A Universal Proof Technique for Deadlock-Free Routing in Interconnection Networks

Loren Schwiebert and D. N. Jayasimha*
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210–1277
{loren,jayasim}@cis.ohio-state.edu

Abstract

An important open problem in interconnection network routing has been to characterize the conditions under which routing algorithms are deadlock-free. Although this problem has been resolved for restricted classes of routing algorithms, no general solution has been found. In this paper, we solve this problem by proving a necessary and sufficient condition that can be used for any interconnection network routing algorithm, as long as only local information is required for routing. Our proof technique is universal: it can be used with any switching technique that is not inherently deadlock-free. This includes switching techniques such as wormhole routing, store-and-forward routing, and virtual cut-through. The proof technique for the necessary and sufficient condition introduces a new type of dependency graph, the *buffer waiting graph*, which omits most dependencies that cannot be used to create a deadlock configuration. Our methodology is illustrated by proving deadlock freedom for a store-and-forward routing algorithm for meshes and a wormhole routing algorithm for hypercubes. The hypercube routing algorithm requires only two virtual channels per physical channel and is more adaptive than any previously proposed wormhole routing algorithm for hypercubes.

1 Introduction

Large-scale multiprocessors use an interconnection network to support communication and synchronization among processors. A variety of switching techniques have been proposed for transmitting data through the interconnection network, including circuit switching, store-and-forward routing, virtual cut-through, and wormhole routing.

Circuit switching sends a probe from the source to the destination and returns an acknowledgment once the path is established. After the acknowledgment is received, multiple messages can be transmitted on this path before the path is released. *Store-and-forward* routing is implemented via packet switching or message switching. Packet switching first packetizes each message and then sends each packet individually, whereas message switching sends the entire message at once. The network treats each packet as a separate message, so for our purposes there is no need to distinguish between packet switching and message switching. Packet switching transmits an entire packet immediately to a neighboring node. The entire packet is buffered at this node and then forwarded to the next node in the path. Kermani and Kleinrock [20] proposed *virtual cut-through* to combine the best features of packet switching and circuit switching. Virtual cut-through works like packet switching with one important difference. When the packet header arrives at an intermediate node and the next buffer in the path is available, the packet is forwarded immediately to the next node. The packet is buffered only when the next buffer is not available. Adaptive cut-through [24] is a modification of virtual cut-through, where all blocked packets are misrouted instead of buffered. Dally and Seitz [8] proposed *wormhole routing* to avoid the large buffers required for virtual cut-through and store-and-forward routing. Wormhole routing works like virtual cut-through except instead of buffering the entire packet when the next buffer is unavailable, the packet remains in the network. Small buffers are associated with each channel (link) in the network. The buffers are large enough to hold only a few bytes (flits) of the packet. Hence, a blocked packet can reserve many channels while waiting at an intermediate node. In order to reduce the effects of blocking, Dally [6] proposed the use of multiple *virtual channels* on each link. Each virtual channel has a separate buffer, with multiple packets multiplexed over the same link. See Ni and McKinley [25] for an in-depth discussion of wormhole routing.

The simplest routing algorithms are *nonadaptive* and define a single path between the source and destination. *Adaptive* routing algorithms, on the other hand, support multiple

* Part of this work was done when the author was on leave at NASA Lewis Research Center, Cleveland, Ohio.

paths between the source and destination. Latency and contention can be reduced by using these multiple paths. Routing algorithms are either minimal or nonminimal. Minimal routing allows only shortest paths to be chosen, while nonminimal routing does not require packets to use only shortest paths. Gaughan and Yalamanchili [13] present a comprehensive overview of adaptive routing protocols. Whether minimal or nonminimal, adaptive routing algorithms can be further differentiated by the fraction of shortest paths they allow. *Partially adaptive* routing algorithms do not allow *all* packets to use *any* shortest path. *Fully adaptive* routing algorithms *do* allow all packets to use any shortest path. Adaptive routing is typically implemented using additional buffers or virtual channels.

Router latency and cycle time increase with the number of virtual channels [3], so fewer virtual channels are generally better. Reducing the number of buffers (virtual channels) needed for a given degree of adaptiveness is accomplished by using a less restrictive routing algorithm. Conversely, when the same number of buffers is used, a less restrictive routing algorithm has better performance than a more restrictive routing algorithm [23, 14]. A natural question that arises is: Exactly how restrictive must the routing algorithm be to guarantee deadlock freedom? In other words, what is a necessary and sufficient condition for deadlock-free routing? In this paper, we present a theoretical result for minimizing the restrictions imposed for deadlock-free routing. This proof technique applies to switching techniques that prevent deadlock through routing restrictions. Other methods of preventing deadlock, such as abort/retry, which is used in circuit switching, and misrouting, which is used in adaptive cut-through, are inherently deadlock-free. The only restriction we impose on the routing algorithms is that only local information available at the router is used to make the routing decision. In general, routing is done based solely on local information, because of the overhead of transmitting non-local information and the additional router complexity that is required to utilize this information.

In addition to providing a necessary and sufficient condition for deadlock freedom, we propose a fully adaptive minimal wormhole routing algorithm for hypercubes that is much less restrictive than previous routing algorithms. We also prove deadlock freedom for a store-and-forward mesh routing algorithm to demonstrate the ease with which our proof technique can be applied.

2 Previous Work

We briefly review the existing techniques for proving deadlock freedom. For nonminimal routing, the issue of livelock freedom also arises, however, livelock freedom and deadlock freedom are independent issues [28].

For store-and-forward routing, most routing algorithms have used the proof technique proposed by Günther [19] to prove deadlock freedom. This methodology requires an acyclic ordering of the buffers. Toueg and Steiglitz [30] have shown that this is necessary and sufficient for nonadaptive routing. Recently, Cypher and Gravano [5] have shown that an acyclic ordering of the buffers is not necessary for adaptive routing. Günther's proof technique is also applicable to virtual cut-through, because any deadlock configuration consists of blocked, and thus buffered, packets.

