

Deadlock-Free Oblivious Wormhole Routing with Cyclic Dependencies

Loren Schwiebert
Wayne State University
Department of Computer Science
5143 Cass Avenue
Detroit, MI 48202-3929
Email: loren@cs.wayne.edu
Phone: (313) 577-5474
Fax: (313) 577-6868

Abstract

A great deal of work has been done recently 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 cyclic configurations to oblivious routing algorithms. An oblivious routing algorithm is thus constructed that is deadlock-free even though there are cycles in the channel dependency graph. The idea of unreachable configurations is then further developed to show various restrictions on when such configurations can exist with oblivious routing algorithms. Finally, the example is generalized to allow the construction of larger networks with unreachable cycles. One benefit of characterizing when unreachable cyclic configurations can occur is that proving deadlock freedom is simplified for networks in which unreachable cycles cannot exist. Another contribution of this work is a first step toward a more formal model of unreachable cycles in wormhole-routed networks.

Keywords: wormhole routing, deadlock freedom, oblivious routing, channel dependency graph, unreachable configuration.

1 Introduction

Wormhole routing [9] is used as a switching technique in distributed memory multiprocessors such as the Cray T3E, as well as, for example, Myrinet [14] switches for connecting networks of workstations. 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. Within the network, each packet is a separate message, so we use the terms message and packet interchangeably. When the header arrives at a router, that router forwards the message header to a neighboring router as soon as an output channel the message can use is available.

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, store-and-forward routing 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 [21] 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. In practice, a continuum exists between virtual cut-through, where the entire message can be buffered, and wormhole routing, where only one or two flits can be buffered. Other options include buffered-wormhole, which uses buffers large enough to contain a few flits, but generally not an entire message. The minimal buffer requirements and limited distance-induced latency account for the popularity of wormhole routing in distributed-memory multiprocessors. See Ni and McKinley [25] 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. Although each channel is released after the entire message traverses that channel, a long message that occupies several channel buffers 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 [7], 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 [9] have shown, however, that 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. The actual path a message follows is typically determined dynamically at each router along the path. Oblivious routing algorithms usually require less complex routers and may have a faster network cycle time [4]. Adaptive routing algorithms can achieve higher utilization for non-uniform traffic and can offer greater fault tolerance.

A routing algorithm is also 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 greater throughput at high traffic

rates and are generally simpler to implement. Nonminimal routing algorithms are useful for fault tolerance. Gaughan and Yalamanchili [18] present a good overview of adaptive routing protocols.

Recent research on routing algorithms for wormhole routing has provided many different techniques for proving deadlock freedom. (Techniques for recovering from deadlocks have also been proposed [1, 24, 26, 27], but are outside the scope of this paper.) In this paper, we build upon these results to show that, even with oblivious routing, cycles in the channel dependency graph do not necessarily imply deadlock. In addition, we show several restrictions on when a deadlock-free oblivious routing algorithm can have cyclic dependencies. The example of a network with an unreachable cycle is then generalized to produce larger networks with the same property. Finally, we discuss the implications of these results on a general methodology for proving deadlock freedom.

Fleury and Fraigniaud [15] independently investigated the problem of whether or not oblivious routing algorithms can have unreachable configurations. Their example, however, requires message lengths of three flits. This requirement, although reasonable in a practical sense, does not satisfy the standard assumption [9, 11] that messages can be of arbitrary length. Furthermore, if shorter messages are used, a deadlock can be formed in their example. Therefore, their example is not consistent with the standard assumptions. Conversely, our example satisfies all of Dally and Seitz's assumptions.

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. Then, in section 6, the example given in section 4 is generalized to produce larger examples and relax one of the assumptions. Finally, in section 7, 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 [9]. 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 $R : C \times N \rightarrow C$, where the input channel, belonging to the set of channels C , and the destination node, belonging to the set of nodes N , define the 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 $R : C \times N \rightarrow \mathcal{P}(C)$, 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 [19], and Boura and Das [3] 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 dependencies 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 [10, 11] 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 [29, 31] 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.* [2] propose a torus routing algorithm that allows cyclic dependencies among the channels.

Dally and Aoki [8] 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 the above-mentioned proof techniques provide only a sufficient condition for deadlock-free adaptive routing. Dally and Seitz’s [9] 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. It was generally believed [10, 11, 23, 30, 32], however, that an acyclic channel dependency graph was required for deadlock-free *oblivious* routing. In fact, Dally and Seitz’s [9] proof applies to only coherent routing algorithms, which are defined later (definition 9). In this paper, we show that some oblivious routing algorithms that are not coherent can be deadlock-free even with cycles in the channel dependency graph.

Finding a necessary and sufficient condition for deadlock-free routing remained an open problem, which was subsequently solved in different ways by several researchers as described below. 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 [5, 6] and used to show that it may be impossible to remove cyclic dependencies from a deadlock-free routing algorithm. An unreachable configuration is derived from a set of channel dependencies that cannot all exist at the same time. Thus, an unreachable configuration cannot be produced by routing messages in the network. Unreachable configurations are 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. Topological properties of the network, including the number and placement of channels, contribute to the occurrence of unreachable configurations. Schwiebert and Jayasimha [30, 32] 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 network. For this reason, they referred to unreachable configurations as *false resource cycles*.

