Computer Science Department Technical Report CSD-960032
University of California, Los Angeles, August 1996.

Deadlock Resolution in Networks Employing Connection-Based Adaptive Routing

Yoshio F. Turner and Yuval Tamir
Computer Science Department
UCLA
Los Angeles, CA 90095

Abstract

Adaptive routing is used in networks in order to achieve fault tolerance and optimize performance for all traffic patterns. Most adaptive routing schemes route each packet independently, requiring addressing and sequencing information to be transmitted and processed for each packet. Connection-based message transport, such as virtual circuits, reduces the overhead for routing packets through the network and maintain FIFO packets delivery. Dynamic Virtual Circuits (DVCs) combine the advantages of adaptive routing and connection-based message transport. When used with arbitrary topologies and without restricting the routes of packets, networks employing DVCs are susceptible to deadlocks. In this paper we investigate a scheme for resolving such deadlocks. The scheme takes into account dependencies introduced by the DVC mechanism as well as the inherent dependencies among packet buffers at the input ports of each switch. Detailed simulations of the scheme have been performed using a multi-threaded event-driven simulator. Preliminary performance results are presented.

Keywords: adaptive routing, deadlock, virtual circuits.

I. Introduction

The performance and reliability of multicomputers, composed of computing nodes interconnected by point-to-point links, is dependent on the performance and reliability of the interconnection network used for interprocessor communication. The routing scheme used in these networks should direct packets through the lowest latency paths from source to destination. It should take into account the topology of the network and *adapt* to the current workload and resource availability to route packets around congested or faulty areas [11, 9, 2].

In order to maximize the performance of a network, it is important to minimize the addressing and control information that must be sent with each packet. This is achieved in *connection-based* routing, where once a path between a source and destination are established, packets can be sent along the path without the addressing and sequencing information needed in packet switched networks. With *virtual circuits* [10, 1] these advantages are achieved but resources are better utilized since the physical resources (buffers, links, etc) are multiplexed between *active* connections.

With virtual circuits (as well as with physical circuit switching) it is possible to achieve good performance even in irregular topologies, where simple algorithmic routing is not possible. For such topologies, routing is based on tables, which are constructed when the system is initialized and may be changed later, as the load on the system changes [11]. Once a virtual circuit is established, forwarding of a packet along its path can be done very quickly — a single lookup in a small table at each hop.

The key problem with conventional virtual circuits is that the paths are static and cannot be easily changed in response to congestion or failure. We have previously introduced a new message transport mechanism, called *Dynamic Virtual Circuits* (DVCs) [12], that has the advantages of virtual circuits but allows individual nodes to make a local decision to break or reroute an existing circuit. Each node maintains sufficient information to re-establish broken circuits while preserving the FIFO ordering of packets. Since routing with Dynamic Virtual Circuits is based on tables, the scheme does not depend on a regular system topology. Hence, efficient routing can be performed even after a large number of system nodes or links have failed.

If the paths taken by packets are not constrained, the network may be susceptible to deadlocks, due to circular dependencies among buffers and network resources [4]. For store-and-forward networks consisting of switches with central packet buffers, it has been shown that this problem can be overcome by detecting and resolving deadlocks when they occur [6, 3]. However, for high-performance switches in multicomputer interconnection networks, it is generally more efficient to use separate buffers at each input port [13] and *virtual cut-through* routing [7]. Furthermore, some the operation involved in manipulating DVCs may introduce additional dependencies, which are not taken intro account in the scheme for centrally-buffered packet-switched networks [6, 3].

This paper presents a deadlock detection with resolution scheme for networks employing dynamic virtual circuits, composed of switches with input buffers. With this scheme, there is the potential of obtaining the advantages virtual circuits and of adaptive routing in irregular topologies (or regular topologies following node or link failures) without constraining packet routes. Detailed cycle-by-cycle simulation of a network employing this scheme was implemented on a multi-threaded, object-oriented, even-driven simulator. This simulation validated the correctness of the scheme and allowed us to perform preliminary performance evaluation.

Dynamic Virtual Circuits are explained in Section II. The deadlock detection and resolution scheme for packet switching centrally-buffered networks is described in Section III. An adaptation of this scheme to networks composed of switches with input buffers is discussed in Section IV. In Section V, the additional dependencies caused by the DVC mechanism are described and a complete scheme for detecting and resolving deadlocks in a network with DVCs is presented. Simulation and preliminary performance evaluation of the scheme is presented in Section VI.

II. Dynamic Virtual Circuits

Virtual circuits are logical connections set up between network sources and network destinations. Each physical link can carry multiple virtual circuits at once, one on each ‘‘virtual channel,’’ where the virtual channel is simply an identifier carried in a header field of each packet. A virtual circuit is a sequence of virtual channels along a physical path from source to destination. Each virtual circuit is set up by a Circuit Establishment Packet (CEP) that travels from the source to the destination, allocating a virtual channel to the circuit from each physical link it traverses on its path. The mapping from each virtual channel at the input port to an output port and some virtual channel of the output port is recorded at circuit establishment time in the Input Mapping Table at each input port. When an incoming packet arrives as part of an established circuit, the Input Mapping Table is accessed both to route the packet to the correct output port and to determine the new output virtual channel number placed in the packet’s header. The packet can be forwarded using virtual cut-through switching for high performance [7]. Virtual circuits are attractive because they provide fast routing of packets once circuits are set up (especially for networks of arbitrary topology with large, relatively slow off-chip routing tables), and they require very low overhead in the header of each packet for addressing and sequencing (namely, just the virtual channel number).

With traditional static virtual circuit switching, each established circuit holds the same resources throughout its life. A circuit cannot adapt to failures or congestion by changing the physical path it uses. Once the virtual channels on a physical link are exhausted, no new circuits can use that link until one of the virtual circuits voluntarily disestablishes itself completely. The Dynamic Virtual Circuits (DVC) switching technique modifies the traditional method to one that allows circuits to be disestablished from intermediate nodes and possibly rerouted onto new paths, all without involving the source node [12]. Hence, the circuit fragment from the source node to the intermediate node remains intact for possible use if the complete circuit needs to be reestablished. Needed channels are released immediately following a local decision at an intermediate node. Tearing a circuit requires the creation of a Circuit Destruction Packet (CDP) that travels from the point of teardown to the destination, releasing a held virtual channel on each hop.

