ switches * spread * fanout$

Figures 31 and 32 show the performance of (2,2,7) DES and SEFD CC-

banyan networks under a 2% hot spot traffic to memory 0 or a uniformly selected destination for infinite, 50, and 100 network cycles. They show that as the duration of accessing a hot spot lasts longer, the performance of the networks degrades more even if all memories are evenly accessed in the steady state. Both networks suffer equally from hot spot traffic.

Finally, Figure 33 shows the performance of a (2,2,7) SEFD CC-banyan network under a bursty traffic. In this traffic pattern, processors generate 32 contiguous packets once they commence to generate packets. The idea behind this is that the lengths of messages used by packet-switching telephone networks or some message-passing computer systems range from a few hundred bits to 10,000 bits. If we assume that our networks are used for those applications, that we choose 1,024 bits as the message length, and that our networks can transmit 32 bits a network cycle, then the processors need to generate 32 contiguous packets per a message. The figure shows that the performance mildly degrades (especially the average response time) because the bursty traffic introduces more contention among packets.

5 Conclusions

Two approaches to evaluating performance of packet-switching interconnection networks are analytic performance modeling and simulation performance modeling. For analytic performance modeling, assumptions are required to make the analysis tractable. However, some of these assumptions are not realistic. In this work, we developed simulation performance models which relax these assumptions and significantly extend the logical design space.

SFL (Spread, Fanout, and Level) is a parallel time-driven simulation program for packet-switching synchronous banyan networks. It is written in PPL and runs on the Sequent Balance 8000. SFL provides users with a three dimensional design space. The first dimemsion provides a choice of either a CC- or an SW-banyan network. The second dimemsion provides a choice of tuples (spread, fanout, level). The last dimension provides a choice of three communication protocols: DES, SEFD, or SEHD. The primary goal of SFL is to ease the decision-making process of designing interconnection networks by providing performance estimates of various design alternatives.

We introduced switch models for DES, SEFD and SEHD protocols. We also presented our simulation method in depth.

We itemize our results as follows:

- The performance of SEFD networks is as good as that of DES networks under a uniform traffic pattern.
- SEHD networks perform worse than SEFD and DES networks, but cost half as much.
- Networks with buffer size 16 show acceptable performance compared to the case of infinite buffer size.
- The performance of DES CC-banyan networks is as good as that of DES SW-banyan networks.
- The performance of single-ended CC-banyan networks is better than that of single-ended SW-banyan networks.
- Under a local traffic pattern, the performance of single-ended networks improves.
- Under a local traffic pattern, the performance of DES networks may suffer from load unbalance.
- The random conflict resolution scheme is the best strategy among the three schemes in terms of performance and hardware cost.
- While networks with small switches show better throughput, networks with large switches show better response time.
- The performance of nonrectangular single-ended networks degrades as the ratio of the number of apex nodes to the number of base nodes decreases.
- The performance of both DES and SEFD networks suffers equally from hot spot traffic pattern.
- Under bursty traffic pattern, SEFD networks show mild degradation of performance. Thus, these computer networks may be appropriate for telephone networks.

In summary, we recommend SEFD CC-banyan networks as excellent interconnection networks for highly parallel multiprocessor systems.

