

# Deadlock-Free Oblivious Wormhole Routing with Cyclic Dependencies

Loren Schwiebert  
Parallel and Distributed Computing Laboratory  
Department of Electrical and Computer Engineering  
Wayne State University  
Detroit, MI 48202–3902  
Email: loren@ece.eng.wayne.edu

## Abstract

A great deal of recent work has been done on developing techniques for proving deadlock freedom for wormhole routing algorithms. One approach has been to restrict the class of routing algorithms for which the proof technique applies. The other approach is to provide a generic method that can be applied to all routing algorithms. Although this latter approach offers clear advantages, a general technique must deal with many complications. Foremost among these is the issue of irreducible cyclic dependencies that cannot result in deadlock. Such dependencies have been referred to alternatively as *unreachable configurations* and *false resource cycles*. In this paper, we apply the notion of unreachable configurations to oblivious routing algorithms and thereby provide a counter-example to Dally and Seitz's [6] theorem for deadlock-free routing in wormhole-routed networks. This idea is then further developed to show various restrictions on when unreachable configurations can exist with oblivious routing algorithms.

## 1 Introduction

Wormhole routing [6] has become the switching technique of choice in most modern distributed-memory multiprocessors. Wormhole routing propagates messages through the network by dividing each message into packets, which are further divided into flits. The header flit of a packet contains the routing information and the data flits of the packet follow the header flit through the network. The network treats each packet as a separate message, so we use the terms message and packet interchangeably. The major advantage of wormhole routing is that when the header arrives at an intermediate router, the router forwards the message header to a neighboring router as soon as an output channel the message can use is available.

To appear in the 9<sup>th</sup> Annual ACM Symposium on Parallel Algorithms and Architectures, June 22–25, 1997, Newport, Rhode Island.

Since the flits of a message are forwarded as soon as possible, the message latency is largely insensitive to the distance between the source and destination. On the other hand, packet switching buffers the entire message at every intermediate node before forwarding any part of the message. Hence, wormhole routing has lower message latency when there is little or no channel contention. Although virtual cut-through also allows messages to progress as soon as an output channel is available, wormhole routing requires only enough storage at each router to buffer a few flits, rather than the entire packet. These two properties account for the popularity of wormhole routing in distributed-memory multiprocessors. See Ni and McKinley [13] for a detailed explanation of wormhole routing.

The primary drawback to wormhole routing is the contention that can occur even with moderate traffic, which leads to higher message latency. Whenever a message is unable to proceed due to contention, the header and data flits are not removed from the network. Instead, the message holds all the channels it currently occupies. A message that requires several channels can block many messages during transmission. These blocked messages can in turn block other messages, which further increases the message latency.

A cost-effective method of reducing message latency, proposed by Dally [4], is to allow multiple *virtual* channels to share the same physical channel. Each virtual channel has a separate buffer, with multiple messages multiplexed over the same physical channel. Both latency and contention can be further reduced by using the multiple paths that exist in the network between the source and destination nodes. Dally and Seitz [6] have shown, however, that since a message holds channels until the entire message has been transmitted, a routing algorithm with no restrictions on the use of virtual or physical channels can result in deadlock.

The simplest routing algorithms are *oblivious* (nonadaptive) and define a single path between the source and destination. Adaptive routing algorithms, on the other hand, support multiple paths between the source and destination. A routing algorithm is either minimal or nonminimal. Minimal routing algorithms allow only shortest paths to be chosen, while nonminimal routing algorithms do not require messages to

use only shortest paths. Minimal routing algorithms provide higher throughput for high message traffic and are generally simpler to implement. Nonminimal routing algorithms are useful for fault tolerance.

Recent research on routing algorithms for wormhole routing has provided many different techniques for proving deadlock freedom. In this paper, we build upon these results to show that, even with oblivious routing, cyclic dependencies do not necessarily imply deadlock. In addition, we show several restrictions on when a deadlock-free oblivious routing algorithm can have cyclic dependencies. Finally, we discuss the implications of these results for a general methodology for proving deadlock freedom.

The remainder of the paper is organized as follows. In section 2, previous work on deadlock-free routing is reviewed, and in section 3 we present our model and assumptions. In section 4, we present an oblivious routing algorithm for wormhole routing that is deadlock-free, but does not have an acyclic channel dependency graph. In section 5, we prove several conditions that must be satisfied in order to have irreducible cyclic dependencies in a deadlock-free oblivious routing algorithm. Finally, in section 6, we discuss the implications of this result and describe possible extensions for future work.

## 2 Previous Work

Designing deadlock-free routing algorithms for wormhole routing was simplified by Dally and Seitz with a proof that an acyclic channel dependency graph guarantees deadlock freedom [6]. Each vertex of the channel dependency graph is a physical or virtual channel. There is a directed edge from one channel to another if a message is permitted to use the second channel immediately after the first. Since the graph is acyclic, deadlock freedom can be shown by assigning a numbering to the edges of the graph, ensuring that all channels are used in strictly increasing or strictly decreasing order.

Dally and Seitz proposed their proof technique for oblivious routing algorithms. Oblivious routing algorithms can be characterized by functions of the form  $R : C \times N \times N \rightarrow C$ , where the input channel, belonging to the set of channels  $C$ , and the current and destination nodes, belonging to the set of nodes  $N$ , define an output channel on which to route the message. An acyclic channel dependency graph has also been used as a basis for developing adaptive routing algorithms defined by functions of the form  $R : C \times N \times N \rightarrow \mathcal{P}(C)$ , where a set of output channels, taken from  $\mathcal{P}(C)$  – the power set of  $C$ , is defined on which to route the message.

Both Glass and Ni [10], and Boura and Das [2] have proposed methodologies for generating deadlock-free routing algorithms. Both proof techniques require an acyclic channel dependency graph. Glass and Ni propose a method of analyzing routing algorithms based on the permitted and prohibited dependencies from one channel to another. These de-

pendencies are characterized as turns, with the set of possible turns defined by the topology. The *turn model* groups the turns into cycles and breaks all cycles by prohibiting some turns. Boura and Das propose a method of proving deadlock freedom by partitioning the channels into two acyclic sets and requiring messages to route completely in the first set before using channels in the second set.