When a DVC is first established, information regarding the ultimate destination (node identifier) of the new circuit is kept at each intermediate node. When a packet on a DVC arrives at a node where the DVC was previously cut, the information regarding the ultimate destination of the cut circuit is used to reestablish the DVC. The node chooses an output port on which to reestablish the cut circuit, creates a CEP, updates the mapping tables, and sends the CEP, reestablishing the circuit on the new path to its destination. The data packet that triggered the reestablishment of the circuit as well as future packets on the same circuit can then be sent along the new path.

Although DVCs provide most of the advantages and overcome the difficulties of static virtual circuits, they do not guarantee the physical FIFO packet arrival at the destination node as with static virtual circuits. For example, a circuit that has been torn down from an intermediate node may be reestablished on a different path before the nonterminal CDP reaches the destination. In this case, the packets on the reestablished branch of the circuit arrive at the destination before all the packets on the torn-down branch arrive. FIFO ordering can be maintained in a DVC environment with the use of local timestamps that uniquely identify at each node all the circuit disestablishment events it has initiated. The timestamp values are placed in tearing CDPs and reestablishing CEPs, and the destination node waits for matching values to arrive to decide ordering. Further details of the DVC mechanism can be found in [12].

III. Deadlock Resolution in Centrally Buffered Packet Switching Networks

In general, packet networks are susceptible to *deadlock* situations when no packets can advance towards their destinations due to full buffers in one or more cycles of communication switches [10]. With a global view of the interconnection topology, it is possible to guarantee deadlock-free routing by

restricting the routing algorithm or the use of the buffers at each node [4, 5, 8]. When deadlocks do not occur, deadlock-free routing results in inefficient use of system resources. An alternative approach to dealing with deadlocks is to detect and resolve them when they occur [6, 3]. With this approach there is no restriction on the routing and all network resources are used for improved performance rather than reserved for eliminating deadlocks. Since Dynamic Virtual Circuits are designed to support distributed adaptive routing in arbitrary topologies [11], the choice of the latter approach to dealing with deadlocks is attractive.

The algorithm proposed in[6], detects and corrects deadlocks in packet switching networks whose nodes have the following properties: they use central queueing (as opposed to input or output queueing); their links are bidirectional; their inputs and outputs all may operate simultaneously; and their routing algorithm leads packets eventually to their destinations for consumption. It is shown in later sections of this paper how this basic algorithm can be modified for use in a network with a different switch model and with Dynamic Virtual Circuit switching.

In the algorithm due to Jaffe and Sidi, when a central buffer becomes full and remains so for some time without transmitting any packets, it is assumed that a deadlock may exist. When a node decides it *may* be in a deadlock, it first attempts to determine if there is a cycle of nodes with full buffers which may form a *potential* deadlock. That is, all actual deadlock cycles are found by the algorithm, but the algorithm may find other cycles as well, if the buffers along those cycles are advancing very slowly. A cycle is found by a distributed technique that requires each blocked node to query a neighbor to which it has packets. The neighbor either replies that there is no cycle, which terminates the algorithm, or it replies with a number which is the maximum of its unique node id and any number it has received in the current search for a cycle. If there is a cycle, eventually the node in the cycle that has the higher node id will receive its own id number from the neighbor that it queried when it first joined the search for a cycle. Such a node that receives its own id declares itself the leader of the cycle, and the leader then sends a notification around the cycle to commit each member to perform rotation. Any cycle member that has discovered in the meantime that it is not part of a deadlock may, until it commits to rotation, cause all the nodes in the cycle to cancel rotation.

If a cycle is found, one packet in each buffer in the cycle is rotated one hop along the cycle so that they move towards their destinations according to the routing algorithm. That completes one iteration of the algorithm. Eventually, perhaps after several iterations, this process may lead to one of the buffers becoming not-full, thus resolving the deadlock. In the worst case, the deadlock persists, because new packets are injected into the cycle from outside it, but even in this worst case all packets are guaranteed to be delivered eventually to their destinations. During cycle detection, nodes that are blocked because their buffers are full are not allowed to inject new packets into the network. To allow nodes to distinguish between distinct iterations of the deadlock detection and resolution algorithm, each node maintains a sequence number and stamps outgoing requests and replies with that sequence number, and queues up requests that arrived stamped with higher sequence numbers than that of the node.

The cycle detection and deadlock resolution algorithms require, at each switch, a fixed amount of storage (equal to the size of one packet) that is available even when the normal buffer is full. Further details of the algorithm can be found in [6].

IV. Deadlock Resolution in Input Buffered Packet Switching Networks

This section considers in detail the task of modifying the deadlock resolution algorithm of section III for use in a pure packet switching system whose switches use input buffers instead of the central buffering assumed by the original algorithm [6]. The difference in buffer structure between the two switch models is accommodated by deriving from the physical network a logically equivalent virtual network that does use central buffering.

The basic idea of the mapping is to associate each node in the virtual network with a distinct input buffer in the physical network. Then, the edges of the virtual network are defined to be those edges that

are necessary to allow packets to traverse virtual nodes in the virtual networks in the same manner that packets traverse the matching input buffers in the physical network. Operating the algorithm on the physical network as though it were being run on the equivalent virtual network guarantees the correctness of the algorithm; the equivalent packet traversal property guarantees that the physical network is deadlocked if and only if the corresponding virtual network is deadlocked.

The physical network can be described as an undirected graph $G = (V, E)$. Here $V = \{v_i\}$ is the set of nodes, and E is the set of undirected edges, where the edge between nodes v_i and v_j is the set $\{v_i, v_j\} \in E$. For each node in the physical network, there is one input port buffer of capacity C for each edge incident to the node.

Physical network G is mapped to a virtual network, which can be described as a directed graph $G'=(V', E')$. V' is the set of virtual nodes, and E' is the set of virtual directed edges, which are ordered pairs of virtual nodes. Here, there is one shared buffer of capacity C per virtual node.

Virtual network G' is constructed from physical network G as follows. There is one virtual node in the set V' for each edge endpoint in the physical network G. We therefore denote each virtual node in V' as an ordered pair (v_i, v_j) to indicate that the virtual node can be mapped to the endpoint at physical node v_j of the physical edge $\{v_i, v_j\} \in E$. Each edge in the set E' is denoted by an ordered pair of virtual nodes in V', indicating a directed edge from the first virtual node to the second virtual node. The edges of E' are chosen to match the connectivity of the physical network. Specifically, $E' = \{((v_i, v_k), (v_j, v_i)) \mid (v_i, v_k), (v_j, v_i) \in V'\}$. The edge $((v_i, v_k), (v_j, v_i))$ has initial endpoint (v_i, v_k) and terminal endpoint (v_j, v_i).