Lin, McKinley, and Ni [22, 23] 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 of deadlock freedom 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. The application of this approach to routing algorithms with unreachable configurations, however, is unclear. Channels are shown to be deadlock-immune by their association with neighboring deadlock-immune channels. The channels in an unreachable configuration form a cycle. Hence, there seems to be no starting point from which to deduce that these are deadlock-immune channels.

Duato [12, 13] 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. For example, fault-tolerant routing algorithms make use of nonminimal routing. If any of the paths defined by the routing algorithm allow a message to visit the same router more than once, that routing algorithm is not coherent [12, 13]. Thus, any routing algorithm that allows backtracking [18] is not coherent. In addition, minimal fully adaptive routing algorithms with straightforward restrictions that are not coherent have been designed [32].

A general necessary and sufficient condition for deadlock-free routing was proved by Schwiebert and Jayasimha [32]. It has none of the restrictions imposed by Duato's proof technique. Furthermore, this methodology can be applied to routing algorithms of the form $R : C \times N \rightarrow \mathcal{P}(C)$ and has been generalized to virtual cut-through and packet switching [30]. 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 generalized by Jayasimha, *et al.* [20] to support routing algorithms for collective communication operations, including multicast. Fleury and Fraigniaud [16, 17] have developed an even more general technique that supports additional definitions of routing functions, such as *source-dependent* and *path-dependent* routing.

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 [32].

3 Assumptions and Definitions

First, a few reasonable assumptions are presented that *increase* the likelihood of deadlock. This makes the task of showing that certain cycles do not lead to deadlock more difficult. This approach is taken to avoid the possibility of designing routing algorithms that seem deadlock-free, but deadlock freedom actually depends on implementation details. 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. Although synchronized clocks are not required, only modest clock skew is permitted. This prevents one message from being forwarded over several channels while another message makes no progress even though its output channel is available.

Note that the preceding assumption on interconnection network behavior is the only assumption that makes deadlock *less* likely. This assumption is used to create small networks exhibiting the desired property of deadlock-free cycles. In section 6, a generalization of these smaller network configurations is used to show that this assumption is, in fact, not required. The result is a general construction that can produce network configurations in which a prolonged delay of the messages in a cycle is possible without deadlock.

Several definitions and additional assumptions are now introduced to facilitate the presentation of the paper. For clarity, the definitions are restricted to oblivious routing. The following are standard assumptions made about wormhole routing and have also appeared in [9, 11].

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. 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. Atomic buffer allocation is used, so 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 $R : C \times N \rightarrow C$ and specifies an output channel based on the input channel 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 $R_A(n_i, n_j)$. R_A defines the path from n_i to n_j . The routing is accomplished by application of the routing function at each router between the source and destination of the message. The routing algorithm may be 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 (atomic buffer allocation). 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 [6].

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 the unavailable output channel held by another message in the set. The deadlock configuration forms a cycle, where the first channel that message m_i uses in the cycle blocks the preceding message in the cycle.

Note: The above definition is specific to oblivious routing. More general definitions of deadlock can be found in [28, 32].

Definition 7 Routing algorithm R_A is *prefix-closed* if the path that R_A specifies from node n_i to node n_j through node n_k ($n_i \neq n_k$) implies that R_A also specifies the partial path from n_i to the first occurrence of n_k on the path from n_i to n_j when n_k is the destination.

Definition 8 Routing algorithm R_A is *suffix-closed* if the path that R_A specifies from node n_i to node n_j through node n_k implies that R_A also specifies 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.

Definition 9 Routing algorithm R_A is *coherent* if R_A is prefix-closed, suffix-closed, and never routes a message through the same node more than once.

4 A Deadlock-Free Cyclic Oblivious Routing Algorithm

Although Duato showed that an acyclic channel dependency graph is not necessary for deadlock-free adaptive routing, no example has been found for oblivious routing. In this section, an 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.

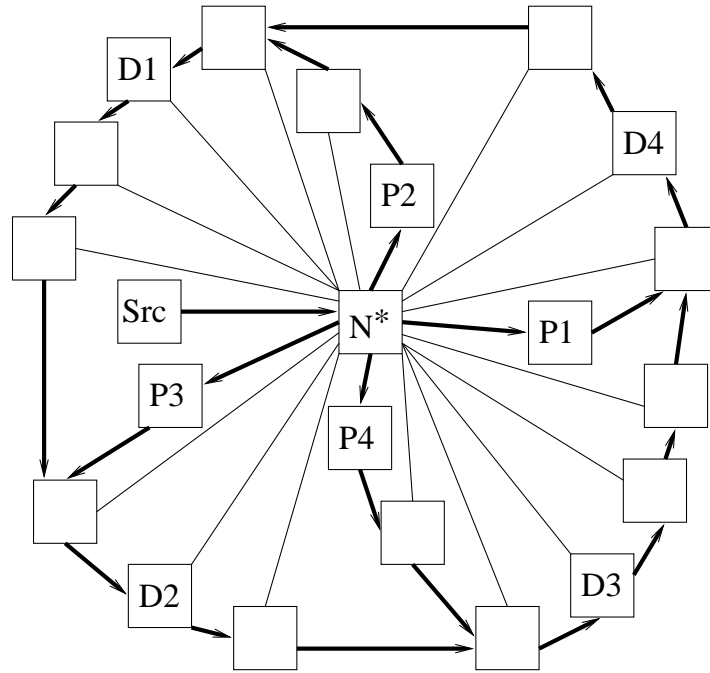


Figure 1: Topology for the Deadlock-Free Routing Algorithm

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 in 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 through the node labeled P_n . For example, the path from Src to $D1$ routes the message from N^* to node $P1$ and then along the highlighted channels 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.