For wormhole routing, many nonadaptive and adaptive routing algorithms have used the proof technique proposed by Dally and Seitz [8], which requires an acyclic ordering of the virtual channels. Dally and Seitz also proved that an acyclic ordering of the channels is a necessary and sufficient condition for deadlock-free nonadaptive wormhole routing algorithms. Glass and Ni [15, 16] and Boura and Das [2] have also proposed methodologies for generating deadlock-free wormhole routing algorithms that require an acyclic ordering of the channels.

Duato [9, 11] proved that requiring an acyclic ordering of the channels is not necessary for adaptive routing algorithms if the output channel is selected independent of the input channel. Schwiebert and Jayasimha [27] have used Duato's proof technique to propose an optimal fully adaptive routing algorithm for meshes. Berman, *et al.* [1] prove deadlock freedom for a torus routing algorithm that has no acyclic ordering of the channels and allows the router to consider the input channel when selecting the output channel. Dally and Aoki [7] prove deadlock freedom for a routing algorithm with cyclic dependencies by guaranteeing an acyclic *packet wait-for graph*. A packet wait-for graph is defined dynamically by the packets in the network and contains an edge from packet p_i to packet p_j if packet p_i is waiting for a channel occupied by packet p_j.

All these proof techniques provide only a sufficient condition for deadlock-free adaptive wormhole routing. Determining what constitutes a necessary and sufficient condition for *adaptive* routing algorithms has remained an open problem. Lin, McKinley, and Ni [22] propose a proof technique based on the fact that a routing algorithm is deadlock-free if none of the channels in the network can be occupied forever. This proof technique was proposed as a necessary and sufficient condition, although Duato points out that only sufficiency is proved [12]. Recently, we have proposed a necessary and sufficient condition for a rich class of *wormhole* routing algorithms [28]. (Duato [12] has independently proposed a necessary and sufficient condition for a restricted class of adaptive wormhole routing algorithms.) In this paper, we propose a necessary and sufficient condition for deadlock freedom that can be applied regardless of the switching technique employed.

3 System Model

Each processing node contains a computation processor and a communication processor. A communication processor contains a finite number of buffers, called standard buffers,

that are used for routing packets between neighboring nodes. Packets are injected into the network by transferring a packet from the computation processor to the injection buffer on the communication processor. Similarly, a packet is delivered by transferring the packet from the delivery buffer on the communication processor to the computation processor. (The injection and delivery buffers are introduced only to simplify the model and may not actually exist in the communication processor.) Neighboring nodes are connected by one or more bidirectional channels (links) between their communication processors. The network topology is defined by the choice of neighboring nodes.

For wormhole routing, each link (physical channel) is partitioned into a set of unidirectional virtual channels. Packets are routed by transferring the data from the input channel to an output channel on the same node and then transferring the data across the physical channel to the input channel of the neighboring node. Each virtual channel can store only a few bytes (flits) of the packet. Since packets typically consist of many flits, packets can span many nodes and simultaneously occupy many virtual channels. For virtual cut-through and store-and-forward routing, each communication processor has one or more standard buffers. Packets are routed by transferring data between standard buffers on neighboring nodes. Virtual cut-through and store-and-forward routing require that each buffer is large enough to store an entire packet. When transferring a packet between two buffers, the packet temporarily occupies both buffers and is removed from the first buffer in a finite amount of time after being transferred to the second buffer.

Several standard assumptions are used.

1. A node can generate messages of arbitrary length destined for any other node at any generation rate. The messages are then divided into fixed-length packets. For wormhole routing, the packets can be large enough to occupy all the virtual channels from the source to the destination.

2. A packet arriving at its destination is eventually consumed.

3. Once a buffer has accepted the packet header, it must accept and transmit the entire packet before accepting data from any other packet.

4. Without loss of generality, the buffers used with store-and-forward routing and virtual cut-through can store only one packet.

5. A router arbitrates among packets that simultaneously request the same buffer. Packets waiting for buffers are chosen in an order that prevents starvation.

For readability, we use only the term buffer in the remainder of the paper, rather than both buffer and virtual channel. In the context of wormhole routing, a buffer refers to the flit buffer associated with a particular virtual channel. When a packet acquires a virtual channel, it actually acquires the flit buffer that constitutes the virtual channel. Thus, there is an implicit one-to-one correspondence between a buffer and a virtual channel. This formulation of the network resources in terms of buffers allows a universal framework for addressing deadlock freedom that is independent of the particular switching technique.

As a consequence of assumptions 3 and 4, two packets can never be stored simultaneously in the same buffer. A packet that arrives at its destination is transferred from the communication processor to the computation processor in a finite amount of time. Similarly, a packet is transferred from an injection buffer or its current standard buffer to an empty standard buffer or a delivery buffer in a finite amount of time. Thus, packets always make progress if possible.

A routing algorithm specifies which buffers can receive a packet from the current buffer. This decision is made based on, in the case of virtual cut-through and store-and-forward routing, the current node and destination node and either the current buffer or the source node. In the case of wormhole routing, this decision is made based on the input channel, the current node, and the destination node. The routing algorithm consists of two parts; a *routing relation*, which defines the set of output buffers to which the packet can be routed, and a *selection function*, which selects a single output buffer based on the status of this set of output buffers.

A routing algorithm is *prefix-closed* if, between any pair of nodes, every partial path through an intermediate node can also be used by the source node to reach this intermediate node. For example, any partial path from n_i to n_j through n_k can also be used to route a packet from n_i to n_k. Similarly, a routing algorithm is *suffix-closed* if, between any pair of nodes, every partial path through some intermediate node can also be used by this intermediate node to reach the destination. A routing algorithm is *coherent* if the routing algorithm is both prefix-closed and suffix-closed.