Suppose a packet in the physical network G resides in physical node v_i at the input port that is fed by an edge connecting v_i and neighbor node v_k. In the physical network, the packet may then in one hop move to physical node v_j by traversing the undirected physical edge $\{v_i, v_j\}$. The packet would end up buffered at the input port of v_j that is fed by physical node v_i.

In the corresponding virtual network G', the analogous situation is that the packet initially resides in the shared buffer of virtual node (v_i, v_k). Then, in the virtual network, the packet may in one hop move from virtual node (v_i, v_k) to virtual node (v_j, v_i) by traversing the edge $((v_i, v_k), (v_j, v_i))$ in E'.

This construction of G' ensures that every step taken by a packet in the physical network (virtual network) corresponds to exactly one step taken by the packet in the virtual network (physical network). Therefore, if the packets are in a deadlock configuration in one graph, then they are also in a deadlock configuration in the other. Furthermore, each step taken by packets when resolving the deadlock in one network corresponds to exactly one step in the other.

For a simple example of constructing the virtual network from a physical network, consider the partial physical network shown in Figure 1. In this physical network, some of the input ports are labelled with capital letters. Consider input port C residing at physical switch S4. This input port can receive only packets that are sent from switch S3. Furthermore, input ports A and B are the only ports in S3 that can buffer packets to be sent through S3's crossbar switch to port C. Once a packet is buffered at port C, it may be sent through switch S4's crossbar switch to either switch S5 or switch S6. If sent to S5, the packet is buffered at port D. If sent to S6, the packet is buffered at port E. Part of the virtual network corresponding to the physical network of Figure 1 is shown in Figure 2. Here, each virtual node is labelled with the same capital letter used to identify the corresponding physical input port in Figure 1. All the inputs and outputs of virtual node C are shown, along with the fan-in and fan-out of virtual nodes A, B, D, and E. Virtual nodes A and B can send packets to virtual node C, which in turn can send to virtual nodes D and E, which is the same behavior as the corresponding physical input ports. Note that virtual node A can send to the virtual node corresponding to a physical input port in switch S2, and similarly virtual node B can send to one corresponding to an input port in S1. They key point is that since the virtual network consists of nodes that use central buffering, it can be used for deadlock resolution as long as it conforms in all other ways to the original algorithm's system model.

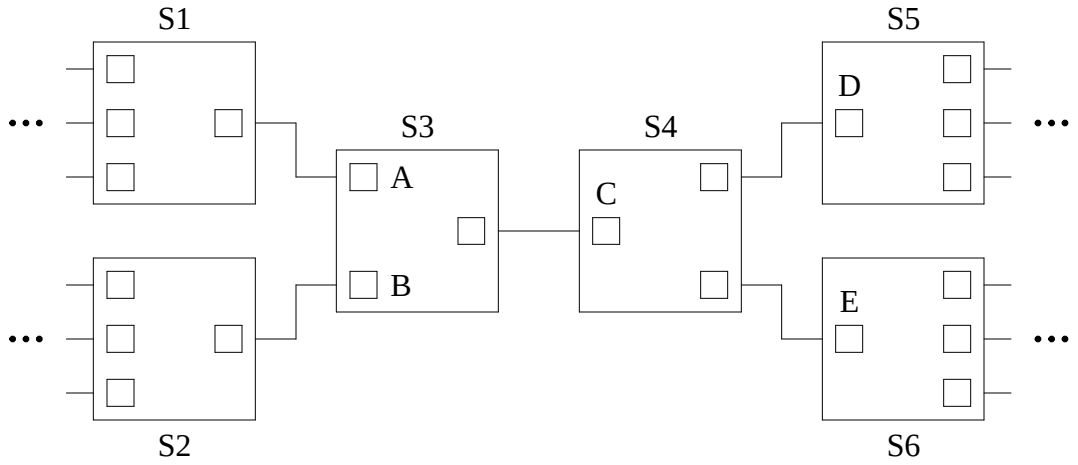

Figure 1: Partial physical network.

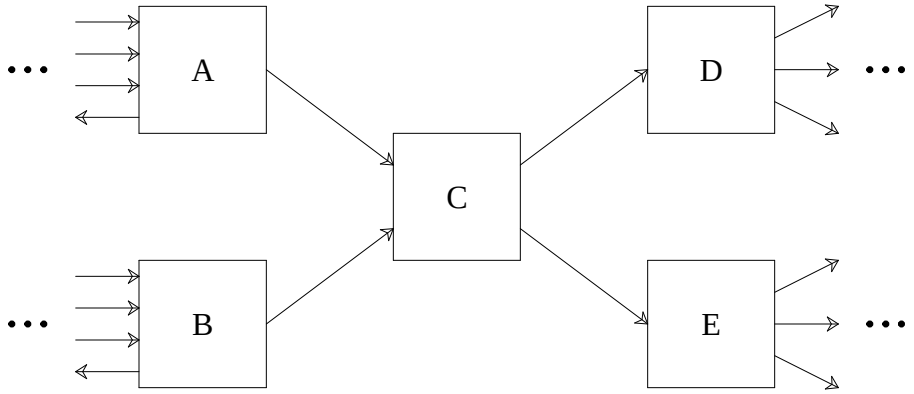

Figure 2: Partial virtual network.

The only difference between the two models is that the deadlock resolution algorithm requires control messages to be able to travel in either direction over a link, but the virtual edges in the virtual network are unidirectional. Additional virtual edges are added to the virtual network for the exclusive use of those control messages that travel opposite to the direction of packet flow.

The transformation of the original resolution algorithm to run as though it were running on the virtual network instead of directly on the physical network is straightforward. The original deadlock resolution algorithm uses sequence numbers on each control message to keep different iterations of the algorithm separate. Each node maintains a sequence number that indicates its count of which iteration it is performing. With input buffering, it is necessary to maintain multiple independent sequence numbers, one for each virtual node mapped to the physical node. Also, it is necessary to send along with each control message enough information to identify for the receiver which virtual node created the message. This is simply an input port number in the physical network. In the case of control messages that traverse the reverse direction of a link in the virtual network, the destination virtual node must be identified as well, since the destination could correspond to any of the input ports at the destination physical node. Whether this information is present depends on the *type* of the control message, since all messages of a single type travel in the same direction along virtual network links. Finally, the set of state variables stored at each node in the original algorithm must be allocated for all the virtual nodes at every physical node.