Since each message always has a single path from the source to the destination, this is an oblivious routing algorithm. Only the cycle generated by 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, all four messages must use the channel from Src to N*. Furthermore, each message must be long enough to hold all the channels in the cycle used by that message in order 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. Otherwise, a message cannot block the preceding message in the cycle from reaching its destination. For example, if M3 holds less than three channels, M3 cannot hold the channel that leads to D2, so M2 can reach its destination.

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 (from Src to N*) consecutively to form a cycle, 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 messages M1 and M3 must hold three channels. Because of the intervening nodes, M2 and M4 use three channels from node N* to the cycle, while M1 and M3 use only two channels from N* to the cycle. This introduces a problem when trying to create the cycle. When M1 releases the shared channel between Src and N* (c_s), 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 inject M1 and M3 in time to block M2 and M4.

A second possibility is to form the cycle with more than four messages, by allowing one of the messages to temporarily prevent another message from entering the cycle. This may allow subsequent messages to arrive in time to form a deadlock. Note that messages M1 and M3 use only two channels from c_s to the cycle, and must hold three channels within the cycle to prevent M4 and M2, respectively, from reaching their destinations. Hence, it is not beneficial to temporarily block M1 or M3, because then none of the other messages are able to use c_s and enter the network. Similarly, messages M2 and M4 use only three channels from c_s 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. The messages that would form the cycle comprise an unreachable configuration. Since this is the only cycle 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 recognize 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 [30, 32]. 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 cycle where the shared channels are within the cycle.*

Proof Sketch. By definition 6, a deadlock configuration consists of each message, m_i , in the cycle blocking another message, m_j , at the point where m_i enters the cycle. All the messages in the configuration can use their initial channel in the cycle simultaneously, because no channel sharing is required prior to entering the cycle. Since the shared channel is within the cycle, any message, m_i , that uses it has already blocked some other message, m_j . On the other hand, 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 cyclic configurations.*

Proof. If a routing function has the form $R : N \times N \rightarrow C$, then any cycle can be generated by messages that use channels only 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 oblivious routing algorithm has no unreachable configurations.*

Proof. Because the routing algorithm is suffix-closed, the messages in the cycle do not require the use of any channels outside the cycle. As with the proof of corollary 1, the proof follows immediately from theorem 2. \square

Corollary 3 *A coherent oblivious routing algorithm has no unreachable configurations.*

Proof. The proof follows immediately from corollary 2 and the fact that a coherent routing algorithm is suffix-closed. \square

Restrictions can also be placed on the situations in which sharing a channel outside the cycle leads to an unreachable configuration. First, minimal routing extensions of the Cyclic Dependency routing algorithm are considered. Then, two theorems are proved that apply to both minimal and nonminimal routing. In order to facilitate the explanation of these latter two 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 message(s) in the cycle that could use that channel.

Theorem 3 *Unreachable cyclic configurations with a single shared channel are not possible with minimal oblivious routing if all the messages in the configuration use the shared channel.*

Proof. Label the messages in the cycle so that M_i blocks $M_{i-1} \forall i, 2 \leq i \leq n$ and M_1 blocks M_n . The path that M_i uses from the shared channel to the cycle is P_i . The destination of M_i is D_i . The first channel M_i uses in the cycle is c_i . Note that M_{i-1} also uses c_i to reach D_{i-1} and c_i is the channel where M_i blocks M_{i-1} .

The routing is minimal, so the length of P_{i-1} , denoted $|P_{i-1}|$, and the number of channels from c_{i-1} to c_i cannot be greater than $|P_i|$, the path M_i uses to reach c_i . M_{i-1} must use some channels in the cycle (at least c_{i-1}) to reach c_i and M_i does not, which implies $|P_i| > |P_{i-1}|$. In other words, M_i uses more channels from the shared channel to the cycle than M_{i-1} does. By a similar argument, $|P_{i+1}| > |P_i|$. By analogous arguments, $|P_i| > |P_{i-1}| \forall i, 2 \leq i \leq n$ and $|P_1| > |P_n|$. This is clearly impossible, so such a configuration does not exist. \square

Theorem 3 precludes the possibility of constructing configurations such as the one shown in figure 1 using minimal routing. It may be possible to construct an unreachable configuration with minimal oblivious routing, either by using multiple shared channels or including messages in the cycle that do not use the shared channel. In either case, minimal paths between the nodes outside the cycle and the nodes within the cycle would be required. This would necessitate the use of multiple virtual channels or additional paths of sufficient length between the nodes in the cycle and the nodes outside the cycle. Thus, any examples of unreachable cycles for minimal oblivious routing algorithms, if any exist, are likely to be much more complicated than the Cyclic Dependency routing algorithm shown in figure 1.