Duato [7, 8] proved that an acyclic channel dependency graph was *not* a necessary condition for deadlock-free routing for *adaptive* routing algorithms defined by functions of the form  $R : N \times N \rightarrow \mathcal{P}(C)$ , where the current node and the destination node, independent of the input channel, define the set of output channels on which to route the message. Schwiebert and Jayasimha [14, 16] have used Duato’s sufficiency condition and the mesh topological properties to propose a fully adaptive routing algorithm for arbitrary dimension mesh networks that is optimal in the number of virtual channels required and in the number of restrictions placed on the use of these virtual channels. Berman, *et al.* [1] propose a torus routing algorithm that allows cyclic dependencies among the channels.

Dally and Aoki [5] 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  $p_i$  is waiting for a channel held by  $p_j$ .

All these proof techniques provide only a sufficient condition for deadlock-free adaptive routing. Dally and Seitz’s [6] proof that an acyclic channel dependency graph guarantees deadlock freedom was originally proposed as a necessary and sufficient condition for deadlock-free routing algorithms. As previously mentioned, Duato showed that this claim does not hold for adaptive routing algorithms, however, it was generally believed [7, 8, 12, 15, 17] that an acyclic channel dependency graph was required for deadlock-free *oblivious* routing. A counter-example is shown in this paper, by presenting a deadlock-free oblivious routing algorithm with cyclic dependencies.

Finding a necessary and sufficient condition for deadlock-free routing remained an open problem. One of the difficulties of providing a necessary and sufficient condition for deadlock-free routing arises from the problem of *unreachable configurations*. Unreachable configurations were identified by Cypher and Gravano [3] and used to show that it may be impossible to remove cyclic dependencies from a deadlock-free routing algorithm. An unreachable configuration consists of a set of channel dependencies that cannot exist simultaneously because of interdependencies among these dependencies. This is possible because the actual dependencies at any instance rely on the *dynamic* interaction among messages in the network at that time, but the channel dependencies are *static*, in that they are determined by the routing algorithm. Schwiebert and Jayasimha [15, 17] prove that an unreachable configuration can occur only when at

least two of the dependencies in a configuration require the simultaneous use of some channel in the interconnection network. For this reason, they refer to unreachable configurations as *false resource cycles*.

Lin, McKinley, and Ni [12] propose a proof technique based on the observation that a routing algorithm is deadlock-free if none of the channels in the network can be held forever. If every message that uses a given channel is guaranteed to reach its destination, then that channel is called a *deadlock-immune* channel. Since sink channels cannot be part of a deadlock configuration, the proof starts with the sink channels and works backward through the network. If it is possible to show that no channel can be held forever by a message, regardless of the destination and subsequent path taken, then the routing algorithm is deadlock-free. This proof technique was proposed as a necessary and sufficient condition, however, it is not clear how to apply this approach to routing algorithms with unreachable configurations. Channels are shown to be deadlock-immune by their association with neighboring deadlock-immune channels. The channels in an unreachable configuration form a cycle, however, so there is no starting point from which to deduce that these are deadlock-immune channels.

Duato [9] has proposed a necessary and sufficient condition for deadlock freedom for adaptive routing algorithms of the form  $R : N \times N \rightarrow \mathcal{P}(C)$ . The class of routing algorithms is restricted further to ensure completeness and prevent unreachable configurations. First, the routing algorithm *must* provide a minimal path between every pair of nodes, even for nonminimal routing algorithms. Second, the routing algorithm must be *coherent*. A routing algorithm is coherent if it permits every partial path from any source to any destination to be used by the same source to reach an intermediate node on the path or by an intermediate node on the path to reach the same destination. Many nonminimal routing algorithms are not coherent.

A general necessary and sufficient condition for deadlock-free routing was proved by Schwiebert and Jayasimha [17]. It has none of the restrictions imposed by Duato's proof technique. Furthermore, this methodology can be applied to routing algorithms of the form  $R : C \times N \times N \rightarrow \mathcal{P}(C)$  and has been generalized to virtual cut-through and packet switching [15]. Because the result applies to a broad class of routing algorithms, unreachable configurations must be addressed. This is done by introducing the notion of *false resource cycles* and providing a design methodology that distinguishes false resource cycles from cycles that produce deadlock configurations. This proof technique was recently generalized by Jayasimha, *et al.* [11] to support routing algorithms for collective communication, including multicasting.

After introducing the assumptions and definitions used in this paper, we will describe how the notions of *unreachable configuration* and *false resource cycle* affect the requirements for deadlock-free oblivious routing. The definitions

are specific to oblivious routing, although these definitions can be easily generalized to include adaptive routing algorithms [17].

### 3 Assumptions and Definitions

This paper adopts a conservative view of deadlock freedom. This approach is taken to avoid the possibility of designing routing algorithms that seem deadlock-free, but deadlock freedom actually depends on implementation aspects. This means deadlock freedom must be independent of certain physical properties of the network, such as the flit buffer size or the minimum message length. No restriction is imposed on message length. Similarly, arbitrary flit buffer sizes are allowed. For instance, if a routing algorithm would deadlock when the routers have a buffer size of one flit, the routing algorithm is considered *not* deadlock-free.

Furthermore, when multiple messages arrive simultaneously and request the same output channel, and one of these messages can lead to a deadlock, that message is assumed to acquire the channel. This is reasonable, since it is unlikely that neighboring routers are completely synchronous, so two messages arriving on the same clock cycle could actually arrive at slightly different times. The routers are assumed to all operate with the same network clock cycle time, however, so one message could not be forwarded over several channels while another message makes no progress even though its output channel is available.