We define a *waiting buffer* to be a buffer that a packet can wait to acquire when all the output buffers permitted by the routing relation are unavailable. A packet may have multiple waiting buffers. The idea of waiting channels was also introduced independently by Lin, McKinley, and Ni [22] for wormhole routing, however, the methodology used in this paper is novel.

The *buffer waiting graph* (BWG) for a given routing algorithm is a directed graph, $BWG = G(B, E)$. The vertices of the BWG are the buffers and the edges of the BWG are pairs of buffers (q_1, q_2) where a packet that currently occupies q_1 can wait for buffer q_2. For virtual cut-through and store-and-forward routing, (q_1, q_2) are buffers on the same node or neighboring nodes, because blocked packets occupy a single buffer. With virtual cut-through, this occurs after the packet is blocked for a finite amount of time. For wormhole routing, there is no requirement that a packet waits for q_2 *immediately* after using q_1, only that the packet length is sufficient to fill the buffers from q_1 to q_2. (Otherwise, the packet

cannot occupy buffer q_1 while waiting for q_2.)

A routing algorithm is *wait-connected* if a packet always has at least one waiting buffer. A packet that has not reached its destination must wait for an output buffer when it is unable to proceed or the packet is never delivered. Hence, every loss-less routing algorithm in our system model is wait-connected.

A *configuration* is an assignment of packets to buffers. A configuration is *legal* if each buffer in the configuration has at most one packet. In addition, the packets in the configuration must be stored in the buffers in accordance with the type of routing chosen. A *deadlock configuration* for a given routing algorithm is a non-empty legal configuration consisting of a set of packets, $p_1, p_2, \ldots, p_n, n > 1$. Each packet in the set, p_i, is not in a delivery buffer and is unable to proceed because every output buffer for p_i is unavailable. Moreover, every waiting buffer for p_i is occupied by another packet in the set. In the case of virtual cut-through, the entire packet has been stored in a single buffer. In the case of wormhole routing, all the buffers occupied by the packet are full (with the exception of the last buffer, which may be only partially filled), so none of the buffers can be released. Thus, each packet is blocked and must wait for an unavailable waiting buffer occupied by another packet in the set. When $n = 1$, p_1 waits for a buffer it already occupies. The set of packets can be ordered such that:

> p_i waits for a buffer occupied by $p_{i+1} \forall i < n$ and
> p_n waits for a buffer occupied by p_1

4 Necessary and Sufficient Condition

Most techniques for proving deadlock freedom require the existence of an acyclic ordering of the buffer usage. From our definition of a deadlock configuration, however, it is clear that every deadlock configuration is formed with the waiting buffers, rather than the entire set of output buffers that a packet could use. The routing algorithm may allow a packet to use a buffer when the buffer is free, even if the packet is not permitted to wait for this buffer when the buffer is occupied. This is our motivation for using the BWG, since it ignores dependencies that cannot result in deadlock. Requiring an acyclic buffer waiting graph is less restrictive than requiring an acyclic ordering of the buffer usage. It also leads to simpler proofs when proving deadlock freedom.

Theorem 1 *If routing algorithm R_A is wait-connected and the BWG for R_A is acyclic, then R_A is deadlock-free.*

Proof. R_A is wait-connected, so every packet always has a waiting buffer when all output buffers are occupied. Assume there is a deadlock configuration involving n packets. If $n = 1$, then there is an edge in the BWG from a buffer to itself, which is not possible since the BWG is acyclic. Otherwise, $(\forall i < n)$ there is an edge in the BWG from every buffer occupied by p_i to the buffer occupied by p_{i+1} for which p_i is waiting (call this buffer q'_{i+1}). There is also an edge in the BWG from every buffer occupied by p_n to the buffer occupied by p_1 for which p_n is waiting (call this buffer q'_1). Hence, there is an edge in the BWG from q'_i to q'_{i+1} $(\forall i < n)$ and from q'_n to q'_1. However, the BWG for R_A is acyclic, so no such set of edges is possible. Therefore, no deadlock configuration exists and R_A is deadlock-free. □

The BWG reflects only the static dependencies among buffers. Note, however, that only as the buffers are used do the dependencies among the buffers arise. The BWG does not capture the dynamic way in which the dependencies are created. Hence, it is possible that a cycle in the BWG exists, but the cycle could arise only from the simultaneous use of the same buffer by more than one packet. For this reason, we divide cycles in the BWG into two classes: False Resource Cycles and True Cycles. A False Resource Cycle is a cycle in the BWG that *requires* at least one buffer to be used simultaneously by more than one packet in order to create the cycle. Obviously, a False Resource Cycle cannot occur, since this is physically impossible. Therefore, a False Resource Cycle cannot be used to create a deadlock configuration. A True Cycle is a cycle in the BWG that can be created without the simultaneous use of any buffer. In other words, a True Cycle is any cycle in the BWG that is not a False Resource Cycle. In Section 5, we provide a more complete description of False Resource Cycles. A technique for distinguishing between False Resource Cycles and True Cycles can be found in [28].

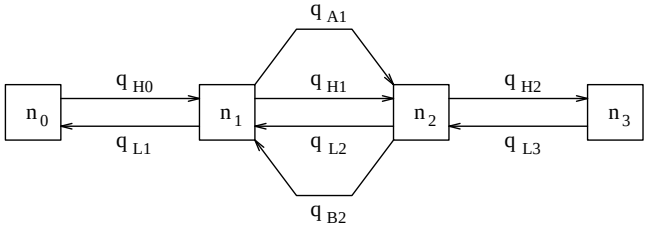

Figure 1: Duato's Example of an Incoherent Routing Algorithm

To illustrate the difference between False Resource Cycles and True Cycles, Duato's example [10] of an incoherent wormhole routing algorithm is presented. The processors and buffers are shown in Figure 1. The buffers are represented by edges that indicate which two processors they connect. The routing algorithm permits only minimal routing, with the exception of buffer q_{B2}. Buffer q_{B2} can be used only by a packet destined for node n_3. Clearly, this routing algorithm is not coherent (not prefix-closed), since a packet from n_2 to n_3 can be routed through n_1 using buffer q_{B2}. However, a packet from n_2 to n_1 cannot use buffer q_{B2}.