Theorem 4 *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 (c_s) is outside the cycle, the two messages can use c_s 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 c_s , any other messages in the cycle can arrive at the appropriate times. 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.

Order the two messages that use c_s such that the message with the longer path from c_s to the cycle is injected first.¹ In figure 2, the first message is labeled M1. Immediately after M1 has traversed c_s , the second message (M2) starts traversing c_s . The distance that the message header of M1 must travel from its current position to the channel in the cycle that blocks M1 (c_2) is equal to the distance from c_s to c_1 . The distance that the message header of M2 has to travel from c_s to c_2 is no more than the distance that M1 still has to travel. This allows *all* the messages in the configuration to enter the cycle before any message reaches the channel where it is blocked. Thus, the cycle can be created and this is a deadlock configuration. Note that this approach works whether both messages use c_s outside the cycle or one message uses c_s within the cycle and the other outside the cycle. \square

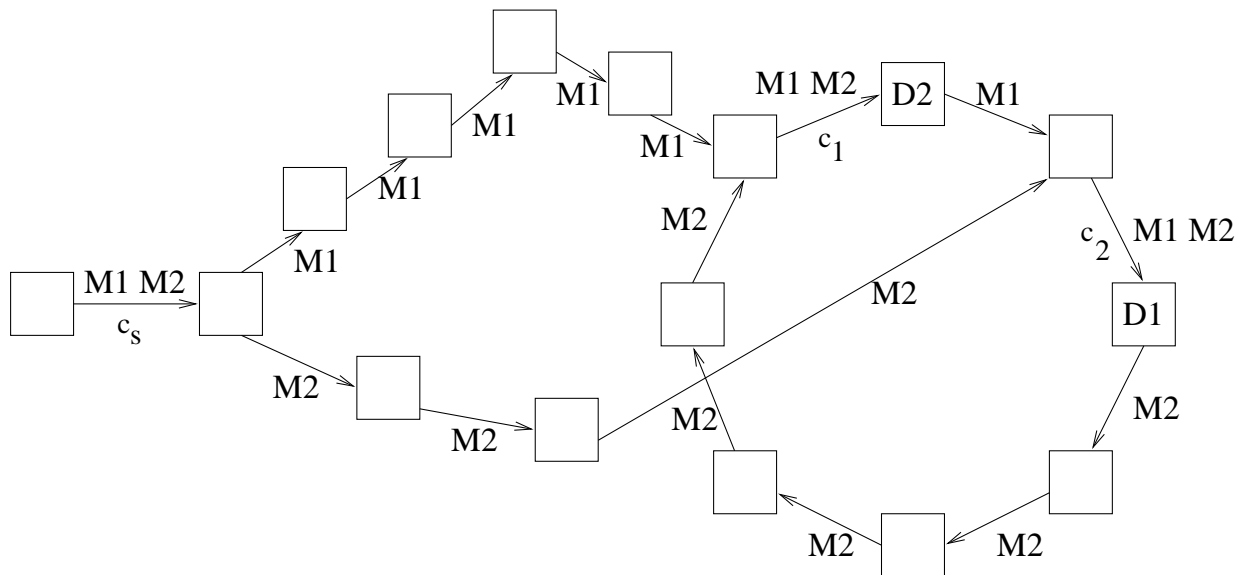


Figure 2: A Deadlock with a Channel Shared by Two Messages

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. Theorem 2 proves that with oblivious routing, a false resource cycle is not possible if the shared channel is within the cycle. Theorem 3 proves that unreachable configurations such as that shown in figure 1 are not possible with minimal routing. Finally, theorem 4 proves that a false resource cycle is impossible if 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 the cycle share a channel. Necessity and sufficiency are proved in theorem 5.

In presenting these conditions, the messages are labeled by the number of channels used between the shared channel (c_s) and the cycle, so m_1 refers to the message that uses the most channels

¹If the distance is the same for both messages, choose either.

between c_s and the first channel that message uses in the cycle. Similarly, the message using the fewest channels between c_s and the cycle is labeled m_3 and the third message using c_s is labeled m_2 . As shown in figure 3, the first channel m_n uses in the cycle is labeled c_n , where $1 \leq n \leq 3$. Note that this does not necessarily mean that m_1 uses the most channels within the cycle, only that this message has the longest path from c_s to the cycle.

1. Relative to the cycle, the order of the messages using c_s is such that m_1 is followed by m_3 . Note that there could be other messages between these two messages, but m_2 is not between m_1 and m_3 .
2. All three messages use c_s outside of the cycle.
3. All three messages use a different number of channels from c_s to the cycle.
4. Message m_1 uses more channels within the cycle than it uses from c_s to c_1 .
5. If the message in the cycle that *immediately* precedes message m_3 in the cycle does not use c_s , then message m_3 uses more channels within the cycle than it uses from c_s to c_3 .
6. Either message m_2 uses more channels within the cycle than it uses from c_s to c_2 or m_3 immediately precedes m_2 in the cycle and m_3 uses fewer channels from c_s to c_2 than m_1 uses from c_s to c_1 .
7. The number of channels used by m_1 from c_s to c_1 plus the number of channels used by other messages in the cycle between m_1 and m_3 is less than the number of channels used in the cycle by m_2 plus the number of channels used by m_3 from c_s to c_3 .
8. The number of channels used by m_3 from c_s to c_3 plus the number of channels used by other messages in the cycle between m_3 and m_2 is less than the number of channels used by m_2 from c_s to c_2 .