Several assumptions and definitions are introduced to facilitate the presentation of the paper. These are standard assumptions made about wormhole routing and have also appeared in [6, 8].

1. A node can generate messages of arbitrary length destined for any other node at any rate.
2. A message arriving at its destination is eventually consumed.
3. Since wormhole routing is used, once a channel queue accepts the header flit of a message, it must accept all the flits of the message before accepting any flits from another message.
4. A channel queue cannot contain flits belonging to more than one message at a time. The channel must transmit the last flit of the current message before the channel queue accepts the header flit of the next message.
5. A node arbitrates among messages which simultaneously request the same output channel. Messages already waiting for a channel are chosen in an order that prevents starvation.

**Definition 1** An *interconnection network*  $I$  is a strongly connected directed multigraph, denoted  $I = G(N, C)$ , where the vertices,  $n_i \in N$ , are the processors and the arcs,

$c_i \in C$ , are channels that connect neighboring processors. Each channel,  $c_i$ , can transmit messages from one processor, denoted  $s_i$ , to a neighboring processor, denoted  $d_i$ .

**Definition 2** A routing function has the form  $R : C \times N \times N \rightarrow C$  and specifies an output channel based on the input channel, the current node, and the destination of the message.

**Definition 3** A routing algorithm  $R_A$  on interconnection network  $I$  takes two node IDs as arguments and is represented by  $R_A(n_i, n_j)$ . For each source-destination pair,  $R_A$  defines the set of paths available to a message. The routing is accomplished by application of a routing function at each router between the source and destination of the message. The routing algorithm may be adaptive or nonadaptive; minimal or nonminimal.

**Definition 4** A configuration is an assignment of messages to channels. The header and data flits of each message are stored in the channel queues and each channel queue holds flits from at most one message. The leading channel is the channel the message has most recently acquired and its channel queue contains the message header. Any other channels occupied by this message contain only data flits. A configuration is legal if each message in the configuration occupies one or more consecutive channels; the message header is stored at the head of the leading channel queue that the message occupies; each message occupies only channels the routing algorithm permits the message to use; and the storage capacity of each occupied channel has not been exceeded.

**Definition 5** A reachable configuration is a legal configuration that can be produced by routing messages when starting from an empty network [3].

**Definition 6** A deadlock configuration for routing algorithm  $R_A$  on interconnection network  $I$  is a non-empty reachable configuration consisting of a set of messages,  $m_1, m_2, \dots, m_n, n \geq 1$ , where each message,  $m_i$ , in the set has acquired at least one channel. The header flit of  $m_i$  has not reached its destination and is unable to proceed because the output channel for  $m_i$  is unavailable. Moreover, the output channel for  $m_i$  is occupied by either data flits of  $m_i$  or the header or data flits of another message in the set. The data flits at the head of any other channel queue held by  $m_i$  are unable to proceed because the next channel queue occupied by  $m_i$  is full. Thus, each message is blocked and must wait for an unavailable waiting channel held by another message in the set.

**Note:** Because oblivious routing is used, each message is blocked by only one other message. The first channel that a message uses in the cycle blocks the previous message in the cycle and that is the only channel in the cycle where that message blocks another message in the cycle.

**Definition 7** Routing algorithm  $R_A$  is suffix-closed if a path that  $R_A$  permits from node  $n_i$  to node  $n_j$  through node  $n_k$  implies that  $R_A$  also permits the partial path from  $n_k$  to  $n_j$  when  $n_k$  is the source. Note that every routing algorithm with a routing function of the form  $R : N \times N \rightarrow \mathcal{P}(C)$  is suffix-closed. This definition is used to facilitate proving restrictions on when unreachable configurations can occur.

#### 4 A Counter-Example for Oblivious Routing

Although Duato presented a counter-example to the claim that an acyclic channel dependency graph is a necessary and sufficient condition for deadlock-free adaptive routing, no such counter-example for oblivious routing has been found. In this section, a counter-example for oblivious routing is presented. The channels and nodes for an oblivious routing algorithm that is deadlock-free even with cyclic dependencies is shown in figure 1. All channels are bi-directional, although the channels used to form the cycle are shown with the direction used for the cycle.

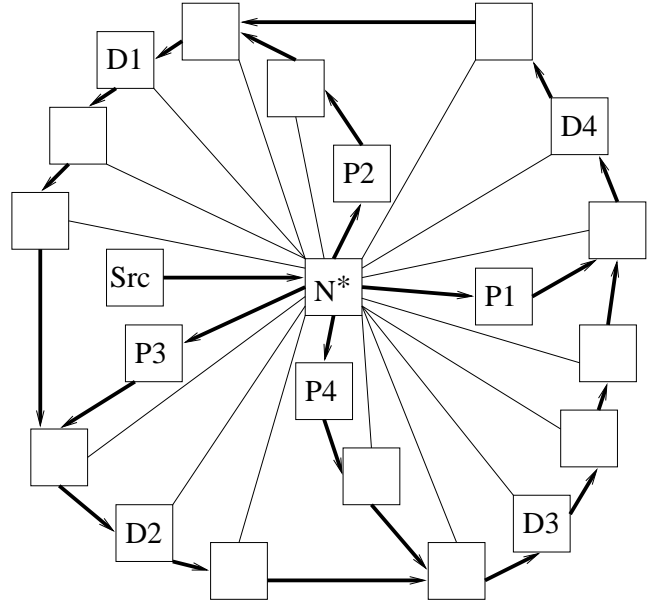


Figure 1: Topology for the Deadlock-Free Routing Algorithm

The routing algorithm associated with the interconnection network shown in figure 1 works as follows. If node  $N^*$  is the source, send the message directly to the destination. With four exceptions, messages from the other nodes are routed by sending the message from the source of the message to node  $N^*$ , which then forwards the message directly to the destination. Note that these messages never use the highlighted channels that form the cycle in figure 1. The four exceptions occur when node  $Src$  sends a message to destination  $D1$ ,  $D2$ ,  $D3$ , or  $D4$ . For these source-destination pairs, the message is routed from  $Src$  to  $N^*$ , and then along

the corresponding path. For example, the path for destination  $D1$  routes the message from  $N^*$  to node  $P1$  and then follows the cycle in the *counter-clockwise* direction until the message reaches  $D1$ . Note that there are cyclic dependencies, since the message destined for  $D1$ , message  $M1$ , routes through  $D4$ , the message destined for  $D2$ , message  $M2$ , routes through  $D1$ , the message destined for  $D3$ , message  $M3$ , routes through  $D2$ , and the message destined for  $D4$ , message  $M4$ , routes through  $D3$ . For convenience, this routing algorithm is called the *Cyclic Dependency* routing algorithm.