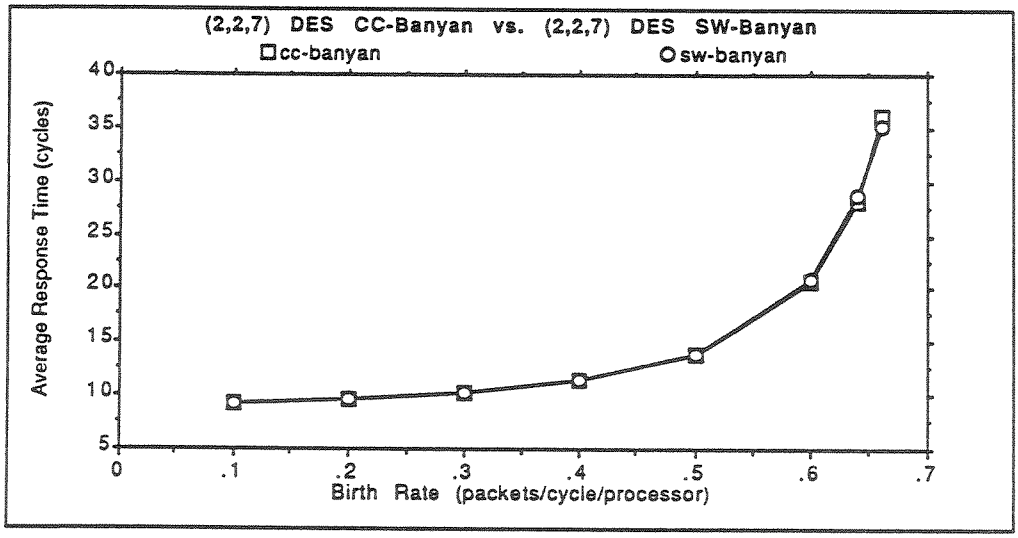


Figure 18: Performance of a DES CC- and SW-banyan network

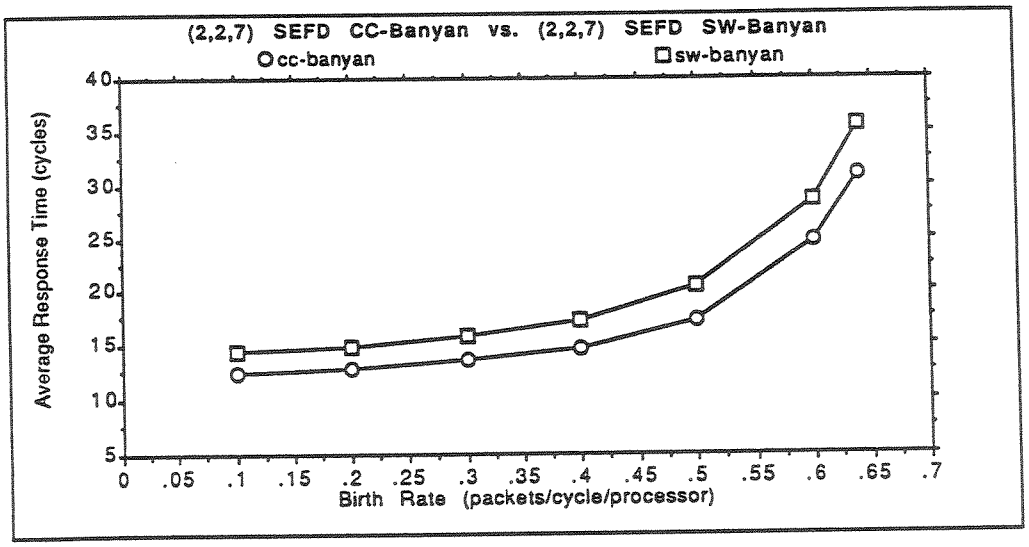


Figure 19: Performance of a SEFD CC- and SW-banyan network

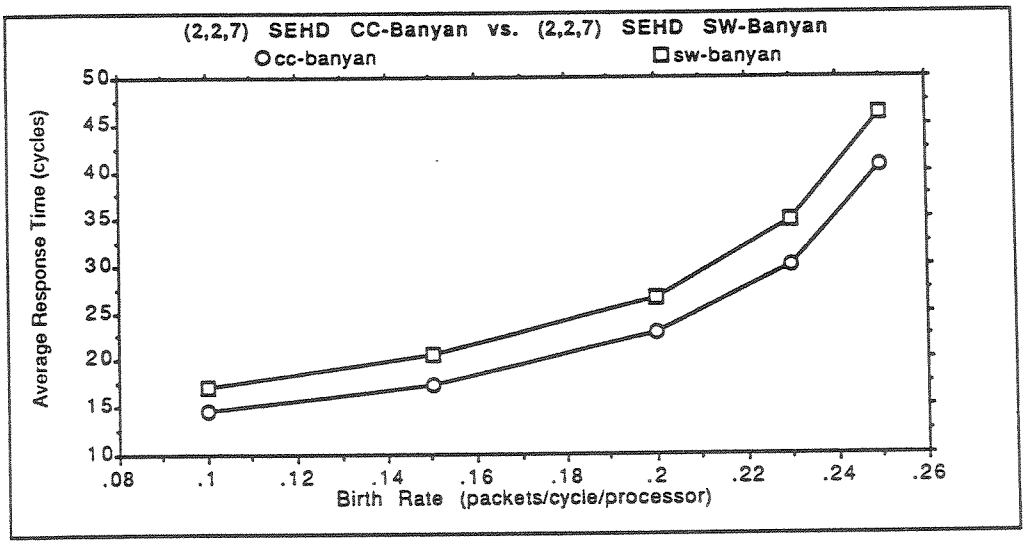


Figure 20: Performance of a SEHD CC- and SW-banyan network

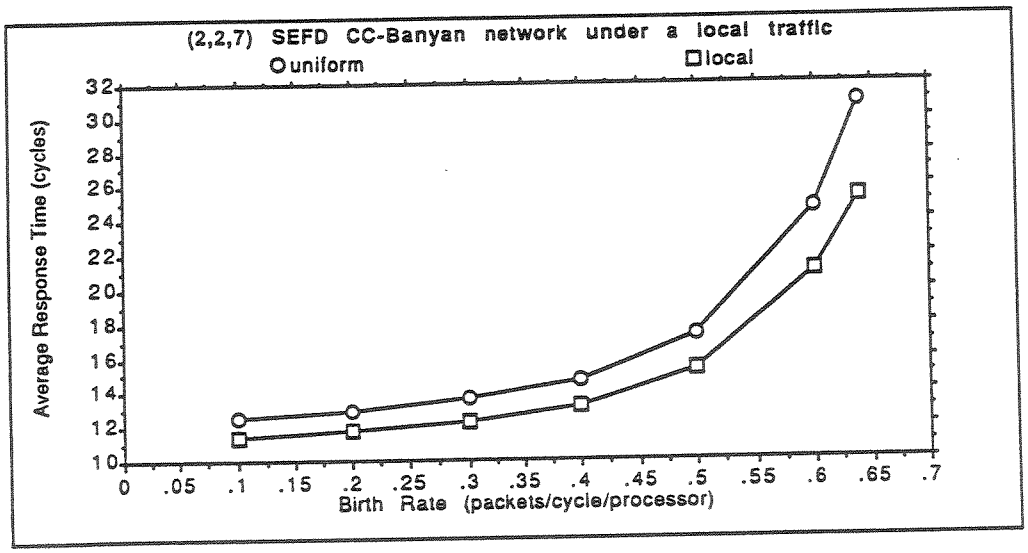


Figure 21: Performance of a SEFD CC-banyan network under a local traffic