This section dealt with one fundamental difference between Dynamic Virtual Circuit networks and

the network model assumed by the original deadlock resolution algorithm. It was shown that use of the virtual network model enables the physical network to be thought of as structurally equivalent to the network model of the original deadlock resolution algorithm. By running the algorithm on each virtual node instead of on each physical node, the model differences are reconciled. This would be enough if the system used packet switching. But in a Dynamic Virtual Circuit network, there are additional issues to consider. These are discussed further in section V.C, which discusses cycle rotation with Dynamic Virtual Circuits.

V. Deadlock Resolution With Dynamic Virtual Circuits

The use of Dynamic Virtual Circuits instead of pure packet switching introduces a new category of buffer dependencies that give rise to deadlocks that do not correspond to cycles in the virtual network graph and therefore cannot be handled by simple cycle detection and rotation. This section discusses this new class of dependencies and how to deal with them.

In a pure packet switching system, a packet can be blocked because other packets are using its desired output port or because the neighboring buffer is full and therefore cannot accept new packets. But in a DVC system, a packet can also be blocked because it is unmapped and requires a virtual channel to be allocated to it before it can proceed. Allocating a virtual channel may first require another virtual circuit at the same switch but from a different input port to be torn down, in which case there is a new dependency from one physical input port of the physical switch, namely that storing the unmapped packet, to a second physical input port at the same physical switch, namely that storing the Circuit Destruction Packet (CDP) introduced to tear the victim circuit. Since there is no virtual edge between the virtual nodes corresponding to the two physical input ports at the same physical switch, the new dependency maps to a dependency in the virtual network between virtual nodes that are not neighbors.

The following subsections respectively show how, in a DVC environment, to avoid deadlocks when the underlying routing algorithm lacks cyclic dependencies, to detect deadlocks, and to resolve deadlocks.

A. Deadlock Avoidance in Networks with Deadlock-Free Routing and Dynamic Virtual Circuits

It is desirable for the system to be free of deadlocks if the underlying routing algorithm does not exhibit cyclic buffer dependencies. But with Dynamic Virtual Circuits, the new dependencies introduced by unmapped packets create the danger of deadlocks even with deadlock-free routing policies such as dimension-order routing in a mesh. What is required to completely prevent deadlocks in such an environment is to guarantee that the new dependencies cannot persist indefinitely. This is accomplished by guaranteeing that every CDP created to tear a victim circuit for an unmapped packet will always be sent eventually so long as its desired output port is free.

For a specific example of the problem, consider the case of a DVC switch with FIFO input port buffers of any length, even infinite. The situation is depicted in figure 3. In this figure, two unmapped packets, U_1 and U_2 are waiting in Auxiliary Buffers for channels to be allocated to them. The Auxiliary Buffer is a buffer able to hold at most one packet, and it is used to hold unmapped packets until valid circuits can be established for them. When the Auxiliary Buffer is occupied, other packets are prevented from arriving at the input port from the physical neighbor node. Once unmapped packets are given valid circuits, they can be enqueued in the input port FIFO buffer. CDP_1 is created to free the channel for U_1. U_1 cannot proceed until CDP_1 proceeds; this is the new dependency introduced by Dynamic Virtual Circuits.

CDP_1 is blocked by P_1 ahead of it in the FIFO buffer. Similarly for U_2, CDP_2, and P_2 at the neighbor switch. P_1 and P_2 are blocked by U_2 and U_1, respectively. Because of the head of line blocking of the FIFO buffers, no packet shown in the figure can advance, and there is a deadlock in the physical network that does not correspond to a cycle in the virtual network.

One way to avoid this problem is to ensure that the new dependency never arises; here, a mechanism requiring hardware support would ensure that the circuit chosen for destruction is not one

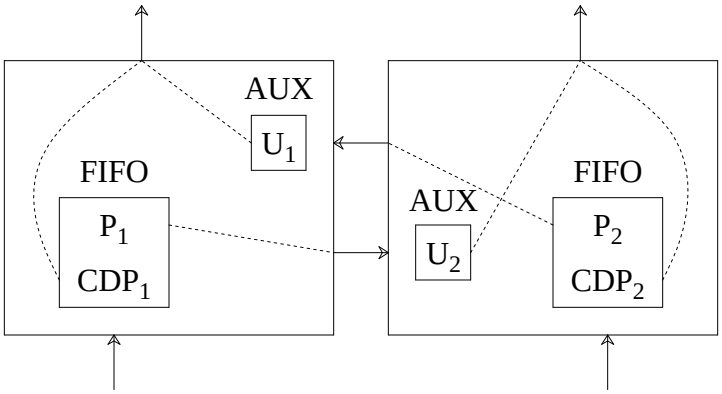

Figure 3: Deadlock with FIFO buffers

being used by any packets currently in the switch. That removes the requirement that the CDP created to tear the circuit down needs to be placed at the end of the queue. Rather, it could simply be forwarded directly out the appropriate output port without waiting for other packets. Therefore, although the unmapped packet still has to wait for the CDP, the CDP cannot be blocked by anything other than the unmapped packet's desired output port. Therefore, there is no new dependency between non-neighbor nodes in the virtual network graph.

A second approach is to allow the new dependencies to exist, but to guarantee that they will disappear eventually under deadlock-free routing. Toward this end, FIFO buffers are replaced by Dynamically Allocated Multi-Queue Buffers (DAMQ Buffers [13]) that do not exhibit head of line blocking. For example, if the FIFO buffers in the previous example were replaced with finite capacity DAMQ buffers, there would not be a deadlock, because the CDPs could be sent immediately, and the new dependency disappears. Unmapped packets still have to wait for CDPs at other input ports to advance, but that advancement depends solely on the output port being free. This is similar to normal packet switching operation in which packets from different input ports arbitrate for access to the same output port and are often made to wait for their turn to transmit. However, there is still an issue of what happens when the DAMQ buffer of the victim circuit is full of packets when the CDP is created. Then, there are two cases.

In the first case, the DAMQ buffer has no packets for the CDP's desired output port, and thus there are no packets there that belong to the victim circuit. In this case, it is safe to send the CDPs ahead of all packets in the DAMQ buffers, completely avoiding a new dependency.

In the second case, the DAMQ buffer does have packets for the same output port desired by the CDP, and therefore the CDP should be sent only after all those packets since some may be using the victim circuit. In this case, the CDP can be buffered off-chip in fixed-sized queues of the Routing Processor's private memory that are reserved for extending the storage of the DAMQ buffers. Before a packet is placed in one of the Routing Processor queues, the corresponding input port is blocked from receiving any packets from the neighbor node. Flow from the neighbor node will not resume until after all Routing Processor queues for the input port are empty. For each internal queue of the DAMQ buffer, one Routing Processor queue can be maintained. Whenever the corresponding internal queue is not empty, the packets in the off-chip queue wait to be inserted at the end of the DAMQ buffer when it frees space.