It is clear from the description of the routing algorithm that only the cycle generated by the four messages originating from node  $Src$  can lead to a deadlock configuration. If this cycle is actually an unreachable configuration (false resource cycle), then the routing algorithm is deadlock-free.

For this to be a deadlock configuration, message  $M1$  must reach  $D4$  before message  $M4$ . Similarly, message  $M2$  must reach  $D1$  before  $M1$ , message  $M3$  must reach  $D2$  before  $M2$ , and  $M4$  must reach  $D3$  before  $M3$ . In addition, each message must be long enough to hold all the channels in the cycle that that message uses to form the deadlock configuration. This means that  $M1$  and  $M3$  must hold at least three channels and  $M2$  and  $M4$  must hold at least four channels.

If the flit buffer size is larger than one flit, then messages  $M1$  and  $M3$  must be at least six flits each, which means that either message would reach  $D1$  or  $D3$ , respectively, at the same time that  $M2$  or  $M4$  enter the network. Similarly, since the messages must use the shared channel consecutively (rather than simultaneously), using a longer message allows that message to progress further before a subsequent message enters the network. Thus, if a deadlock configuration cannot be created when the buffer size is one flit and the messages have their minimum length, then the routing algorithm is deadlock-free.

**Theorem 1** *The Cyclic Dependency routing algorithm is deadlock-free.*

**Proof.** To form the cycle, messages  $M2$  and  $M4$  must hold four channels and  $M1$  and  $M3$  must hold three channels. Because of the intervening nodes,  $M2$  and  $M4$  must use three channels between node  $N^*$  and the cycle, while  $M1$  and  $M3$  use only two channels between node  $N^*$  and the cycle. This introduces a problem when trying to create the cycle. When  $M1$  releases the shared channel (the channel between  $Src$  and  $N^*$ ),  $M1$  needs to use only two more channels to bypass  $M2$ 's entry point into the cycle, but  $M2$  needs to use three more channels to block  $M1$ . Thus,  $M2$  must be injected before  $M1$  in order to block  $M1$ . For the same reason,  $M4$  must be injected before  $M3$ . If both  $M2$  and  $M4$  are injected prior to either  $M1$  or  $M3$ , however, then the first message injected ( $M2$  or  $M4$ ) is not blocked. Hence, it is impossible to inject both  $M2$  and  $M4$  before  $M1$  and  $M3$  and also block  $M2$  and  $M4$ .

A second possibility is to generate the cycle using more than four messages, by allowing one of the messages to temporarily block another message, thereby allowing subsequent messages to arrive in time to form a deadlock. Note that messages  $M1$  and  $M3$  use only two channels from the shared channel to the cycle and must hold three channels within the cycle. Hence, it is not beneficial to temporarily block  $M1$  or  $M3$ , because then none of the other messages are able to enter the network. Similarly, messages  $M2$  and  $M4$  use only three channels from the shared channel to the cycle and must hold four channels within the cycle.

Since it is impossible to generate a sequence of messages that form a deadlock, the cycle is a false resource cycle and the messages that would form the cycle comprise an unreachable configuration. Since this is the only cyclic dependency in the routing algorithm, the routing algorithm is deadlock-free.  $\square$

## 5 Required Conditions for Unreachable Configurations

An unreachable configuration similar to the one presented in section 4 would be unlikely to arise in a typical routing algorithm. That unreachable configuration has four messages that share a channel outside the cycle and the channels in the cycle can be used by only a restricted set of messages. A general technique for proving deadlock freedom, however, must be able to resolve such unlikely situations. One way of simplifying the process of identifying false resource cycles is to demonstrate restricted circumstances under which cycles can be unreachable.

As previously mentioned, an unreachable configuration results from a set of channel dependencies that require the *simultaneous* use of a channel. The Cyclic Dependency routing algorithm has an instance of this, where the cycle can be formed only if multiple messages use the channel between  $Src$  and  $N^*$  at the same time. Since simultaneous use of a channel is not possible, the configuration cannot be generated. Under some circumstances, however, it may be possible for messages to use the shared channel consecutively instead of simultaneously and still form a cycle. To explore these possibilities further, the cycles are divided between configurations with the shared channel within the cycle and those with the shared channel outside the cycle. A shared channel is considered to be within the cycle only when the shared channel is within the cycle for *all* messages in the cycle that use the channel.