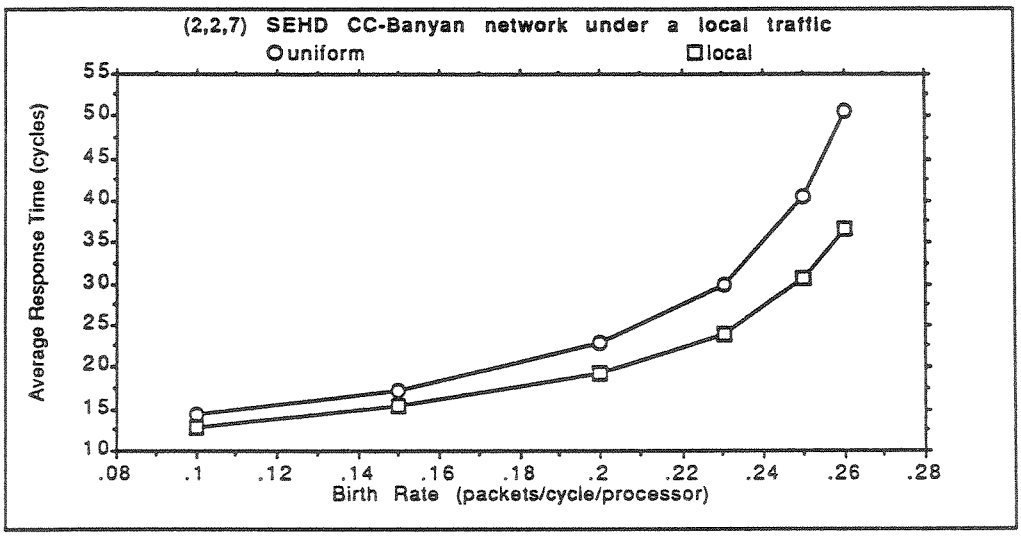


Figure 22: Performance of a SEHD CC-banyan network under a local traffic

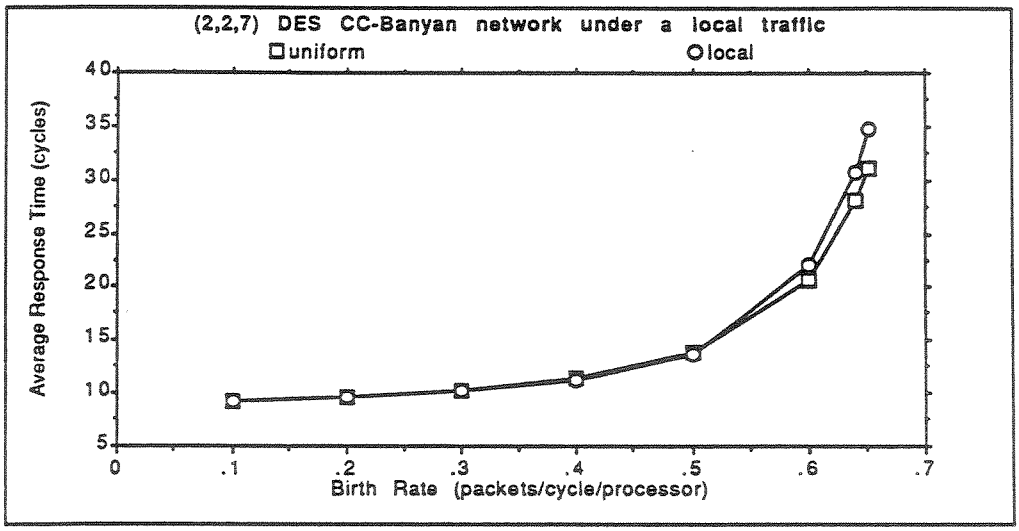


Figure 23: Performance of a DES CC-banyan network under a local traffic

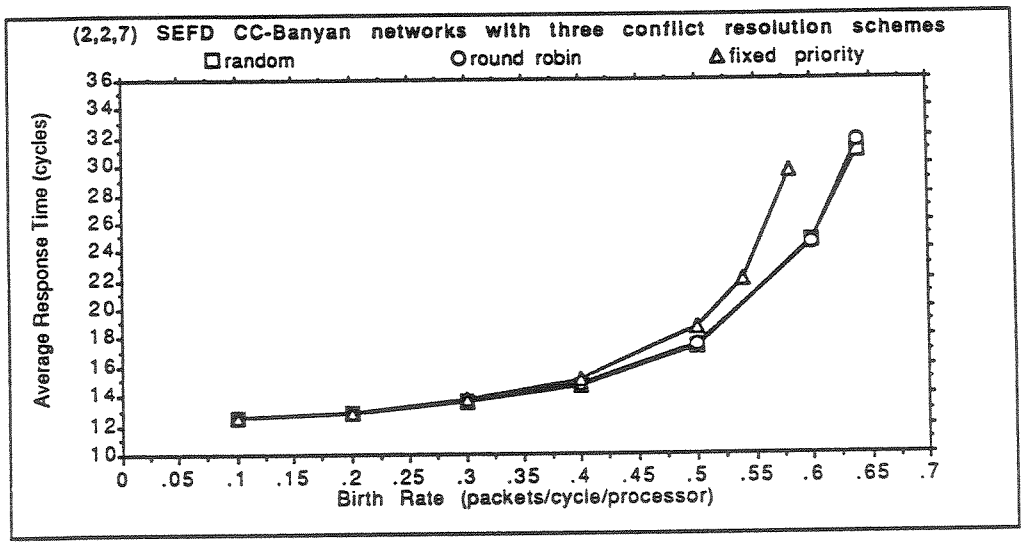


Figure 24: Performance of a SEFD CC-banyan network with three conflict resolution schemes