Figure 2 depicts the buffer waiting graph for the incoherent routing algorithm. The BWG for this routing algo-

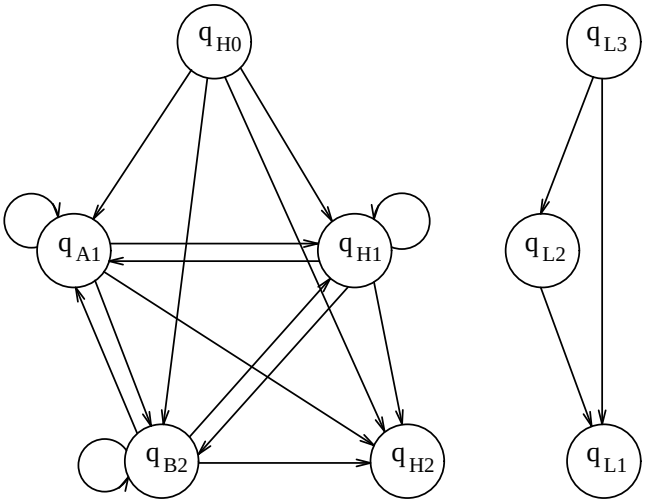

Figure 2: The BWG for the Incoherent Routing Algorithm

rithm has a False Resource Cycle and a True Cycle. A packet whose input buffer is q_{B2} can wait for q_{A1} or q_{H1}. If the packet waits for q_{A1}, there is a True Cycle from q_{A1} to q_{A1} that uses q_{B2}. Otherwise, the True Cycle is a cycle from q_{H1} to q_{H1} that uses q_{B2}. There is a False Resource Cycle that involves two packets. A packet that occupies q_{A1} and q_{B2} and waits for q_{H1} and a packet that occupies q_{H1} and q_{B2} and waits for q_{A1}. Obviously, this False Resource Cycle exists only because both packets simultaneously occupy q_{B2}, which is impossible.

A packet is unable to proceed when all output buffers for the packet are occupied. This situation can be resolved in one of two ways: (1) The packet could wait for a specific output buffer to become free or (2) The packet could wait until *any* permitted output buffer becomes free. For case (1), the routing algorithm must choose a buffer for which waiting is permitted. Although it is possible that the routing algorithm has a choice of more than one waiting buffer, once a waiting buffer is chosen, the packet must wait for that specific buffer to become free. For case (2), the routing algorithm also has the possibility of waiting for a subset of more than one output buffer. In fact, case (2) includes any routing algorithm that does not conform to case (1). That is, any routing algorithm that does not select a specific waiting buffer and wait for that buffer until it becomes free. We first prove a necessary and sufficient condition for routing algorithms that belong to case (1), followed by a necessary and sufficient condition for routing algorithms that belong to case (2).

Theorem 2 *A routing algorithm, R_A, that requires a blocked packet to wait for a specific output buffer is deadlock-free iff R_A is wait-connected and the BWG for R_A has no True Cycles.*

Proof. First, note that R_A is wait-connected by definition. From theorem 1, an acyclic BWG is a sufficient condition for deadlock freedom. A False Resource Cycle cannot result in deadlock, so any False Resource Cycles can be ignored. Because there are no True Cycles, the routing algorithm is deadlock-free.

To prove necessity, assume that a True Cycle with n packets exists. A deadlock configuration can be created from this True Cycle. For each $i < n$, allow packet p_i to occupy buffer q'_i, possibly some additional buffers, and then wait for buffer q'_{i+1} occupied by packet p_{i+1}. (Assume that p_i and q'_i are defined as in theorem 1.) Similarly, packet p_n occupies buffer q'_n and waits for buffer q'_1. Since the cycle can be created without the simultaneous use of any buffer by more than one packet, it is possible to generate a set of packets that are able to occupy the required buffer(s) and then wait for the appropriate buffer. To force p_i to wait for q'_{i+1}, it is necessary to guarantee that *any* output buffer for p_i is occupied. For any output buffer available to p_i that is also available to the source node of buffer q'_{i+1}, assume the source has injected a packet that is using this buffer. If the routing algorithm is not suffix-closed, however, then some of the output buffers available to p_i may be unavailable to the source node. For each of these output buffers, assume that a previous packet, p_j, that was permitted to use this output buffer is currently occupying this output buffer. (For wormhole routing, the length of p_j is assumed to be short enough that it releases the input buffer that p_i occupies but long enough that it occupies the output buffer at this node.) Since p_j can occupy this buffer for an indefinite amount of time, it is always possible to force p_i to wait for q'_{i+1}. Clearly, each packet in the set is waiting for a buffer occupied by another packet in the set and none of the packets can make progress. Therefore, a deadlock configuration can always be constructed from a True Cycle. □

For routing algorithms that permit a packet to wait for *any* of the output buffers to become free, an acyclic BWG is not a necessary condition. Since each blocked packet has a choice of output buffers, packets may be able to avoid buffers that form cycles in the BWG by using an alternative buffer that is not part of a cycle. Deadlock can be avoided, however, only if at least one of the output buffers for which the packet is waiting is *guaranteed* to become free. For this reason, we selectively remove edges from the BWG to *resolve all True Cycles* as long as the routing algorithm remains wait-connected for the resulting graph, called BWG'. Removing edges from the BWG is equivalent to reducing the set of waiting buffers for a given buffer. Although the set of waiting buffers is reduced, there is no change to the set of output buffers the routing algorithm can use. We next prove that if no such BWG' exists, then the routing algorithm is not deadlock-free. If such a BWG' does exist, however, then the following theorem can be used to prove deadlock freedom:

Theorem 3 *A routing algorithm, R_A, that allows a blocked packet to wait for multiple output buffers is deadlock-free iff R_A is wait-connected for some BWG' and this BWG' has no True Cycles.*

Proof. If R_A is wait-connected for the BWG and the BWG has no True Cycles, then the result follows immediately from theorem 2, with $BWG = BWG'$. Assume the BWG contains True Cycles. In this case, R_A must be wait-connected for some BWG' without True Cycles.