**Theorem 2** *An oblivious routing algorithm cannot have an unreachable configuration when all shared channels are within cycles.*

**Proof.** Since the shared channel is within the cycle, any message,  $M_i$ , that uses it has already blocked some other message,  $M_j$ . All the messages in the configuration can use their initial channel in the cycle simultaneously, because no channel sharing is required prior to the cycle. If  $M_i$  reaches

a shared channel that is already in use by another message, then  $M_i$  is blocked because  $M_i$  can use only that channel. Thus, every message is blocked after blocking the previous message in the cycle, so a deadlock configuration has been produced.  $\square$

**Corollary 1** *A routing algorithm with a routing function of the form  $R : N \times N \rightarrow C$  has no unreachable configurations.*

**Proof.** If a routing function has the form  $R : N \times N \rightarrow C$ , then any cycle can be generated using only channels in the cycle. Each message can originate from the node connected to the first channel that message uses in the cycle. This ensures that no channels are used outside of the cycle. Since a shared channel is required for an unreachable configuration, the proof follows immediately from theorem 2.  $\square$

**Corollary 2** *A suffix-closed routing algorithm with a routing function of the form  $R : C \times N \times N \rightarrow C$  has no unreachable configurations.*

**Proof.** The proof follows immediately from corollary 1 and the fact that if the routing algorithm is suffix-closed, then a routing function of the form  $R : C \times N \times N \rightarrow C$  reduces to the form  $R : N \times N \rightarrow C$ .  $\square$

Restrictions can also be placed on the situations in which sharing a channel outside the cycle leads to an unreachable configuration. In order to facilitate the explanation of these theorems, figures 2 and 3 have been provided. For clarity, only the channels used to form the cycles are depicted. In figure 2, each channel is labeled with the messages in the cycle that could use that channel.

**Theorem 3** *If a shared channel outside of the cycle is used by only two messages, the cycle forms a deadlock configuration.*

**Proof.** Since the shared channel is outside the cycle, the messages can use the channel in consecutive order, provided that they both arrive in the cycle in time to block the preceding message in the cycle and arrive late enough to also become blocked. Since there are only two messages that use the shared channel, any other messages in the cycle can arrive at the appropriate times. Order the two messages that use the shared channel such that the message with the longer path from the shared channel to the cycle is first.<sup>1</sup> In figure 2, the first message would be M1. The length of both messages is minimized, so that each message is only long enough to hold the channels within the cycle. Furthermore, the flit buffers hold only a single flit. Immediately after the first message has traversed the shared channel, the second message starts traversing the shared channel. The distance that the message header of the first message must travel from its current position to the channel in the cycle that blocks

this message is equal to the distance from the shared channel to the first channel in the cycle. The distance that the message header of the second message has to travel from the shared channel to the *first* channel in the cycle is no more than the distance that the first message still has to travel. This allows *all* the messages in the cycle to enter the cycle before any message reaches the channel where it is blocked. Note that this approach works whether both messages use the shared channel outside the cycle or one message uses the shared channel within the cycle and the other outside the cycle. Thus, the cycle can be created and this is a deadlock configuration.  $\square$

An example has been shown where a false resource cycle can be created using a channel that is shared by four messages prior to entering the cycle. Theorems 2 and 3 prove that with oblivious routing, a false resource cycle is not possible if the shared channel is within the cycle or only two messages share a channel outside of the cycle. The possibility of generating an unreachable configuration with three messages sharing a channel is considered next.

Eight conditions must be satisfied in order to produce a false resource cycle with oblivious routing when only three messages in a cycle share a channel. Necessity and sufficiency is proved in theorem 4.

In presenting these conditions, the messages are labeled by the number of channels used between the shared channel and the cycle, so  $m_l$  refers to the message that uses the most channels between the shared channel and the first channel that message uses in the cycle. Similarly, the message using the fewest channels between the shared channel and the cycle is labeled  $m_s$  and the third message using the shared channel is labeled  $m_m$ . Note that this does not necessarily mean that  $m_l$  uses the most channels in the cycle, only that this message has the longest path from the shared channel to the cycle.

1. Relative to the cycle, the order of the messages using the shared channel is such that  $m_l$  is followed by  $m_s$ . Note that there could be other messages between these two messages, but  $m_m$  is not between  $m_l$  and  $m_s$ .
2. All three messages use the shared channel outside of the cycle.
3. All three messages use a different number of channels from the shared channel to the cycle.
4. Message  $m_l$  uses more channels within the cycle than it uses from the shared channel to the cycle.
5. If the message in the cycle that *immediately* precedes message  $m_s$  in the cycle does not use the shared channel, then message  $m_s$  uses more channels within the cycle than it uses from the shared channel to the cycle.
6. Either message  $m_m$  uses more channels within the cycle than it uses from the shared channel to the cycle

<sup>1</sup>If the distance is the same for both messages, choose either.

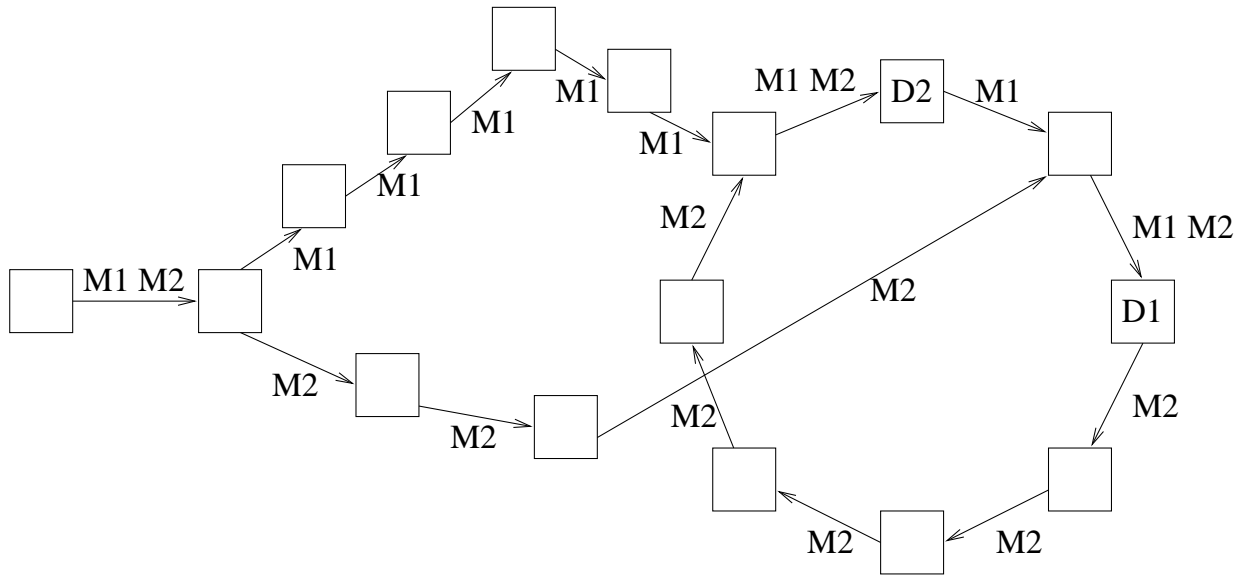


Figure 2: A Deadlock with a Channel Shared by Two Messages

or message  $m_s$  immediately precedes  $m_m$  in the cycle and  $m_s$  uses fewer channels from the shared channel to the first channel  $m_m$  uses in the cycle than message  $m_l$  uses from the shared channel to the cycle.