For the special case of deadlock-free routing, it is not necessary to maintain a queue if the CDP can be sent directly out the output port. However, for the general case in which routing can have cyclic dependencies, when the DAMQ buffer is full but its DAMQ buffer's internal queue is empty, then the CDP is enqueued in the off-chip queue and waits to be sent directly out the desired output port. The

Routing Processor explicitly requests access to the output port for these queues. In the worst case, all n input ports have unmapped packets to send. Then, there are at most n CDPs created by the local switch to free up channels, and there are at most n CEPs created to establish or reestablish the mappings for the unmapped packets. Since the only packets that will be stored in the offchip queues (assuming, again, that the routing algorithm lacks cyclic dependencies) are these locally created CDPs and CEPs, the off-chip storage must be sufficient to store 2n packets.

The technique just described eliminates all deadlocks if the routing algorithm does not have cyclic dependencies, but otherwise deadlocks involving DVC dependencies can still occur. The following subsections describe techniques for detecting deadlocks and resolving them by breaking the extra DVC dependencies and then performing cycle rotation.

B. Deadlock Detection Phase

The first step in deadlock detection is to identify buffers whose packets have not made progress recently. The second step is to discover a cycle of blocked buffers whose packets are waiting for one another to advance.

Blocked buffer identification is accomplished by having the switch periodically check whether some buffer in the switch has become blocked since the last checking period. This is accomplished by attaching a status bit to each buffer in the switch.

Periodically, the switch sets the status bit for each buffer that contains a packet. Once that is completed, the switch allows a waiting period to elapse. During the waiting period, whenever a packet is moved from a buffer, the status bit associated with that buffer is reset, indicating that the buffer is not blocked. Once the waiting period elapses, the switch checks the status bits. Any that are still set must belong to buffers whose packets have not been able to move during the waiting interval. The first such buffer found is declared to be ‘‘Blocked.’’

Once a blocked buffer is found, the deadlock detection algorithm is triggered, since a blocked buffer may be caused by a deadlock rather than being caused by ordinary processing or queueing delays. The output port for which the packet in the blocked buffer is waiting is declared to be the next hop along a possible cycle. The algorithm proceeds with cycle detection by sending a TEST control message out that port. Note that the sending of control messages requires a separate flow control mechanism, since often when control messages are sent the receiving input ports are unable to receive normal traffic (perhaps because they are in a deadlock). One way to accommodate the control messages is to use the Auxiliary Buffer to store them during execution of deadlock detection. A node that notices its input port is blocked can cause any packet in the Auxiliary Buffer to be brought into the Routing Processor’s private memory, thereby freeing up space in the Auxiliary Buffer for control messages. As for flow control, two flow control lines could be used on each communication link that allows the input port to signal to the neighbor whether or not data or control messages can be received. Alternatively, flow control lines can be omitted at the expense of some throughput by transmitting flow control information on the data lines as piggybacked information on packets going in the opposite direction. For this to work it must be possible to transmit the flow control information even when the receiving input port is unable to receive packets.

One possible outcome of cycle detection is the failure to locate a cycle, in which case there is no deadlock and the algorithm terminates. The other possible outcome is that a cycle is found, in which case cycle rotation is committed to by all members of the cycle, and rotation can then proceed. This is the subject of the next section.

C. Cycle Rotation: Handling DVC Dependencies

Two approaches to cycle rotation in a DVC environment are presented here. The first approach reserves some specific virtual channels to be used only to break DVC dependencies for deadlock rotation. Packets arriving on those specific virtual channels are handled specially in a relatively slow procedure.

The advantages of this scheme are that it can be implemented mostly in software by a Routing Processor attached to each node in the network that also controls the complex circuit manipulation operations, and the scheme does not require any additional functionality at the receiving host interface over what is already necessary for DVC processing. The second approach is much faster than the first. In this approach, unmapped packets that need to be rotated have some bytes attached to them (growing their length) that provide enough information for the packet to be forwarded via pure packet switching to the destination, while also providing enough sequencing information to allow the endpoint receiving node to properly order the packet within the stream of packets on its virtual circuit. This scheme has the advantages of simplicity and speed but requires extra storage in the input buffers of each node and extra complexity at the receiving host interface because the scheme introduces a new packet type that the interface must process.

In the first approach, each switch maintains a bank of n normally free channels at each of the n output ports to ensure that in deadlock rotation, each unmapped packet that needs to be rotated can allocate a channel from the free channel bank and therefore does not have to wait for circuit teardown to get a valid mapping. This ensures that any detected cycle can always be rotated, because each packet in the cycle is mapped. The key point is that this approach simply breaks the DVC dependencies by directing unmapped packets to use the bank of free channels. The following discussion presents this n-free-channels technique in more detail.

At each output port, the free channel bank of n normally free virtual channels is used only for handling packets that were unmapped during a deadlock cycle rotation. Each of the n input ports has exclusive use of exactly one of the n virtual channels in the free channel bank of each output port. This ensures that each input port can reuse its virtual channel in the free channel bank at any time. Since each input port sends packets destined for a particular output port in first-come first-served order, there can be no erroneous intermingling of packets that are on different virtual circuits but happen to use the same virtual output channel. There must be n free channels at the output port to satisfy the worst case, in which in rapid succession each input port at a switch is a member of a different deadlock cycle, and all the deadlock cycles use the same output port. Only the input port from the source node can be ignored, since no deadlock cycle can involve it. Once a deadlock cycle is detected, each unmapped packet can be rotated by using one of the n free channels at the output port. The details of cycle rotation are provided below. The terms ‘‘virtual node’’ and ‘‘input port’’ are synonymous in this discussion.

When a virtual node detects that it is the leader of a deadlock cycle, it executes the procedure in figure 4 to prepare to perform rotation.

```
mode := CYCLE_MODE
prohibit packet flow from neighbor into the input port
active := active + 1  /* active > 0 prohibits circuit ops */
if currently unmapped packet is in the Auxiliary Buffer then
    transfer packet to Alternate Auxiliary Buffer
send a CYCLE message to successor node in the deadlock cycle
```

Figure 4: Prepare for rotation

Each other node along the deadlock cycle that receives a CYCLE message executes the same operations, which prepare the node to perform a rotation. Normal flow of packets from the neighbor to the input port is prevented, and circuit manipulation operations that interfere with the state of the input port are disabled until after the cycle rotation. Any packet in the Auxiliary Buffer is transferred to the Alternate Auxiliary Buffer in the Routing Processor’s private memory.