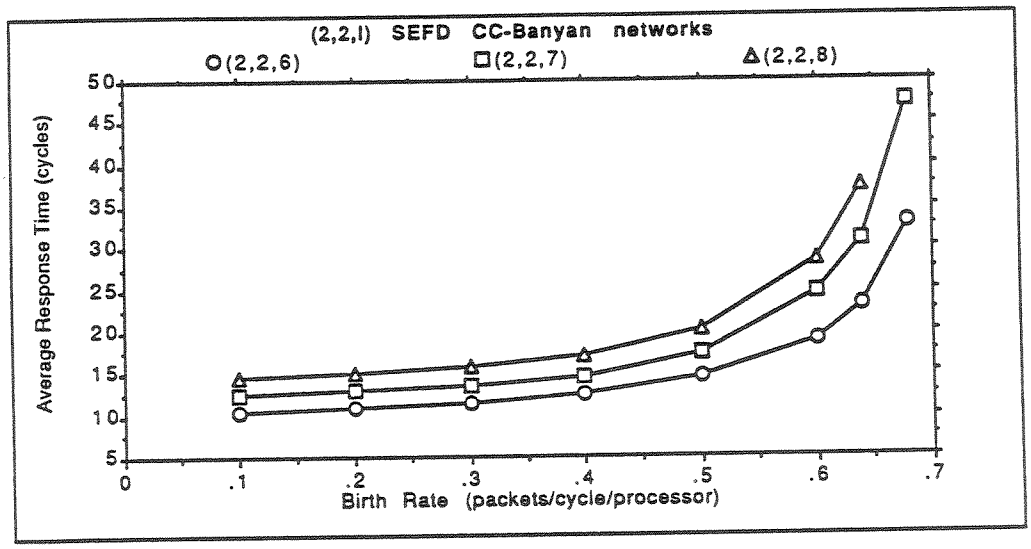


Figure 25: Performance of (2,2,l) SEFD CC-banyan networks

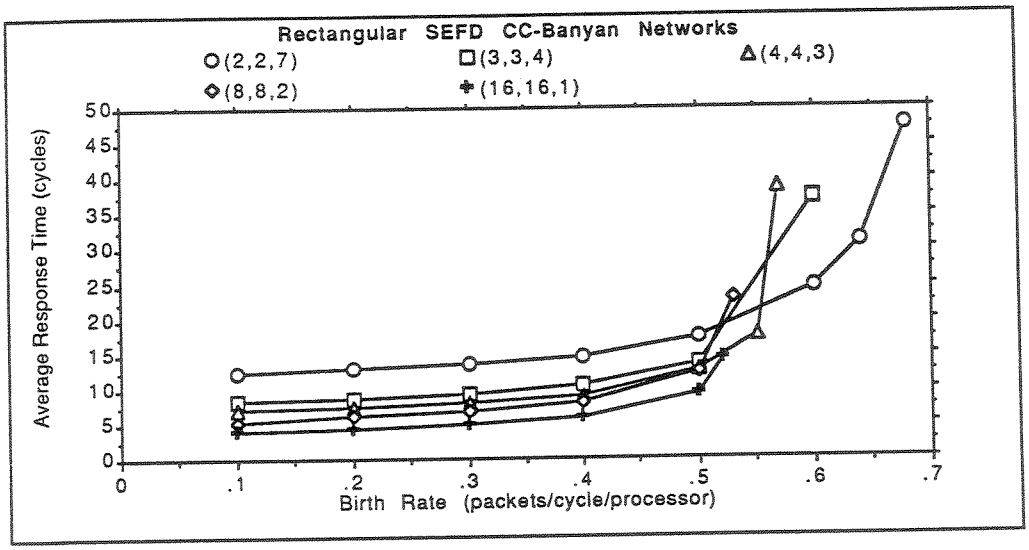


Figure 26: Performance of SEFD CC-banyan networks with various switch sizes

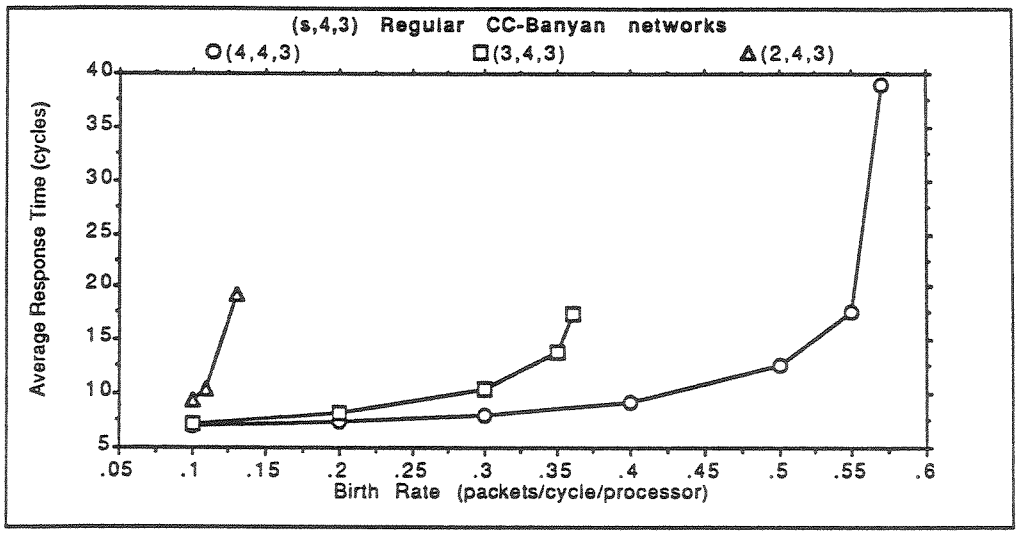