7. The number of channels used by  $m_l$  from the shared channel to the cycle plus the number of channels used by other messages in the cycle between  $m_l$  and  $m_s$  is less than the number of channels used in the cycle by  $m_m$  plus the number of channels used by  $m_s$  from the shared channel to the cycle.
8. The number of channels used by  $m_s$  from the shared channel to the cycle plus the number of channels used by other messages in the cycle between  $m_s$  and  $m_m$  is less than the number of channels used by  $m_m$  from the shared channel to the cycle.

In order to facilitate the understanding of these conditions, six different cycles are presented in figure 3. The routing algorithm is similar to the routing algorithm presented in section 4, except that the channels not used in the cycle have been removed for clarity. In all six cases,  $m_l$  is message 1,  $m_s$  is message 2, and  $m_m$  is message 3. Figure 3(a) depicts a false resource cycle that occurs because all three messages use more channels within the cycle than they use from the shared channel to the cycle. Figure 3(b) depicts a false resource cycle that occurs even though message  $m_m$  can be blocked between the shared channel and the cycle, because message  $m_s$  is too short to block  $m_m$  long enough. The deadlock shown in figure 3(c) occurs because condition 4 is not satisfied. Similarly, figure 3(d) shows a deadlock that occurs when condition 6 is not satisfied, because the path taken by message  $m_s$  is too long. Figure 3(e) depicts a deadlock

that arises when condition 7 is not satisfied. Finally, figure 3(f) introduces a fourth message, which routes from  $S_4$  to  $D_4$  and does not use the shared channel. This configuration does not satisfy conditions 6 or 8, so a deadlock can be generated.

**Theorem 4** *If a channel is shared by exactly three of the messages in a cycle, that cycle is an unreachable configuration iff all eight of the preceding conditions hold.*

**Proof.** Note that if only one of the messages uses the shared channel from outside the cycle, that message can acquire the shared channel at the same time that the other two messages acquire their first channel in the cycle. The first message can be made long enough to hold the shared channel, so the other two messages are blocked. The other messages do not use the shared channel, so each of these messages can complete its part of the cycle in time to block the preceding message. Thus, a deadlock configuration has been formed.

Likewise, if only two messages use the shared channel from outside the cycle, then the approach taken in theorem 3 for ordering the messages can be used to produce a deadlock configuration. The message that uses the shared channel within the cycle can be started so that it arrives at the shared channel immediately after the second message starts using this channel. Hence, an unreachable configuration is not possible unless all three messages use the shared channel prior to entering the cycle.

Since the shared channel is used outside the cycle by all three messages, these messages must use the channel in consecutive order. A deadlock occurs if they all arrive in the cycle in time to block the preceding message in the cycle and arrive late enough to also become blocked. Since there are only three messages that use the shared channel, any other

messages can enter the cycle at the appropriate times. The sequence of messages that form the cycle starts with one of the messages using the shared channel, followed by zero or more messages not using the shared channel, followed by another message using the shared channel again followed by zero or more messages not using the shared channel, and then again followed by a message that uses the shared channel and then possibly additional messages that do not use the shared channel.

If one of the messages in the cycle uses at least as many channels from the shared channel to the cycle as it uses within the cycle, then it can be blocked outside of the cycle after using the shared channel. If the preceding message in the cycle does not use the shared channel, then that message can block this message indefinitely (by creating a long enough message). This effectively reduces the cycle to a case of only two messages using the shared channel and a deadlock configuration can be constructed as shown in the proof of theorem 3.

Order the three messages that use the shared channel such that  $m_l$  is first, followed by the other two messages in the order in which they appear in the cycle. If this ordering of the messages places them in the order  $m_l$  followed by  $m_m$  and then  $m_s$ , a deadlock configuration can be generated when these messages use the shared channel consecutively in this order. The proof is similar to the proof of theorem 3.

It is easily verified that such an ordering is possible unless  $m_l$  is followed by  $m_s$  and all three messages use a *different* number of channels from the shared channel to the cycle. In this case,  $m_s$  could bypass  $m_m$  in the cycle if  $m_s$  uses the shared channel before  $m_m$ . If the other messages in the cycle (if any) interposed between these two messages ( $m_s$  and  $m_m$ ) use at least as many channels as the difference between the number of channels used from the shared channel to the cycle by  $m_m$  and  $m_s$ , then  $m_s$  is successfully blocked and a deadlock is constructed.

Otherwise, it may be possible to generate a deadlock from the cycle by injecting  $m_m$  before  $m_s$ , but after  $m_l$ . In this case,  $m_l$  could bypass  $m_s$ . This does not occur, however, if  $m_s$  reaches the cycle in time.

Message  $m_s$  can acquire the shared channel as soon as  $m_m$  is finished with it. At this time,  $m_l$  has traveled as many channels as  $m_m$  uses in the cycle, because that is the length of  $m_m$ . If  $m_l$  still needs to travel at least as far to reach its blocking channel as  $m_s$  needs to travel to reach the cycle, then the deadlock configuration can be formed. Of course, messages interposed between  $m_l$  and  $m_s$  can be used to provide the necessary additional channels.