Once the CYCLE message finally traverses the entire ring and reaches the leader node again, all nodes in the cycle have prepared for rotation, and therefore the leader initiates the actual rotation by forwarding a packet.

The packet chosen by the virtual node for forwarding is the one that is first in line to be sent to the successor from the input port. The buffers checked for packets are, in order, the DAMQ buffer, the queue in Routing Processor private memory that extends the DAMQ buffer, and the Alternate Auxiliary Buffer. Finally, if there is no packet, then a ‘‘dummy’’ packet is rotated to the next node, where it is discarded. If the packet found is mapped, then it is handled as shown in figure 5.

```
if packet is mapped then
    rotate it to successor node
else
    allocate channel from free channel bank to the packet
    if packet is a CEP initializing a new circuit then
        record destination id in Circuit Destruction Table
    if packet is not a CEP then
        create a CEP to establish its mapping on the free channel
    if packet is not a CDP then
        create a CDP to release its mapping after it is sent
    timestamp := timestamp + 1
    rotate CEP if it was created
    rotate the packet
    rotate the CDP if it was created
```

Figure 5: Rotating

Rotating an unmapped packet requires that the packet first is given the correct output channel in the free channel bank on the output port. A CEP is created to allocate the channel for the packet, and a CDP is allocated to free, unless the packet is itself a CEP or CDP. If the packet is a CEP, then only the CDP is created; if the packet is a CDP, then only the CDP is created. The created CEP and CDP each contain the current node id and timestamp that together allows the ultimate receiving node to properly sequence packets that belong to the same flow. The timestamp value is incremented after each circuit destruction event and after each conversion of an unmapped packet to a sequence of packets that use a channel in a free channel bank. Incrementing the timestamp value in this manner ensures that these are uniquely identifiable events; a CEP and CDP with matching teardown id and timestamp values must be consecutive packets in the circuit stream, even if they do not arrive consecutively or even in the logically correct order [12]. The resulting group of packets is then forwarded.

When a rotated packet arrives, the Routing Processor is interrupted. If the receiving virtual node is not the leader of the cycle, then it rotates a packet in the manner described above. Only after rotating its own packet is the packet that arrived handled. Rotating a packet before handling the arriving packet ensures that all nodes along the deadlock cycle receive rotated packets as quickly as possible. The handling of the arriving rotated packet P is shown in figure 6. In addition, during normal operation, whenever a packet is received on a channel from the free channel bank, the Routing Processor is interrupted and the packet is placed in the Routing Processor private memory allocated for that virtual channel. Once a CDP on that channel arrives, all the packets stored for that virtual channel are routed to an output port, allocated the appropriate channel in the output port’s free channel bank, and then enqueued. Keeping the packets together in this manner while they traverse the network ensures that the channels in the free channel bank are always available for use in deadlock rotation.

On each rotation a node undergoes, a group of up to three new packets enter the node and a group of up to three packets leaves. The total number of such packet groups that are present in the node cannot increase in a series of rotations. At worst, each rotation results in no change in the total number of packet groups present, and on some rotations the number of groups may go down if a dummy packet is rotated in. Since the number of packet groups present in the switch before deadlock first occurs is bounded, and on each rotation the number of packet groups does not increase, the storage requirements for this

```
if P is a dummy packet then
    discard P
else if P uses a channel from the free channel bank then
    transfer P to Routing Processor private memory
    while P is not a CDP do
        wait for a new packet P to rotate in
        transfer new P to Routing Processor private memory
    determine output port OP for the saved packets
    map saved packets to channel in OP's free channel bank
    enqueue packets in DAMQ buffer or in DAMQ buffer extension
else
    if P is mapped or there is a CEP waiting to map P then
        if P and the Alternate Auxiliary Buffer packet AAB are
        of the same virtual circuit then

            enqueue AAB packet in DAMQ buffer or extension

        enqueue P in DAMQ buffer or extension
    else
        if P and the AAB packet are of the same virtual circuit then
            allocate channel from the free channel bank for AAB
            create CEP and/or CDP for AAB
            enqueue CEP, AAB, and CDP in DAMQ buffer or extension
        allocate channel from the free channel bank for P
        create CEP and/or CDP for P
        enqueue CEP, P, and CDP in DAMQ buffer or extension
active := active - 1  /* possibly enables circuit operations */
```

Figure 6: Handling arriving packet

mechanism are finite and equal to the capacity needed for normal operation plus the extra capacity needed to hold a CEP and a CDP for each packet when the storage for normal operation is depleted.

An alternative to the n-free-channels technique is to introduce a new packet type that is routed using pure packet switching. An unmapped packet that needs to be rotated is first converted into this packet type by adding a few bytes to the header. The new bytes identify the packet as being of the new type and also contain the same information present in a CEP created in the n-free-channels technique (namely, destination id, teardown node id, timestamp). The Auxiliary Buffer needs to be made large enough to hold the maximum sized packet that is modified in this manner. In addition, in normal operation the flow control signalling must not allow a packet to be sent to the neighbor until that neighbor's DAMQ buffer has sufficient space to store a maximum sized packet that has been converted to this new type. With this technique, there is no need for two banks of virtual channels, and there is no need to create new packets to handle unmapped channels. Handling a rotated packet P is shown in figure 7.

VI. Performance

To investigate the behavior of deadlock detection and resolution, a simulation testbed was developed that provides cycle-level emulation of the algorithm described in this paper. The simulator perfectly models the details of packet forwarding and circuit manipulation when deadlocks do not occur, but it makes the simplifying assumption that control messages sent by the nodes executing the deadlock detection and resolution algorithm are assumed not to interfere with ordinary traffic. Each control message traverses a physical link in a fixed 16 clock cycles. This assumption is justified since control

```
if P is a dummy packet then
    discard P
else if P is of the new packet type then
    determine output port needed by P
    enqueue P in DAMQ buffer or extension
else
    if P is mapped or there is a CEP waiting to map P then
        if P and the Alternate Auxiliary Buffer packet AAB are
        of the same virtual circuit then

            enqueue AAB in DAMQ buffer or extension

        enqueue P in DAMQ buffer or in DAMQ buffer extension
    else
        if P and AAB are of the same virtual circuit then
            convert AAB to the new packet type
            enqueue AAB in DAMQ buffer or extension
        convert P to the new packet type
        enqueue P in DAMQ buffer or DAMQ buffer extension
active := active - 1  /* possibly enables circuit operations */
```

Figure 7: Handling rotated packet

messages in a reasonable system are at most a small fraction of the overall traffic, and furthermore many control messages are sent over links that are not usable by ordinary packets because of the congestion that triggers the deadlock detection algorithm. For the latter control messages, the simplifying assumption made by the simulator has no effect on the measured performance.