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 to form the cycle have been removed for clarity. 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_2 can be blocked between the shared channel and the cycle, because message m_3 is too short to block m_2 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_3 is too long. The deadlock is formed by injecting messages in the order m_3, m_2, m_1, m_3 . 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 S4 to D4 and does not use the shared channel. This configuration does not satisfy conditions 6 and 8, so a deadlock can be generated.

Theorem 5 *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. If only one of the messages (m_1) uses the shared channel (c_s) before using any channels in the cycle, m_1 can acquire c_s at the same time that m_2 and m_3 acquire c_2 and c_3 , respectively. Message m_1 can be made long enough to hold c_s , so m_2 and m_3 are blocked. Any other messages do not use c_s , so each of these messages can enter the cycle in time to block the preceding message. Thus, a deadlock configuration has been formed.

Likewise, if only two messages (m_1 and m_2) use c_s before any channels in the cycle, then the approach taken in theorem 4 for ordering the messages can be used to produce a deadlock configuration. Message m_3 can be started so that m_3 arrives at c_s immediately after m_2 starts using c_s and m_2 can be made long enough to hold c_s . Message m_2 can acquire c_s immediately after m_1 releases c_s and this is before m_1 reaches the channel where it is blocked. Hence, an unreachable configuration is not possible unless all three messages use c_s prior to using a channel in the cycle.

Assume c_s is used by all three messages prior to using a channel in the cycle. These messages must use c_s in consecutive order. A deadlock occurs if all messages 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 c_s , all 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 c_s , followed by zero or more messages not using c_s , followed by another message using c_s again followed by zero or more messages not using c_s , and then again followed by a message that uses c_s and then possibly additional messages that do not use c_s . In other words, each pairing of messages that use the shared channel could have other messages between them within the cycle.

If one of the messages in the cycle, m_i , uses at least as many channels from c_s to c_i as m_i uses within the cycle, then m_i can be blocked outside of the cycle after using c_s . If the preceding message in the cycle does not use c_s , then that message can block m_i indefinitely by creating a long enough message. This effectively reduces the cycle to a case of only two messages using c_s and a deadlock configuration can be constructed as described in the proof of theorem 4.

Order the three messages that use c_s such that m_1 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_1 followed by m_2 and then m_3 , a deadlock configuration can be generated when these messages use c_s consecutively in this order. The construction is similar to the one used in the proof of theorem 4. Each message reaches the cycle in time to block the preceding message.

It is easily verified that such an ordering is possible unless m_1 is followed by m_3 and all three messages use a *different* number of channels from c_s to the cycle. In this case, m_3 could bypass m_2 in the cycle if m_3 uses c_s before m_2 does. If the other messages in the cycle (if any) interposed between these two messages (m_3 and m_2) use at least as many channels as the difference between the number of channels used from c_s to c_2 by m_2 and from c_s to c_3 by m_3 , then m_3 is successfully blocked and a deadlock is constructed.

Otherwise, it may be possible to generate a deadlock from the cycle by injecting m_2 before m_3 , but after m_1 . In this case, m_1 could bypass m_3 . This does not occur, however, if m_3 reaches the cycle in time. Message m_3 can acquire c_s as soon as m_2 is finished with c_s . At this time, m_1 has traveled as many channels as m_2 uses in the cycle, because that is the minimum length of m_2 . If m_1 still needs to travel at least as far to reach its blocking channel as m_3 needs to travel to reach c_3 , then the deadlock configuration can be formed. Of course, messages interposed between m_1 and m_3 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_1

is temporarily blocked outside the cycle so that m_2 and m_3 can reach the cycle in time. This temporary blocking is beneficial only when m_1 uses no more channels within the cycle than m_1 uses from c_s to c_1 , because then m_1 can be blocked without holding c_s . This allows the deadlock to be initiated by injecting m_2 first in order to block m_1 . (Message m_1 is blocked at c_3 , the first channel m_3 uses.²) Then inject m_1 , followed by m_2 , and finally m_3 . Message m_2 blocks m_1 from the time m_1 reaches the cycle until m_1 enters the cycle (acquires c_1). In other words, until the end of m_2 traverses c_1 . This means that m_1 reaches its blocking channel (c_3) after m_3 acquires c_3 , because m_1 enters the cycle after the last flit of m_2 releases c_1 and must then travel the length of m_1 to reach c_3 . This means the last flit of m_2 must travel from c_s to c_1 . This is equal to the length of m_2 plus the number of channels used from c_s to c_2 . Since m_3 uses fewer channels from c_s to the cycle than m_2 does, there is time to inject another message for m_2 and still allow time for m_3 to reach c_3 before m_1 does.