The remaining possibility is when none of the above cases are satisfied. In this case,  $m_l$  is temporarily blocked outside the cycle so that the other two messages can reach the cycle in time. This temporary blocking is beneficial only when  $m_l$  uses no more channels within the cycle than  $m_l$  uses from the shared channel to the cycle, because then  $m_l$  can be blocked without it holding the shared channel. This

allows the deadlock to be initiated by injecting  $m_m$  first in order to block  $m_l$ .<sup>2</sup> Then inject  $m_l$ , followed by  $m_m$  and finally  $m_s$ . Message  $m_m$  blocks  $m_l$  from the time  $m_l$  reaches the cycle until  $m_l$  enters the cycle. In other words, until the end of  $m_m$  traverses the channel that blocks  $m_l$ . This means that  $m_l$  reaches its blocking channel after  $m_s$  acquires this channel, because  $m_l$  enters the cycle after the last flit of  $m_m$  releases that channel and must then travel the length of  $m_l$  to reach its blocking channel. This means the last flit of  $m_m$  must travel from the shared channel to the channel where  $m_l$  is blocked. This is equal to the length of  $m_m$  plus the number of channels used from the shared channel to the cycle. Since  $m_s$  uses fewer channels from the shared channel to the cycle than  $m_m$  does, there is time to inject another message for  $m_m$  and still allow time for  $m_s$  to reach the cycle before  $m_l$  reaches its blocking channel.

If message  $m_l$  uses more channels within the cycle than it uses from the shared channel to the cycle, then blocking  $m_l$  cannot lead to a deadlock. If message  $m_m$  uses fewer channels within the cycle than it uses from the shared channel to the cycle, then blocking  $m_m$  temporarily may lead to a deadlock configuration. In this case,  $m_s$  is injected first to temporarily blocks  $m_m$ . Message  $m_m$  is then injected, followed by  $m_l$  and then  $m_s$ . Obviously, message  $m_s$  reaches the cycle before  $m_l$  reaches its blocking channel, so a deadlock configuration can be formed if  $m_m$  can be blocked long enough to permit  $m_l$  to enter the cycle before  $m_m$  reaches its blocking channel.<sup>2</sup> It can be verified that  $m_s$  will block  $m_m$  long enough only if the number of channels used by  $m_s$  from the shared channel to the channel where  $m_m$  enters the cycle is greater than or equal to the number of channels used by  $m_l$  from the shared channel to the cycle.

A careful examination of the situations considered above shows that all possible message orderings have been considered and a deadlock configuration can arise from the cycle whenever even one of these conditions does not hold.  $\square$

## 6 Conclusion

In this paper, we have constructed an oblivious routing algorithm that is deadlock-free, even though the routing algorithm has cyclic dependencies. This routing algorithm is a counter-example to a long-standing claim by Dally and Seitz [6]. One consequence of this result is that although an acyclic channel dependency graph can be used to guarantee deadlock freedom, the existence of a cycle does not guarantee the routing algorithm can deadlock, even for non-adaptive routing algorithms. The methodology in [15, 17] handles false resource cycles and thus provides a necessary and sufficient condition for oblivious and adaptive routing.

The cyclic dependencies in our counter-example do not lead to a deadlock configuration because the cycle is an unreachable configuration. This false resource cycle arises

<sup>2</sup> An analogous argument can be constructed using an interposed message if required.

when four messages use a shared channel outside the cycle.

We have shown that any such unreachable configuration in an oblivious routing algorithm must have at least three messages that share a channel and that channel must be shared outside of the cycle. Furthermore, restrictions have been placed on the types of oblivious routing algorithms for which an unreachable configuration can be created. Namely, the routing algorithm cannot be suffix-closed.

The conditions that permit a false resource cycle to occur when three messages in a cycle share a channel have been precisely determined. These results could be extended to the case of four messages and beyond. Conditions could also be derived for multiple shared channels. Of course, the number of cases to consider increases with more shared channels or more messages using the shared channel.

A more interesting extension of this work would be to apply these techniques to adaptive routing algorithms and better characterize when false resource cycles can occur with adaptive routing. Because adaptive routing algorithms have more dependencies between channels and a choice of output channels, adaptive routing algorithms are more likely to contain unreachable configurations. For example, Cypher and Gravano [3] present a false resource cycle for a nonminimal routing algorithm for packet-switched networks. Duato has presented a false resource cycle for a nonminimal wormhole routing algorithm. This routing algorithm is reproduced in [17], along with a minimal wormhole routing algorithm that contains a false resource cycle, developed by Schwiebert and Jayasimha.

A better understanding of when unreachable configurations can and cannot arise with adaptive routing should lead to simpler proofs of deadlock freedom for certain classes of adaptive routing algorithms.