Figure 27: Performance of (s,4,3) regular SEFD CC-banyan networks

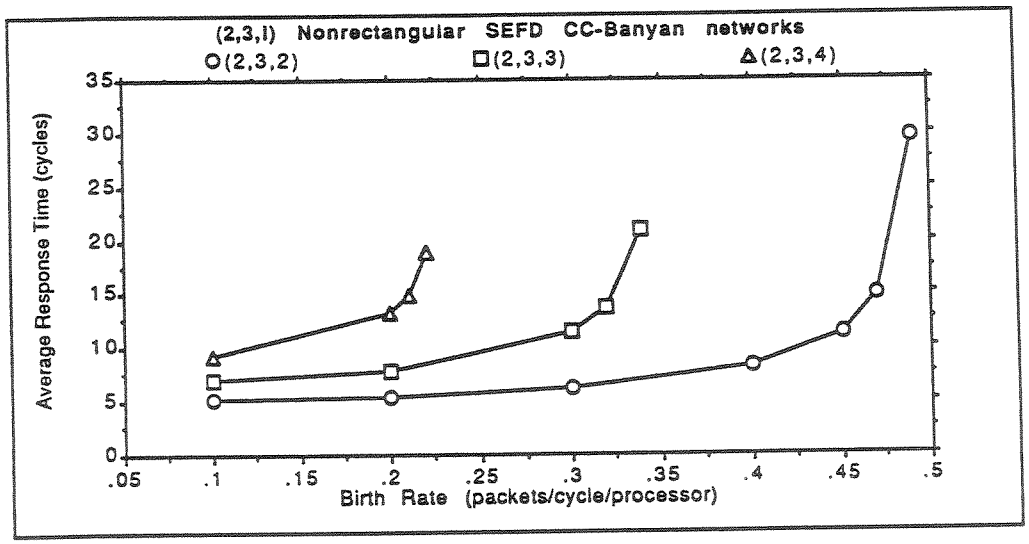


Figure 28: Performance of (2,3,l) nonrectangular SEFD CC-banyan networks

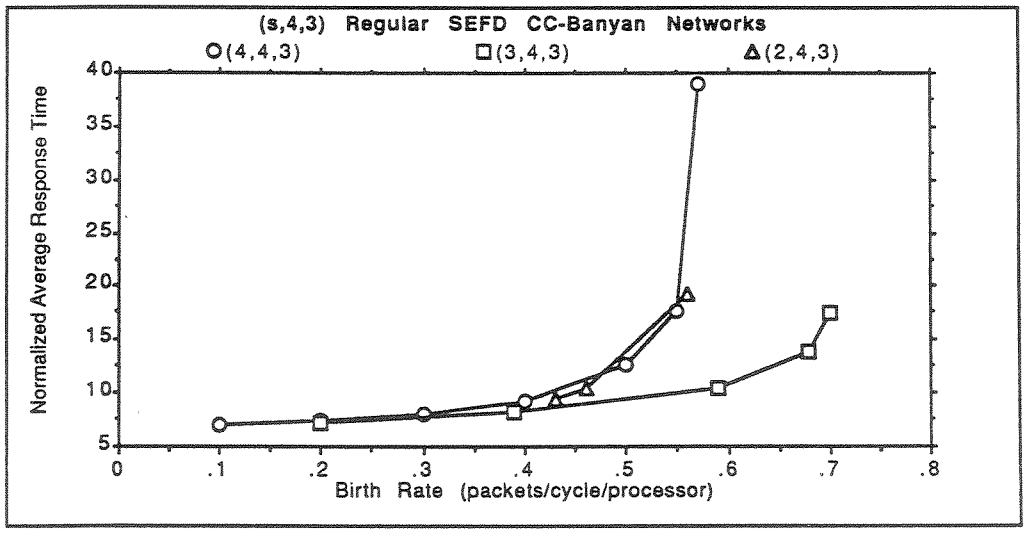


Figure 29: Normalized performance of (s,4,3) regular SEFD CC-banyan networks