We first prove sufficiency. Consider a potential deadlock configuration for R_A, involving a cycle of n packets ($n > 0$). This requires that every packet in the configuration is waiting for a buffer occupied by itself or another packet in the cycle. Because each packet can wait for multiple output buffers and R_A is wait-connected for BWG', at least one of the waiting buffers for each packet is in BWG'. Since BWG' has no True Cycles, an output buffer in BWG' eventually becomes free and some packet in the set, p_i, is forwarded. There is no guarantee, however, that the output buffer that p_i acquires is a buffer in BWG'. (It is possible that p_i proceeds to a different buffer before the buffer in BWG' becomes free.) If p_i has reached its destination, then the cycle has been resolved. Otherwise, whether or not p_i acquires a buffer in BWG', p_i can acquire an output buffer in BWG' at the next router, because R_A is wait-connected for BWG'. Hence, one of the packets can always be routed and a deadlock configuration cannot occur.

We now prove necessity by showing that the routing algorithm is not deadlock-free if every wait-connected BWG' has True Cycles. Assume that every wait-connected BWG' has True Cycles. Thus, it is possible to generate a set of packets that have no waiting buffer that is guaranteed to become available. Furthermore, these packets are all blocking each other, since otherwise it would be possible to guarantee that a waiting buffer does become free. Hence, this set of packets form a deadlock configuration. However, since the routing algorithm is deadlock-free, no deadlock configuration is possible. Therefore, a wait-connected BWG' without True Cycles must exist. □

Reducing the BWG to BWG' should not be difficult. For example, in Section 6.1 we present a proof that uses a straightforward reduction from the BWG to BWG'. For completeness, however, we present a formal method for reducing the BWG to an appropriate BWG'. Our approach consists of first distinguishing between False Resource Cycles and True Cycles. If the BWG is not acyclic, edges are removed from the BWG to generate BWG'. An overview of these two steps can be found in Section 5.

5 False Resource Cycles

The BWG is a static graph, however, the dependencies that arise among the buffers are dynamic. False Resource Cycles capture this notion of dynamic dependencies among the buffers. *When two edges in the BWG both require the use of a common buffer, then these two dependencies cannot occur simultaneously.* Cycles in the BWG that are formed from such dependencies cannot occur in reality, and hence, cannot lead to a deadlock configuration. False Resource Cycles can arise with minimal or nonminimal routing algorithms. Duato's incoherent routing algorithm is an example of the latter. An example of a False Resource Cycle using minimal routing can be found in [28].

In order to create a deadlock configuration, each packet in the set, p_i, must acquire q'_i before p_{i-1} arrives at q'_i. This is always possible with a True Cycle, since no buffer is occupied by more than one packet at a time. A False Resource Cycle requires that at least two packets share a buffer and this sharing leads to a cycle in the BWG. Such a cycle can arise in one of two ways. Either a buffer in the cycle is shared, or a buffer outside the cycle is shared simultaneously by more than one packet before entering the cycle. The second possibility can be ignored for suffix-closed routing algorithms, because a cycle can be created without using buffers outside the cycle. If the shared buffer is part of the cycle, then each packet, p_i, that occupies the shared buffer has already acquired q'_i. Hence, the False Resource Cycle would be a True Cycle if p_i could reach q'_{i+1} without using a shared buffer. On the other hand, if the shared buffer is used prior to the buffers in the cycle, then the False Resource Cycle exists because p_i cannot acquire q'_i before p_{i-1}. Note, however, the False Resource Cycle would be a True Cycle if p_i could reach q'_i without using a shared buffer.

Our method of distinguishing between True Cycles and False Resource Cycles requires examining the possible paths each packet in the cycle could use, avoiding shared buffers if possible, and backtracking as necessary to adjust paths used by other packets in the cycle. This is a complete solution for suffix-closed routing algorithms. For routing algorithms that are not suffix-closed, this is only a partial solution. It is then necessary to determine whether the buffers that are shared outside the cycle can be used consecutively instead of simultaneously. If this is the case, then sharing the buffer is possible. Otherwise, the buffer is a simultaneously shared buffer and the cycle is a False Resource Cycle. A complete description of the procedure for distinguishing between True Cycles and False Resource Cycles can be found in [28].

In practice, reducing the BWG to BWG' should not be difficult. For completeness, however, a formal method of reducing the BWG to BWG' is required. This reduction is only necessary for routing algorithms that do not require a packet to wait for a specific output buffer. The design methodology requires the identification of all True Cycles in the BWG and proceeds by removing a dependency from the BWG to resolve each cycle, backtracking as necessary to previously resolved cycles. This is an exponential time algorithm. Other general techniques for proving deadlock freedom also require exponential time in the worst case [9, 12, 22]. A description of the algorithm for reducing the BWG to BWG' can be found in [28].

6 Routing Examples

In order to demonstrate the usefulness of the necessary and sufficient condition, deadlock freedom is proved for a fully adaptive store-and-forward routing algorithm for n-dimensional meshes. We then prove deadlock freedom for a fully adaptive wormhole routing algorithm for hypercubes. The necessary and sufficient condition is then used to prove that any relaxation of the restrictions imposed by the routing algorithm introduces the possibility of deadlock.