The particular network simulated is a two dimensional square mesh of 36 nodes. Extensive simulation revealed that with nonadaptive routing policies that exhibit many dependency cycles, deadlocks are extremely rare under light load, but under heavier load deadlocks are plentiful. Under heavy load, the deadlock resolution algorithm is unable to resolve deadlocks faster than new packets are injected into the network to cause new deadlocks. Therefore, once the first deadlock occurs, and cycle rotation begins, the network remains forever in ‘‘deadlock mode,’’ continually rotating packets around detected deadlock cycles. Since deadlock detection and rotation is relatively slow compared to normal operation, the resulting network latency is high and throughput is very low. The conclusion of these investigations is that for deadlock resolution to be effective, the traffic conditions that give rise to deadlocks must be infrequent.

To further investigate the behavior of this deadlock resolution technique, a traffic pattern was devised to cause deadlocks only occasionally. In these experiments, the routing algorithm is set so that all destination nodes except two are reached using row-first routing. The two remaining nodes are routed to using column-first routing. The two remaining nodes are at opposite corners of the mesh (namely, one node at row 2 column 5, and one at row 5 column 2). Most of the time, the destination distribution is uniform, and the load is light (using only 10% of the mesh’s bisection bandwidth). However, periodically, a burst of traffic lasting for 10000 clock cycles at high load (60% of bisection bandwidth) and a non-uniform destination distribution occurs. The destination distribution is chosen so as to make deadlocks very likely. Namely, the two nodes at the opposite corners that are routed to column-first send all their injected packets to each other (column-first), and the nodes at the other two corners send all of their injected packets to each other. The result, as shown in Figure 8 is a cyclic traffic pattern that makes deadlock very likely during the burst. Note that since all nodes route to the two corner nodes column-first, other deadlocks can occur in this system as well.

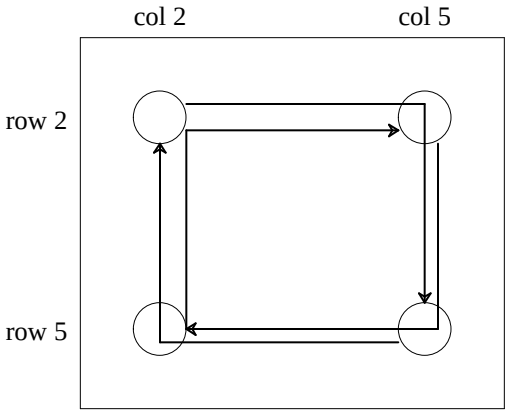

Figure 8: Mesh with cyclic routing pattern

Once the burst of heavy deadlock-prone traffic is over, deadlocks persist in the network until the deadlock resolution algorithm can clear the network of enough packets that normal operation can proceed. Figure 9 shows the time required after the burst period ends to completely recover from the effects of the burst versus the number of clock cycles between subsequent checks by each routing processor for input ports that are not making progress.

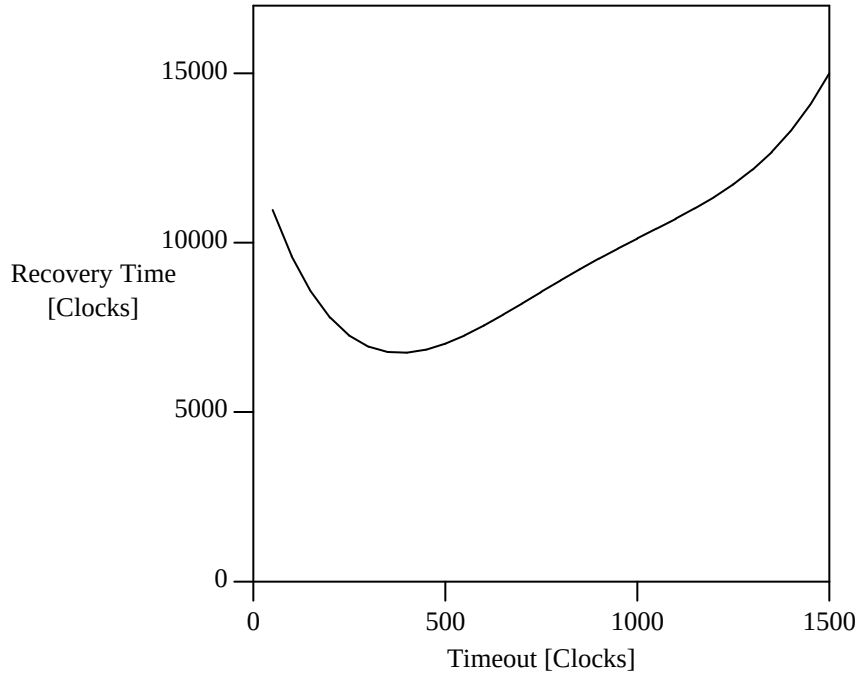

Figure 9: Recovery time versus timeout period

The graph shows that for both very large and very small timeout periods, recovery time is long. Recovery time is long when timeouts are long because deadlocks can persist for a long time before the detection algorithm is triggered by a timeout. Therefore, packets are blocked in deadlock for a long time without the system making any attempt to force them to advance through cycle rotation. Understanding why small timeout values cause long recovery times requires more explanation.

Figure 10 shows the average number of times over ten runs of ten bursts each that nodes declare

themselves blocked and later return to an unblocked state without first participating in a cycle rotation. These nodes declare they are blocked, participate in the deadlock detection algorithm, and then find that they are not in a deadlock because they are able to send a packet before identifying a deadlock cycle.