If message m_1 uses more channels within the cycle than m_1 uses from c_s to c_1 , then blocking m_1 cannot contribute to forming a deadlock. If message m_2 uses fewer channels within the cycle than m_2 uses from c_s to c_2 , then blocking m_2 temporarily may lead to a deadlock configuration. In this case, m_3 is injected first to temporarily block m_2 . Message m_2 is then injected, followed by m_1 and then m_3 . Obviously, message m_3 reaches c_3 before m_1 does, so a deadlock configuration can be formed if m_2 can be blocked long enough to permit m_1 to acquire c_1 before m_2 reaches c_1 .² It can be verified that m_3 will block m_2 long enough only if the number of channels used by m_3 from c_s to c_2 is greater than or equal to the number of channels used by m_1 from c_s to c_1 .

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

6 Generalization

The example presented in figure 1 would be a deadlock configuration if both M1 and M3 were delayed one or more clock cycles. Since both messages are still moving toward their destinations, waiting for the processor at each destination to consume that message would not provide this needed delay. However, it may seem unreasonable to expect all the routers in the network to operate in such a tightly synchronous fashion with minimal clock skew. In this section, we show that configurations which can tolerate additional delay can be constructed using larger networks. In other words, substantial clock skew among the routers does not prevent the creation of unreachable cycles.

The example uses two features of the network in figure 1 to avoid deadlock. The first relevant feature is that each message uses more channels in the cycle than from the shared channel to the cycle. The result is that blocking a message before it enters the cycle prevents the other messages from using the shared channel. Therefore, blocking a message outside the cycle does not contribute to forming a deadlock. The second feature is that the odd-numbered messages use fewer channels than the even-numbered messages. Regardless of the order in which the messages use the shared channel, it is impossible for the even-numbered messages to block the corresponding odd-numbered message every time. These two features can easily be generalized into larger networks

²An analogous argument can be constructed using an interposed message if required.

with unreachable cycles.

In order to simplify the generalization, three parameters are defined for each message:

- s_i – the distance (number of channels) that message m_i must travel from the shared channel to the cycle.
- f_i – the length of message m_i in flits.
- d_i – the distance (number of channels) message m_i must travel from its entry into the cycle until m_i reaches its destination.

As discussed previously, a deadlock configuration is most likely when the routers have one-flit buffers and the message length is just long enough to hold the channels the messages in the cycle. These two factors allow subsequent messages to acquire the shared channel as soon as possible. For this reason, $\forall i, f_i = d_i$ in the following discussion.

Consider a network similar to figure 1, except $s_1 = s_3 = n - 1$ and $s_2 = s_4 = 2n - 1$. In addition, $d_1 = d_3 = n$ and $d_2 = d_4 = 2n$. (By comparison, in figure 1, $s_1 = s_3 = 2$, $s_2 = s_4 = 3$, $d_1 = d_3 = 3$, and $d_2 = d_4 = 4$.) First, recall that since all messages use more channels in the cycle than between the shared channel and the cycle, blocking a message from entering the cycle also blocks the shared channel. Due to symmetry in the network, there are only two cases to consider:

Case (1): Message m_1 is injected first.

Assume that message m_2 is injected immediately after m_1 . When m_2 acquires the shared channel, m_1 is $n - 1$ hops from its destination and m_2 is $2n - 1$ hops from blocking m_1 . This means that m_1 must be delayed in the network for at least n clock cycles even though the output channel is always free when m_1 arrives at a router. After m_2 releases the shared channel, m_3 can use the shared channel and arrive in time to block m_2 . However, m_3 must also be delayed at least n clock cycles in order for m_4 to block m_3 . If a message other than m_2 is injected immediately after m_1 , then m_1 will have to be arbitrarily delayed even longer or m_1 will arrive at its destination.

Case (2): Message m_2 is injected first.

Assume that message m_3 is injected immediately after m_2 . As in the previous case, m_3 arrives in time to block m_2 , but m_3 must be delayed at least n extra clock cycles in order for m_4 to block m_3 . It is also possible to order the messages – m_2, m_1, m_3 . In this instance, m_3 still arrives in time to block m_2 . Message m_3 must still be delayed at least n clock cycles, however, so that m_4 has time to reach the cycle. If m_4 is injected immediately after m_2 then m_2 must be delayed at least n clock cycles in order for m_3 to enter the cycle and block m_2 .

In both cases, the network configuration requires at least one message in the cycle to be delayed at least n clock cycles in order to form a deadlock. Since n can be made arbitrarily large, a network configuration can be constructed requiring any amount of extra delay before deadlock can occur. Thus, the routers do not need to operate in a tightly synchronous manner in order to construct a deadlock-free cycle.

7 Conclusion

In this paper, we have constructed an oblivious routing algorithm that is deadlock-free, even though the routing algorithm has cyclic dependencies. 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 nonadaptive routing algorithms. The methodologies proposed by Schwiebert and Jayasimha [30, 32] and by Fleury and Fraigniaud [16, 17] handle false resource cycles, and thus provide a necessary and sufficient condition for both oblivious and adaptive routing.

The cyclic dependencies in our example do not lead to a deadlock configuration because the cycle is an unreachable configuration. This false resource cycle arises when four messages use a shared channel outside the cycle. The example has been generalized to larger networks, thereby allowing construction of configurations where lengthy arbitrary delay of messages does not permit deadlock.