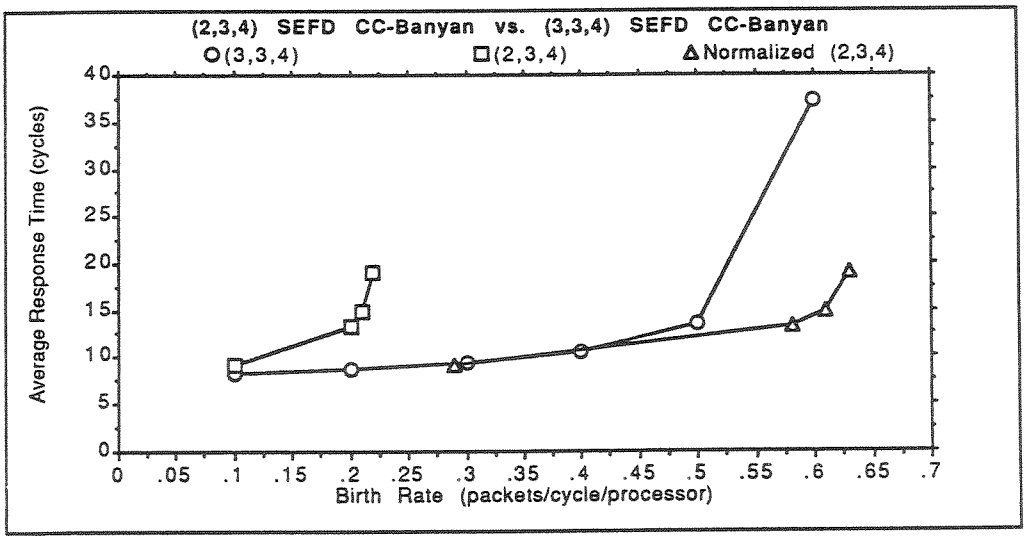


Figure 30: Normalized performance of a (2,3,4) nonrectangular SEFD CC-banyan network

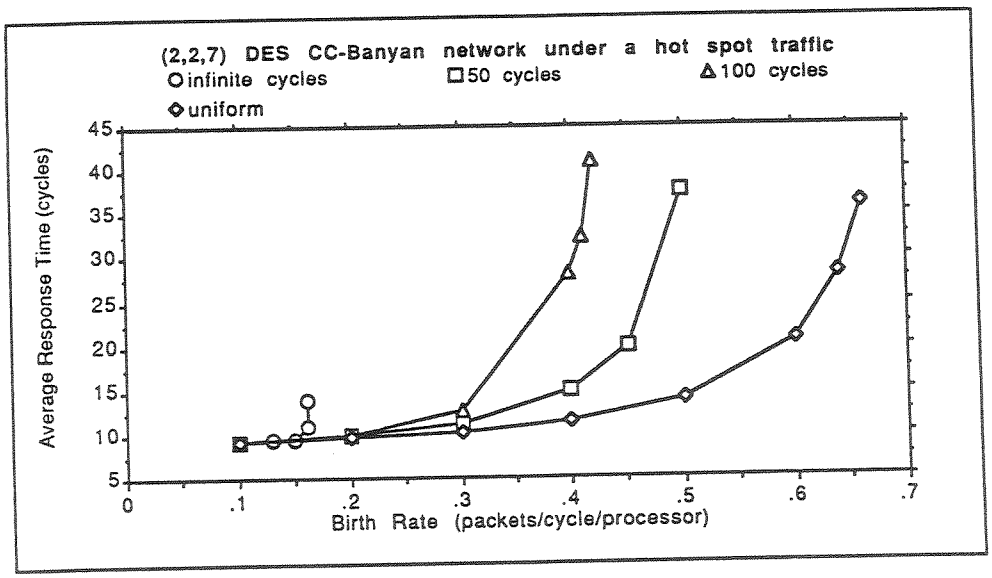


Figure 31: Performance of a (2,2,7) DES CC-banyan network under 2% hot spot traffic pattern