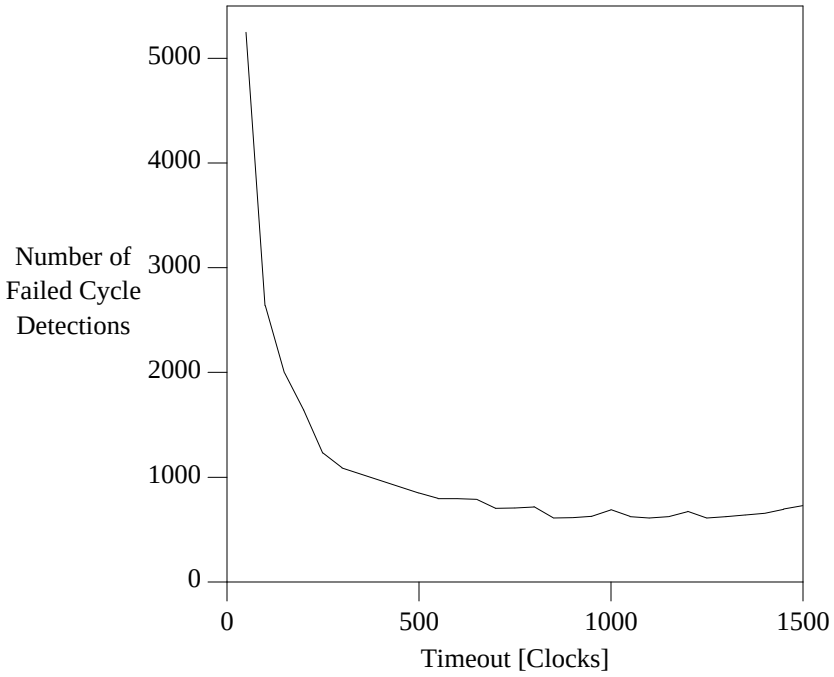

Figure 10: Number of failed cycle detection attempts versus timeout

The frequency of such failed detection attempts has an effect on the time needed to discover deadlocks because of its effects on the sequence numbers used by the detection and resolution algorithm. The algorithm relies on the use of sequence numbers to distinguish between different iterations. This prevents deadlock cycles from being missed due to the arrival of obsolete information [6]. This means that for a deadlock cycle to be detected, all the nodes in the deadlock cycle must agree on the current sequence number. As the graph shows, when the timeout value is very low, nodes time out very frequently without detecting a deadlock. This happens because transient congestion that does not constitute actual deadlocks are likely to cause timeouts when the timeout value is small. Each timeout causes the node that timed out to increment its local sequence number by one. Because nodes are timing out at random times and incrementing their sequence numbers at different rates, once a deadlock happens, the nodes in the deadlock cycle can have significantly different sequence numbers and therefore require a time-consuming synchronization. When the timeout period is long, many fewer failed deadlock cycle detections occur. Those that do occur are mostly at nodes that are in deadlock but are not in the deadlock cycles at the ‘‘core’’ of the deadlocks. The nodes must wait for the deadlock cycles to be rotated before they can advance. With larger timeouts, since failed attempts are rare, the sequence numbers of the nodes in actual deadlocks are relatively close together when deadlock detection begins, resulting in lower synchronization costs.

VII. Summary and Conclusions

Schemes that prevent deadlocks in buffered packet networks restrict the use either of buffer resources or of network routes, whereas deadlock detection and recovery enables flexible resource usage. Based on a scheme for recovering from deadlocks in centrally-buffered packet switching networks, we have presented presented a technique for deadlock detection and resolution in an input buffered network which uses Dynamic Virtual Circuits. Input buffers are accommodated by mapping the physical network

to an equivalent virtual network that uses central buffering.

Dynamic Virtual Circuits support efficient adaptive routing in networks of arbitrary topology. However, the DVC mechanism introduces a new class of buffer dependencies not present with conventional virtual cut-through packet switching. These additional buffer dependencies have the potential of causing deadlocks. We have shown how to design the mechanism such that when the underlying packet routing algorithm does not exhibit cyclic dependencies, the new dependencies do not cause deadlocks. Furthermore, with general routing, modifications to the deadlock resolution algorithm and packet handling were presented that remove the new dependencies that would prevent cycle rotation.

We investigated the performance of deadlock resolution. As expected, networks tend not to deadlock under light load even when there is the potential for deadlocks. However, under constant heavy load with cyclic routing dependencies, deadlocks can be frequent, deadlock resolution cannot remove packets from deadlock cycles faster than they are replenished by the introduction of new packets, and the resulting performance is poor. If the traffic injected into the network gives rise to deadlocks only occasionally, recovery occurs and normal operation can proceed. In an environment of only occasional deadlocks, we investigated the effect on recovery time of varying how long a node is blocked before it initiates detection. Both very small and very large timeouts were found to cause the longest recovery times, because of synchronization overhead with small timeouts and long periods of no activity with large timeouts.

References

1. D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall (1987).
2. E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, ‘‘Hyperswitch Network for the Hypercube Computer,’’ *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 90-99 (May 1988).
3. I. Cidon, J. M. Jaffe, and M. Sidi, ‘‘Local Distributed Deadlock Detection by Cycle Detection and Clustering,’’ *IEEE Transactions on Software Engineering* **SE-13**(1), pp. 3-14 (January 1987).
4. W. J. Dally and C. L. Seitz, ‘‘Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,’’ *IEEE Transactions on Computers* **C-36**(5), pp. 547-553 (May 1987).
5. S. A. Felperin, L. Gravano, G. D. Pifarré, and J. L. C. Sanz, ‘‘Routing Techniques for Massively Parallel Communication,’’ *Proceedings of the IEEE* **79**(4), pp. 488-503 (April 1991).
6. J. M. Jaffe and M. Sidi, ‘‘Distributed Deadlock Resolution in Store-and-Forward Networks,’’ *Algorithmica* **4**(3), pp. 417-436 (1989).
7. P. Kermani and L. Kleinrock, ‘‘Virtual Cut Through: A New Computer Communication Switching Technique,’’ *Computer Networks* **3**(4), pp. 267-286 (September 1979).
8. J. V. Leeuwen and R. B. Tan, ‘‘Interval Routing,’’ *The Computer Journal* **30**(4), pp. 298-307 (August 1987).
9. J. Y. Ngai, ‘‘A Framework for Adaptive Routing in Multicomputer Networks,’’ Computer Science Technical Report 89-09, California Institute of Technology, Pasadena, CA (May 1989).
10. D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, The MIT Press (1987).
11. W. D. Tajibnapis, ‘‘A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network,’’ *Communications of the ACM* **20**(7), pp. 477-485 (July 1977).

12. Y. Tamir and Y. F. Turner, ‘‘High-Performance Adaptive Routing in Multicomputers Using Dynamic Virtual Circuits,’’ *6th Distributed Memory Computing Conference*, Portland, OR, pp. 404-411 (April 1991).

13. Y. Tamir and G. L. Frazier, ‘‘Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches,’’ *IEEE Transactions on Computers* **41**(6), pp. 725-737 (June 1992).