## References

- [1] BERMAN, P., GRAVANO, L., PIFARRÉ, G., AND SANZ, J. Adaptive Deadlock- and Livelock-Free Routing With All Minimal Paths in Torus Networks. In *4<sup>th</sup> Annual ACM Symposium on Parallel Algorithms and Architectures* (1992), pp. 3–12.
- [2] BOURA, Y. M., AND DAS, C. R. A Class of Partially Adaptive Routing Algorithms for  $n$ -dimensional Meshes. In *International Conference on Parallel Processing* (1993), vol. III, pp. 175–182.
- [3] CYPHER, R., AND GRAVANO, L. Requirements for Deadlock-Free, Adaptive Packet Routing. *SIAM Journal on Computing* 23, 6 (December 1994), 1266–1274.
- [4] DALLY, W. J. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems* 3, 2 (March 1992), 194–205.
- [5] DALLY, W. J., AND AOKI, H. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems* 4, 4 (April 1993), 466–475.
- [6] DALLY, W. J., AND SEITZ, C. L. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers* C-36, 5 (May 1987), 547–553.
- [7] DUATO, J. On the Design of Deadlock-Free Adaptive Routing Algorithms for Multicomputers: Design Methodologies. In *Parallel Architectures and Languages Europe 91* (1991), vol. I, pp. 390–405.
- [8] DUATO, J. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems* 4, 12 (December 1993), 1320–1331.
- [9] DUATO, J. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. In *International Conference on Parallel Processing* (1994), vol. I, pp. 142–149.
- [10] GLASS, C., AND NI, L. M. The Turn Model for Adaptive Routing. In *19<sup>th</sup> Annual International Symposium on Computer Architecture* (1992), pp. 278–287.
- [11] JAYASIMHA, D. N., MANIVANNAN, D., MAY, J. A., SCHWIEBERT, L., AND HARY, S. L. A Foundation for Designing Deadlock-free Routing Algorithms in Wormhole Networks. In *Symposium on Parallel and Distributed Processing* (1996), pp. 190–197.
- [12] LIN, X., MCKINLEY, P. K., AND NI, L. M. The Message Flow Model for Routing in Wormhole-Routed Networks. *IEEE Transactions on Parallel and Distributed Systems* 6, 7 (July 1995), 755–760.
- [13] NI, L. M., AND MCKINLEY, P. K. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer* 26, 2 (February 1993), 62–76.
- [14] SCHWIEBERT, L., AND JAYASIMHA, D. N. Optimal Fully Adaptive Wormhole Routing for Meshes. In *Supercomputing '93* (1993), pp. 782–791.
- [15] SCHWIEBERT, L., AND JAYASIMHA, D. N. A Universal Proof Technique for Deadlock-Free Routing in Interconnection Networks. In *7<sup>th</sup> Annual ACM Symposium on Parallel Algorithms and Architectures* (1995), pp. 175–184.
- [16] SCHWIEBERT, L., AND JAYASIMHA, D. N. Optimal Fully Adaptive Minimal Wormhole Routing for Meshes. *Journal of Parallel and Distributed Computing* 27, 1 (May 1995), 56–70.
- [17] SCHWIEBERT, L., AND JAYASIMHA, D. N. A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing. *Journal of Parallel and Distributed Computing* 32, 1 (January 1996), 103–117.

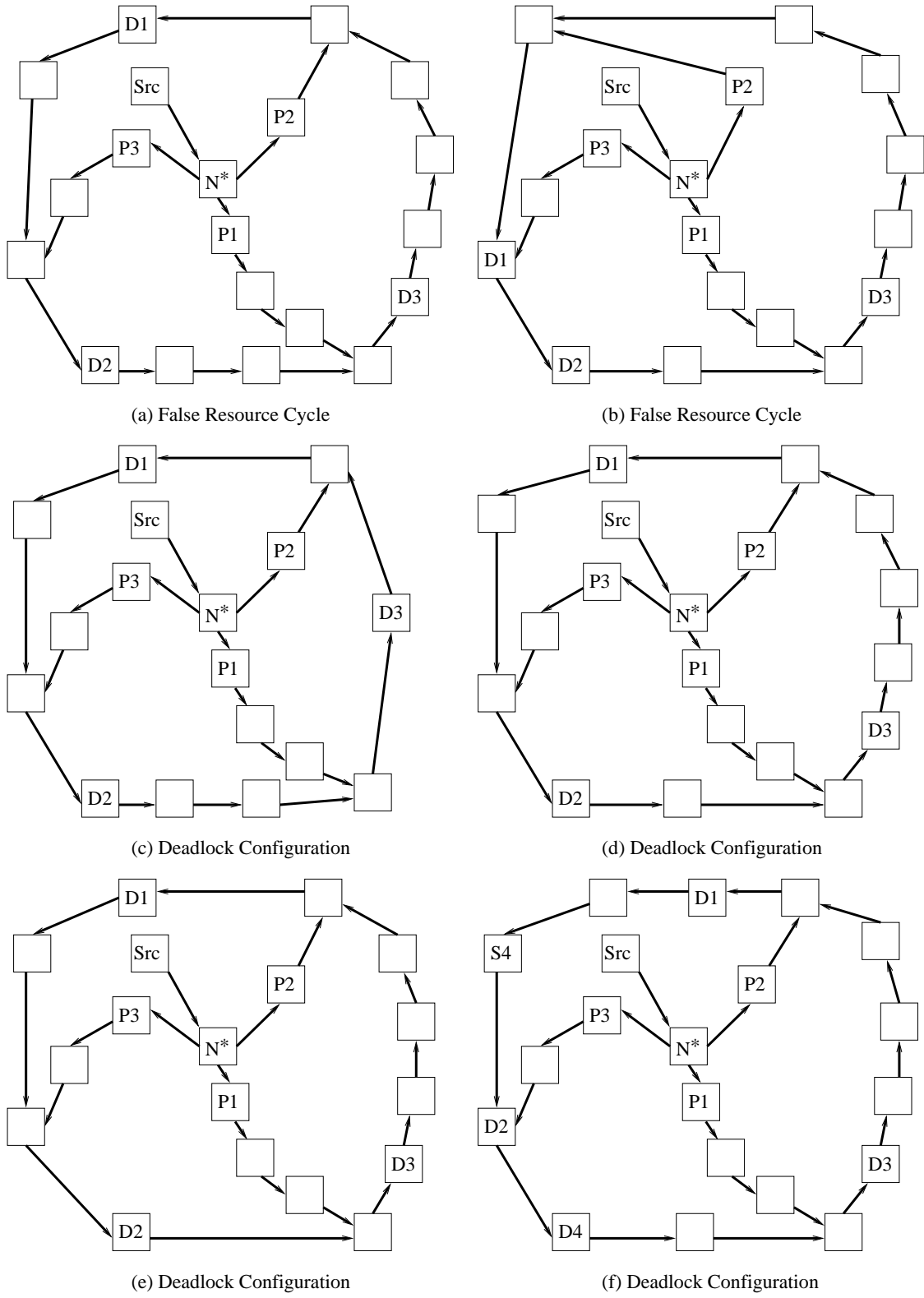


Figure 3: Six Possibilities when a Channel is Shared by Three Messages