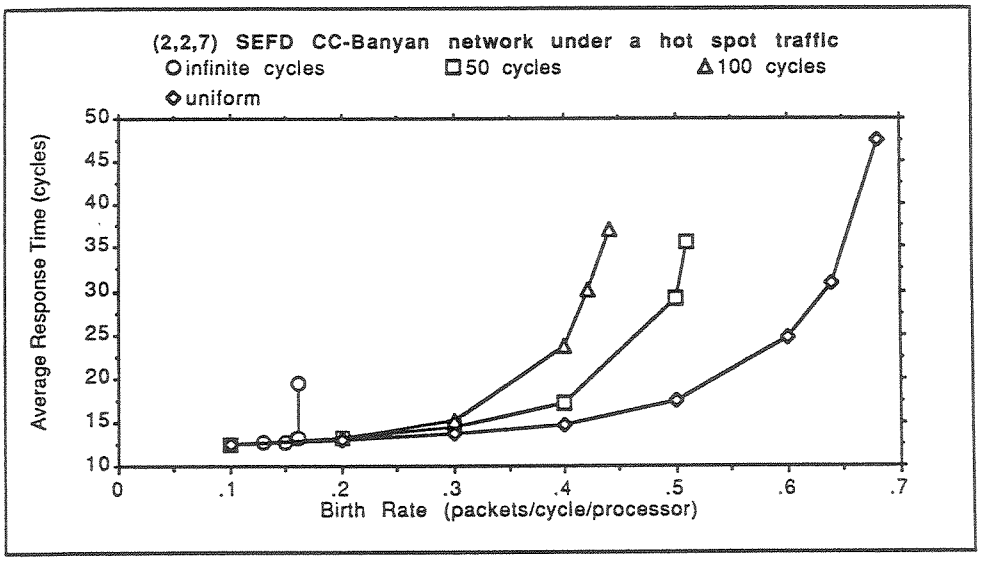


Figure 32: Performance of a (2,2,7) SEFD CC-banyan network under 2% hot spot traffic pattern

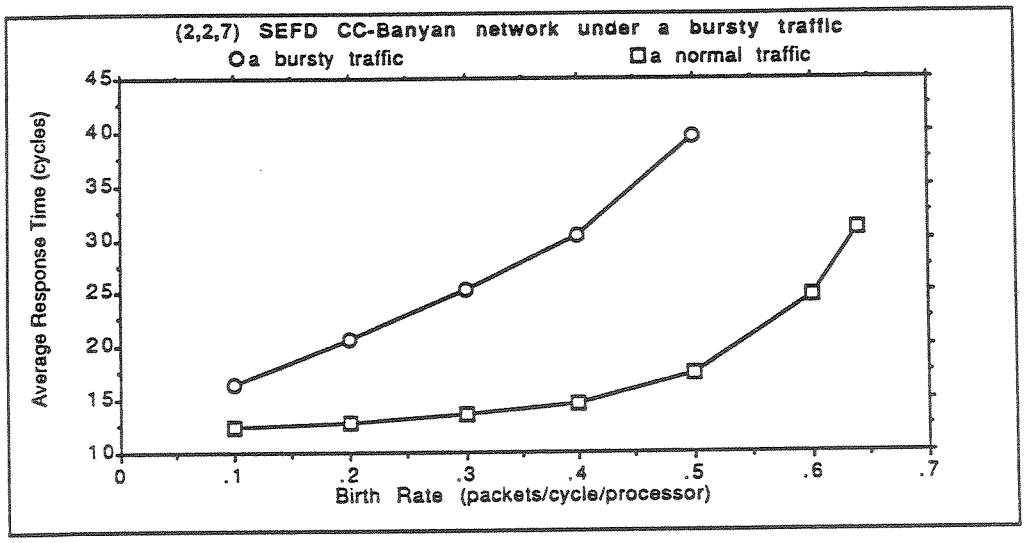


Figure 33: Performance of a (2,2,7) SEFD CC-banyan network under bursty traffic pattern

References

[1] Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios-Gomez, F., "Open, closed, and mixed networks of queues with different classes of customers," *J. Assoc. Comput.*, vol. 22, pp. 248-260, March 1975.

[2] Ben-Ari, M., *Principles of Concurrent Programming*, Prentice-Hall Inc., Englewood Cliffs, N.J. 1982.

[3] Bhandarkar, D.P., "Analysis of memory interference in multiprocessors," *IEEE Trans. on Computers*, Vol. C-24, No. 9, Sept. 1975.

[4] Bruell, S.C. and Balbo, G., *Computational Algorithms for Closed Queueing Networks*, Elsevier North Holland, Inc., 1980.

[5] Chang, D.Y., Kuck, D.J., and Lawrie, D.H., "On the effective bandwidth of parallel memories," *IEEE Trans. on Computers*, Vol. C-26, No. 5, May 1977.