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. The routing algorithm cannot be suffix-closed, which includes the class of coherent routing algorithms. In addition, minimal oblivious routing algorithms cannot have unreachable configurations if a single shared channel is used in the cycle and there are no additional messages in the deadlock configuration.

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 when there are multiple shared channels for the same cycle. Of course, the number of cases to consider increases with more shared channels or more messages using the shared channel.

Because adaptive routing algorithms have a choice of output channels and more dependencies between channels, adaptive routing algorithms are more likely to contain unreachable configurations. For example, Cypher and Gravano [6] 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 that could be embedded in a mesh or torus. This routing algorithm is reproduced in [32], along with a minimal wormhole routing algorithm for tori that contains a false resource cycle, developed by Schwiebert and Jayasimha.

Therefore, a more interesting extension of this work would be to apply these techniques to better characterize when false resource cycles can occur with adaptive routing. Similar techniques that distinguish classes of adaptive algorithms that permit cyclic unreachable configurations from those that do not still need to be developed. Duato [12, 13] has partially addressed this issue by proving that coherent routing algorithms of the form $R : N \times N \rightarrow \mathcal{P}(C)$ cannot have unreachable cycles. 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.

Acknowledgments

The author thanks Melanie Fulgham and D. N. Jayasimha for suggestions that have been incorporated into section 6. The author appreciates the detailed comments provided by the anonymous reviewers, which resulted in many improvements. The author also thanks the attendees of SPAA

'97 for useful feedback that has improved this paper.

References

- [1] K. V. Anjan and T. M. Pinkston. An Efficient, Fully Adaptive Deadlock Recovery Scheme: Disha. In *22nd Annual International Symposium on Computer Architecture*, pages 201–210, June 1995.
- [2] P. Berman, L. Gravano, G. Pifarré, and J. Sanz. Adaptive Deadlock- and Livelock-Free Routing With All Minimal Paths in Torus Networks. In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 3–12, June 1992.
- [3] 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, August 1993.
- [4] A. A. Chien. A Cost and Speed Model for k-ary n-cube Wormhole Routers. In *Proceedings of Hot Interconnects '93*, August 1993.
- [5] R. Cypher and L. Gravano. Requirements for Deadlock-Free, Adaptive Packet Routing. In *11th ACM Symposium on Principles of Distributed Computing*, pages 25–33, August 1992.
- [6] R. Cypher and L. Gravano. Requirements for Deadlock-Free, Adaptive Packet Routing. *SIAM Journal on Computing*, 23(6):1266–1274, December 1994.
- [7] W. J. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.
- [8] 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.
- [9] 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.
- [10] J. Duato. On the Design of Deadlock-Free Adaptive Routing Algorithms for Multicomputers: Design Methodologies. In *Parallel Architectures and Languages Europe 91*, pages 390–405, June 1991.
- [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, August 1994.

- [13] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, October 1995.
- [14] N. J. Boden *et al.* Myrinet: A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–36, February 1995.
- [15] E. Fleury and P. Fraigniaud. Deadlocks in adaptive wormhole routing. Technical Report RR94-09, Laboratoire de l’Informatique du Parallélisme, LIP, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France, March 1994.
- [16] E. Fleury and P. Fraigniaud. Deadlock Avoidance in Wormhole-Routed Networks. In *10th International Conference on Parallel and Distributed Computing Systems*, pages 378–384, October 1997.
- [17] E. Fleury and P. Fraigniaud. A General Theory for Deadlock Avoidance in Wormhole-Routed Networks. *IEEE Transactions on Parallel and Distributed Systems*, 9(7):626–638, July 1998.
- [18] P. T. Gaughan and S. Yalamanchili. Adaptive Routing Protocols for Hypercube Interconnection Networks. *IEEE Computer*, 26(5):12–23, May 1993.
- [19] C. Glass and L. M. Ni. The Turn Model for Adaptive Routing. In *19th Annual International Symposium on Computer Architecture*, pages 278–287, May 1992.
- [20] D. N. Jayasimha, D. Manivannan, J. A. May, L. Schwiebert, and S. L. Hary. A Foundation for Designing Deadlock-free Routing Algorithms in Wormhole Networks. In *Symposium on Parallel and Distributed Processing*, pages 190–197, October 1996.
- [21] P. Kermani and L. Kleinrock. Virtual Cut-Through: A New Computer Communication Switching Technique. *Computer Networks*, 3(4):267–286, September 1979.
- [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, August 1993.
- [23] X. Lin, P. K. McKinley, and L. M. Ni. The Message Flow Model for Routing in Wormhole-Routed Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(7):755–760, July 1995.
- [24] J. M. Martínez, P. López, J. Duato, and T. M. Pinkston. Software-Based Deadlock Recovery Technique for True Fully Adaptive Routing in Wormhole Networks. In *International Conference on Parallel Processing*, pages 182–189, August 1997.
- [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] F. Petrini and M. Vanneschi. Performance Analysis of Minimal Adaptive Wormhole Routing with Time-Dependent Deadlock Recovery. In *International Parallel Processing Symposium*, pages 589–595, April 1997.