6.1 Mesh Routing Algorithm

Many adaptive store-and-forward routing algorithms have been proposed, including those described in [4, 17, 21, 26]. These routing algorithms are buffer reservation algorithms and select an output buffer based on only local information, such as the availability of the buffers at neighboring nodes. To illustrate the simplicity of applying our proof technique and to demonstrate the ease with which the BWG can be reduced to BWG', we present an alternative proof of the fully adaptive minimal n-dimensional mesh routing algorithm proposed by Pifarré *et al.* [26].

This routing algorithm, which we will call the *Two-Buffer* routing algorithm, requires two standard buffers per router, labeled A and B, respectively. A packet remains in the A buffers until it has completed routing in the positive direction of *all* dimensions. Since minimal routing is used, the packet can move to the A buffer of *any* neighboring node that moves it closer to its destination. Once the packet has completed routing in all positive directions, the packet then moves to the B buffers and routes in the B buffers until the destination is reached.

Theorem 4 *The Two-Buffer routing algorithm is deadlock-free.*

Proof. The B buffers are used only when routing in the negative directions and a packet can never move from a B buffer to an A buffer. Since there are no wrap-around links on a mesh, it is obvious that no cycle can be created using the B buffers. Consider the A buffers. There are no restrictions on the use of the A buffers, provided that the packet needs to route in at least one positive direction. Hence, there are cycles in the BWG. Define BWG' such that all edges in the BWG from an A buffer to a neighboring A buffer in the negative direction are removed. Notice that BWG' remains wait-connected, since a packet is in an A buffer only if it needs to route in the positive direction of at least one dimension. Also, BWG' is acyclic, since there are no wrap-around links and all the edges are either from an A buffer to another A buffer in a positive direction or from an A buffer to a B buffer or from a B buffer to another B buffer in a negative direction. Deadlock freedom follows immediately from theorem 3. □

6.2 Hypercube Routing Algorithm

The following conventions are used to present the hypercube routing algorithm. $B^i_{n\pm}$ is used to denote buffer n in the $\pm$ direction of dimension i. For example, B^3_{1-} is buffer one in the negative direction of the third dimension. An asterisk in the superscript denotes all dimensions. Thus, B^*_{2-} denotes the second buffer in the negative direction of all dimensions. The $\pm$ is omitted when referring to the channels in both directions, so B^2_1 denotes the first virtual channel in either direction of the second dimension.

Each processor of an n-dimensional hypercube can be labeled with n bits. The source of a packet is denoted $S = (s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$ and the destination is denoted $D = (d_{n-1}, d_{n-2}, \ldots, d_1, d_0)$. A packet routes from the source to the destination by routing in dimensions in which the corresponding bit in the source differs from that of the destination. The packet routes in the *positive* direction of dimension i if $s_i = 0$ and $d_i = 1$. Similarly, the packet routes in the *negative* direction of dimension i if $s_i = 1$ and $d_i = 0$. With minimal routing, a packet routes in each dimension at most once and a packet does not route in dimension i if $s_i = d_i$.

Partially adaptive routing algorithms for wormhole-routed hypercubes and meshes have been proposed by many authors. A survey of these results can be found in [28]. A fully adaptive hypercube routing algorithm has been proposed by several authors [9, 18, 22, 29]. This routing algorithm requires two buffers in each direction of each dimension. A packet routes in dimension order along the first set of buffers. Each packet also has the possibility of routing in any dimension that moves the packet closer to the destination along the second set of buffers.

We now propose a fully adaptive minimal routing algorithm for hypercubes. This routing algorithm is substantially more adaptive than any previously proposed fully adaptive wormhole routing algorithm for hypercubes. All previous fully adaptive wormhole routing algorithms for hypercubes require that the first set of buffers be used only for nonadaptive routing. This new routing algorithm permits the first set of buffers to be used for partially adaptive routing. The routing algorithm is defined as follows:

Enhanced Fully Adaptive Routing Algorithm

- Assign two buffers to each direction of each dimension.
- Allow a packet to route along the second buffer at any time.

Let l be the lowest dimension in which the packet still needs to route. The first set of buffers is used in the following way:

- A packet that needs to route in the negative direction of dimension l can use any of the first set of buffers.
- A packet that needs to route in the positive direction of dimension l *must* use B^l_{1+}.
- If all output buffers a packet can use are occupied, the packet waits for B^l_1.

The Enhanced Fully Adaptive routing algorithm restricts a packet from using the first buffer in the positive direction after using the first buffer in a higher dimension. However, a packet can use the first buffer in a higher dimension whenever the packet needs to route in the negative direction of the lowest dimension in which the packet still needs to route. This is a significant relaxation of the routing restrictions, especially when compared with dimension-order routing.

The degree of adaptiveness is the ratio of the number of paths permitted by the routing algorithm to the total number of paths, averaged over all source-destination pairs [16]. The degree of adaptiveness for the Enhanced Fully Adaptive routing algorithm and Duato's routing algorithm is shown in Figure 3. For comparison, the degree of adaptiveness for *e*-cube (nonadaptive dimension order) routing is also shown. Paradoxically, the degree of adaptiveness is not zero for nonadaptive routing. Nonadaptive routing always allows one path and the degree of adaptiveness is zero only when there are *no* permitted paths.

The results clearly show the increase in adaptiveness that the Enhanced Fully Adaptive routing algorithm exhibits. Duato's routing algorithm, which was previously the most adaptive, has a significant decrease in the degree of adaptiveness as the size of the hypercube increases. The Enhanced Fully Adaptive routing algorithm also has a decreasing degree of adaptiveness, however, the decrease is much more modest. The difference in the degree of adaptiveness between Duato's routing algorithm and Enhanced Fully Adaptive becomes more pronounced as the number of dimensions increase. For a 12D hypercube, Duato's has a degree of adaptiveness of about 16%, while the corresponding number for Enhanced Fully Adaptive is over 50%.