[6] Cheemalavagu, S. and Malek, M., "Analysis and simulation in banyan networks with 2x2, 4x4, and 8x8 switching elements," *Proc. of Real-Time Syst. Symp.*, Dec. 1982, pp. 83-89.

[7] Chen, Pin-Yee, *Multiprocessor Systems: Interconnection Networks, Memory Hierarchy, Modeling and Simulations*, Ph.D. Diss., Department of Computer Science, University of Illinois at Urbana-Champaign, Dec. 1981.

[8] Cherkassky, V., *Performance and Reliability Evaluation of Banyan Networks*, Ph.D. Diss., University of Texas at Austin, 1985.

[9] Cherkassky, V. and Malek, M., "Analysis of CC-banyan networks," *Proc. of 1986 ICPP*, Aug. 1986, pp. 115-117.

[10] Chin, C.-Y., and Hwang, K., "Connection principles for multipath packet switching networks," *Proc. of 11th Annual Symposium on Computer Architecture*, May 1984, pp. 99-107.

[11] Christos, B., John, G., Paul, S., Vassilis, T., "Queueing delays in buffered multistage interconnection networks," *Proc. of 1987 ACM SIGMETRICS*, Vol. 15, No. 1, May 1987.

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

[13] DeMelo, J. and Jenevein, R., "SK-banyans: a unified class of banyan networks," *Proc. of 1986 ICPP*, Aug. 1986.

[14] Denning, P.J., "The working set model for program behavior," *Commum. of the ACM*, May 1968, pp.323-333.

[15] Deshpande, S.R., Jenevein, R., and Lipovski, G.J., "TRAC: an experience with a novel architectural prototype," *AFIPS Conf. Proc. 1985 NCC*, pp. 247-258.

[16] Dias, D.M. and Jump, J.R., "Analysis and simulation of buffered delta networks," *IEEE Trans. on Computers*, Vol. C-30, No. 4, April 1981.

[17] Gelernter, D., "A dag-based algorithm for prevention of store-and-forward deadlock in packet networks," *IEEE Trans. on Computers*, Vol. C-30, No. 10, Oct. 1981, pp. 709-715.

[18] Goke, L.R. and Lipovski, G.J., "Banyan networks for partitioning multiprocessor system," *Proc. of First Annual Symposium on Computer Architecture*, Dec. 1973, pp. 21-28.

[19] Gottlieb, A., Grishman, R., Kruskal, C.P., McAuliffe, K.P., Rudolph, L., and Snir, M., "The NYU Ultracomputer—Designing an MIMD shared memory parallel computer," *IEEE Trans. on Computers*, Vol. C-32, No. 2, Feb. 1983, pp. 175-189.

[20] Hansen, P.B., "The programming language concurrent Pascal," *IEEE Trans. on Software Engineering*, Vol. SE-1, No. 2, June 1975.

[21] Heidelberger, P. and Lavenberg, S.S., "Computer performance evaluation methodology," *IEEE Trans. on Computers*, Vol. C-33, No. 12, Dec. 1984.

[22] Hoogendoorn, C.H., "A general model for memory interference in multiprocessors," *IEEE Trans. on Computers*, Vol. C-26, No. 10, Oct. 1977.

[23] Karp, A.H., "Programming for parallelism," *IEEE Computer*, May 1987, pp. 43-57.

[24] Kleinrock, L., *Queueing Systems, Volume 1: Theory*, John Wiley & Sons, Inc., 1975.

[25] Kleinrock, L., *Queueing Systems, Volume 2: Computer Applications*, John Wiley & Sons, Inc., 1976.

[26] Kruskal, C.P. and Snir, M., "The performance of multistage interconnection networks for multiprocessors," *IEEE Trans. on Computers*,

Vol. C-32, No. 12, Dec. 1983.

[27] Kumar, M. and Pfister, G.F., "The onset of hot spot contention," *Proc. of 1986 ICPP*, Aug. 1986, pp. 28-34.

[28] Lam, S.S., *Tutorial: Principles of Communication and Networking Protocols*, IEEE Computer Society Press, Los Angeles, 1984.

[29] Law, A.M. and Kelton, W.D., *Simulation Modeling and Analysis*, McGraw-Hill, Inc., 1982.

[30] Lazowska, E.D., Zahorjan, J., Graham, G.S., and Sevcik, K.C., *Quantitative System Performance, Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[31] Lea, C.A., "The load-sharing banyan network," *IEEE Trans. on Computers*, Vol. C-35, No. 12, Dec. 1986, pp. 1025-1034.

[32] Lee, G., Kruskal, C.P., and Kuck, D.J., "The effectiveness of combining in shared memory parallel computers in the presence of hot spots," *Proc. of 1986 ICPP*, Aug. 1986, pp. 35-41.

[33] Lee, M. and Wu, C.L., "Performance analysis of circuit switching baseline interconnection networks," *Proc. of 11th Annual International Symposium of Computer Architecture*, 1984, pp. 82-90.

[34] Lipovski, G.J., and Malek, M., *Parallel Computing: Theory and Comparisons*, John Wiley & Sons, Inc., New York, 1987.