- [27] T. M. Pinkston. Flexible and Efficient Routing Based on Progressive Deadlock Recovery. *IEEE Transactions on Computers*, 48(7):649–669, July 1999.
- [28] T. M. Pinkston and S. Warnakulasuriya. Characterization of Deadlocks in k -ary n -Cube Networks. *IEEE Transactions on Parallel and Distributed Systems*, 10(9):904–921, September 1999.
- [29] L. Schwiebert and D. N. Jayasimha. Optimal Fully Adaptive Wormhole Routing for Meshes. In *Supercomputing '93*, pages 782–791, November 1993.
- [30] L. Schwiebert and D. N. Jayasimha. A Universal Proof Technique for Deadlock-Free Routing in Interconnection Networks. In *7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 175–184, July 1995.
- [31] L. Schwiebert and D. N. Jayasimha. Optimal Fully Adaptive Minimal Wormhole Routing for Meshes. *Journal of Parallel and Distributed Computing*, 27(1):56–70, May 1995.
- [32] L. Schwiebert and D. N. Jayasimha. A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing. *Journal of Parallel and Distributed Computing*, 32(1):103–117, January 1996.

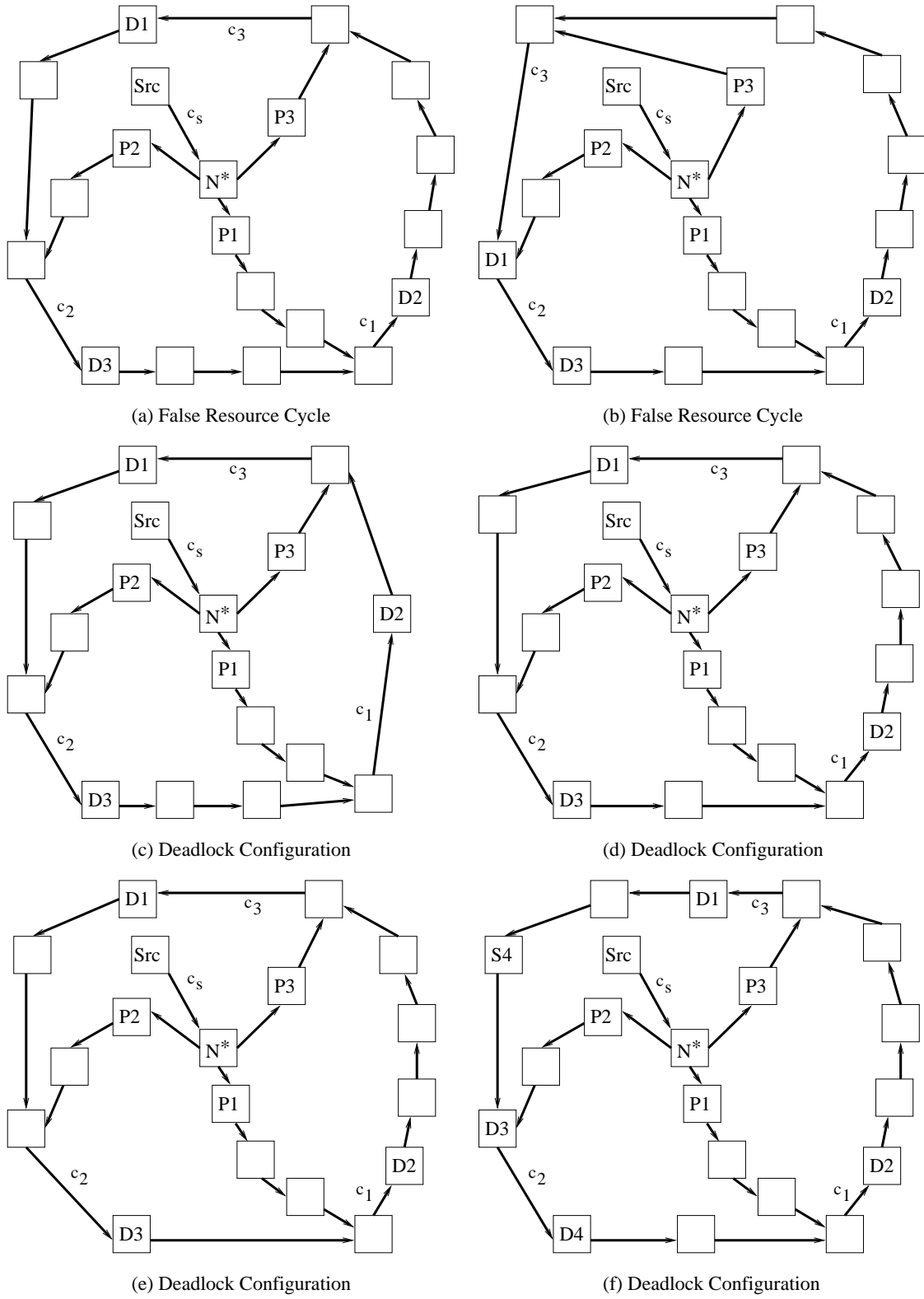


Figure 3: Six Possibilities when a Channel is Shared by Three Messages