Theorem 5 *The Enhanced Fully Adaptive routing algorithm is deadlock-free.*

Proof. This proof uses the sufficient condition proved in theorem 1. R_A is wait-connected, since a packet is always permitted to wait for buffer one in the lowest dimension in which the packet needs to route. Since a packet can wait for only B_1^* buffers, any cycle in the BWG must be created from waiting dependencies among the B_1^* buffers (although the intermediate use of B_2^* buffers could be used to create these dependencies). Wormhole routing provides separate buffers in each direction of a dimension and minimal routing is used, so any cycle in the BWG requires at least two dimensions and must use both directions of each dimension in the cycle. Let l be the lowest dimension of the potential cycle. A cycle in the BWG requires one of two situations: either a packet, p_i, waits for B_{1+}^l while occupying B_1^* in a higher dimension of the cycle or p_i occupies B_{1+}^l or B_{2+}^l as well as B_1^* in a higher dimension and then waits for B_1^* in a different higher dimension. Since l is the lowest dimension in the cycle, p_i does not need to route in any dimension lower than l after using B_1^* in a higher dimension. Otherwise, p_i would be waiting for a buffer in this lower dimension and l would not be the lowest dimension in the cycle. Clearly, neither situation can occur, since p_i cannot use B_1^* in a higher dimension when p_i needs to route in B_+^l and l is the lowest dimension in which p_i needs to route. Since the BWG is acyclic, the routing algorithm is deadlock-free. □

Theorem 6 *No restrictions imposed on the Enhanced Fully Adaptive routing algorithm can be relaxed without permitting a deadlock configuration.*

Proof. There are no restrictions on the use of the second set of buffers, so the restrictions on only the first set of buffers must be considered. The only restriction on the first set of buffers is that a buffer in a higher dimension cannot be used by a packet that needs to route in the positive direction of the lowest dimension in which the packet needs to route. Let l be the lowest dimension and assume that the buffer used in a higher dimension is the first buffer in the positive direction of dimension i. Then there is an edge in the BWG from B_{1+}^i to B_{1+}^l. The BWG already has edges from B_{1+}^l to B_{1-}^i, from B_{1-}^i to B_{1-}^l, and from B_{1-}^l to B_{1+}^i. These edges form a True Cycle. A packet waits for only B_1^l and the routing algorithm is not wait-connected if B_1^l is not a waiting buffer. The existence of a deadlock configuration follows immediately from theorem 2. □

7 Conclusion

A necessary and sufficient condition for proving deadlock freedom for both nonadaptive and adaptive interconnection network routing algorithms has been proposed. This is a universal necessary and sufficient condition, which can be used with *any* switching technique that is not inherently deadlock-free and can be applied to any network topology. The usefulness of our proof technique has been demonstrated with a fully adaptive store-and-forward routing algorithm for n-dimensional meshes. The technique described in this paper has also been used to prove deadlock freedom for a fully adaptive wormhole routing algorithm for hypercubes. This new hypercube routing algorithm is substantially more adaptive than any previous fully adaptive routing algorithm for hypercubes.

We have shown that the restrictions on the Enhanced Fully Adaptive routing algorithm cannot be relaxed without creating a deadlock configuration. It is possible, although unlikely, that a routing algorithm with a different set of restrictions could be deadlock-free while being less restrictive. The number of restrictions cannot be reduced, however, since there is only one restriction for each pair of dimensions.

A partial result has been given for distinguishing between False Resource Cycles and True Cycles. In practice, the identification of False Resource Cycles should not require resorting to this formal technique. This technique works for all suffix-closed routing algorithms, which represent all but a very restricted class of routing algorithms. Providing an algorithm to distinguish between False Resource Cycles and

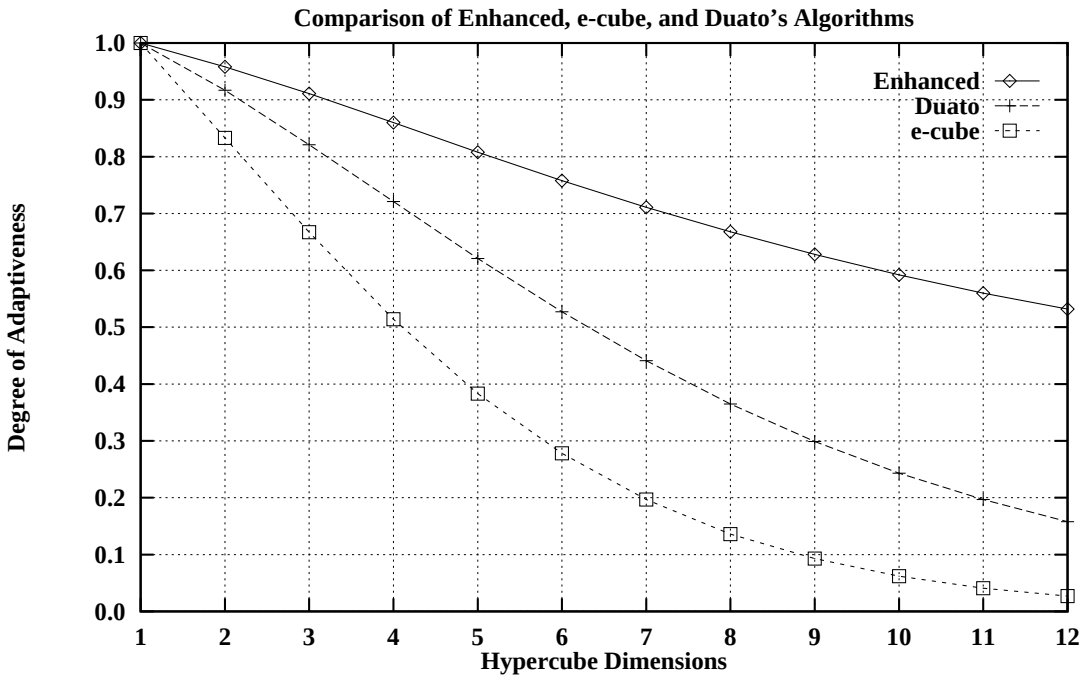

Figure 3: Degree of Adaptiveness for Hypercube Routing Algorithms

True Cycles for routing algorithms that are not suffix-closed is an open problem.

The use of theorem 3 requires the identification of BWG'. This is needed only for routing algorithms that do not require a packet to wait for a specific output buffer. It should be straightforward to determine BWG' for regular topologies such as k-ary n-cubes and meshes. However, a formal design methodology has been provided for those cases where it is difficult to reduce the BWG to BWG'. This automates the task of proving deadlock freedom and should be of use for routing algorithm designers.

Acknowledgments

The authors thank José Duato for many helpful and detailed suggestions; Yu-Chee Tseng for insightful comments on the idea of buffer waiting; Jeff May for valuable discussions concerning the necessary and sufficient condition; and Anish Arora, Dave Lutz, and Kant Patel for many thoughtful comments which have improved the quality of the paper.

References

[1] P. Berman, L. Gravano, G. Pifarré, and J. Sanz. Adaptive Deadlock- and Livelock-Free Routing With All Minimal Paths in Torus Networks. In 4^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 3–12, 1992.

[2] Y. M. Boura and C. R. Das. A Class of Partially Adaptive Routing Algorithms for n_dimensional Meshes. In *International Conference on Parallel Processing*, volume III, pages 175–182, 1993.

[3] A. A. Chien. A Cost and Speed Model for k-ary n-cube Wormhole Routers. In *Hot Interconnects '93*, August 1993.

[4] R. Cypher and L. Gravano. Adaptive, Deadlock-Free Packet Routing in Torus Networks with Minimal Storage. In *International Conference on Parallel Processing*, volume III, pages 204–211, 1992.

[5] R. Cypher and L. Gravano. Requirements for Deadlock-Free, Adaptive Packet Routing. *SIAM Journal on Computing*, 23(6):1266–1274, December 1994.

[6] W. J. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[7] W. J. Dally and H. Aoki. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[8] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[9] J. Duato. On the Design of Deadlock-Free Adaptive Routing Algorithms for Multicomputers: Design Methodologies. In *Parallel Architectures and Languages Europe 91*, volume I, pages 390–405, 1991.

[10] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. Technical report, Universidad Politecnica de Valencia, 1993.

[11] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[12] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. In *International Conference on Parallel Processing*, volume I, pages 142–149, 1994.

[13] P. T. Gaughan and S. Yalamanchili. Adaptive Routing Protocols for Hypercube Interconnection Networks. *IEEE Computer*, 26(5):12–23, May 1993.

[14] C. Glass and L. M. Ni. Maximally Fully Adaptive Routing in 2D Meshes. In *International Conference on Parallel Processing*, volume I, pages 101–104, 1992.

[15] C. Glass and L. M. Ni. The Turn Model for Adaptive Routing. In 19^{th} *Annual International Symposium on Computer Architecture*, pages 278–287, 1992.

[16] C. Glass and L. M. Ni. The Turn Model for Adaptive Routing. *Journal of the Association for Computing Machinery*, 41(5):874–902, September 1994.

[17] I. S. Gopal. Prevention of Store-and-Forward Deadlock in Computer Networks. *IEEE Transactions on Communications*, COM-33(12):1258–1264, December 1985.

[18] L. Gravano, G. Pifarré, G. Denicolay, and J. Sanz. Adaptive Deadlock-free Worm-hole Routing in Hypercubes. In *International Parallel Processing Symposium*, pages 512–515, 1992.

[19] K. D. Günther. Prevention of Deadlocks in Packet-Switched Data Transport Systems. *IEEE Transactions on Communications*, COM-29(4):512–524, April 1981.

[20] P. Kermani and L. Kleinrock. Virtual Cut-Through: A New Computer Communication Switching Technique. *Computer Networks*, 3(4):267–286, September 1979.

[21] S. Konstantinidou and L. Snyder. Chaos router: architecture and performance. In 18^{th} *Annual International Symposium on Computer Architecture*, pages 212–221, 1991.

[22] X. Lin, P. K. McKinley, and L. M. Ni. The Message Flow Model for Routing in Wormhole-Routed Networks. In *International Conference on Parallel Processing*, volume I, pages 294–297, 1993.

[23] J. May, D. N. Jayasimha, and K. Patel. Comparison of Multiplexing Schemes for Wormhole-Routed Distributed Memory Multiprocessors. In 1^{st} *International Workshop on Parallel Processing*, pages 211–215, 1994.

[24] J. Y. Ngai and C. L. Seitz. A Framework for Adaptive Routing in Multicomputer Networks. In 1^{st} *Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–9, 1989.

[25] L. M. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, 26(2):62–76, February 1993.

[26] G. D. Pifarré, L. Gravano, S. A. Felperin, and J. L. C. Sanz. Fully Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks: Algorithms and Simulations. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):247–263, March 1994.

[27] L. Schwiebert and D. N. Jayasimha. Optimal Fully Adaptive Wormhole Routing for Meshes. In *Supercomputing '93*, pages 782–791, 1993. An extended version of this paper will appear in the Journal of Parallel and Distributed Computing.

[28] L. Schwiebert and D. N. Jayasimha. A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing. Technical Report OSU-CISRC-4/94-TR22, The Ohio State University, April 1994. Revised December 10, 1994. Accepted for publication in the Journal of Parallel and Distributed Computing.

[29] C.-C. Su and K. G. Shin. Adaptive Deadlock-Free Routing in Multicomputers Using Only One Extra Virtual Channel. In *International Conference on Parallel Processing*, volume I, pages 227–231, 1993.

[30] S. Toueg and K. Steiglitz. Some Complexity Results in the Design of Deadlock-Free Packet Switching Networks. *SIAM Journal on Computing*, 10(4):702–712, November 1981.